VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

SYSTÉM PRO ZPRACOVÁNÍ DAT Z REGULÁTORU HAWK FIRMY HONEYWELL

HAWK CONTROLLER DATA PROCESSING SYSTEM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE Bc. Jiří Dostál

AUTHOR

VEDOUCÍ PRÁCE Ing. Václav Kaczmarczyk, Ph.D.

SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Jiří Dostál ***ID:*** 145347
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Systém pro zpracování dat z regulátoru HAWK firmy Honeywell

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte a realizujte systém pro vyčítání, archivaci a jednoduchou prezentaci dat získaných ze systému HAWK firmy Honeywell. Navrhněte jednoduchou demonstrační aplikaci využívající tento systém. Navržené řešení podrobně zdokumentujte.

1. Popište možnosti a způsob vývoje se systémem HAWK a prostředím COACH firmy Honeywell.
2. Nastudujte a popište hlavní principy Niagara Frameworku.
3. Navrhněte a v jazyce JAVA s použitím Niagara Frameworku implementujte tyto komponenty:
a) Komponenta pro sběr dat z datových bodů.
b) Služba pro odesílání získaných dat do nadřazeného systému.
4. Navrhněte a implementujte PC aplikaci pro příjem dat ze systému HAWK a jejich následné zpracování (ukládání do MySQL DB, případně další funkcionalita).
5. Otestujte funkčnost vašeho řešení.

DOPORUČENÁ LITERATURA:

Honeywell: COACH AX User Guide,

Albahari et al.: c# 6.0 in a Nutshell

Termín zadání: 6.2.2017 ***Termín odevzdání:*** 15.5.2017

Vedoucí práce: Ing. Václav Kaczmarczyk, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

UPOZORNĚNÍ:
Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

Abstrakt

Tato práce se zabývá vývojem programových komponent pro sběr sémanticky označených dat z regulátoru Hawk firmy Honeywell. Jsou popsány základní principy a možnosti vývoje v Niagara Frameworku, na kterém je řídicí systém Hawku založen. Dále je popsáno konkrétní řešení navržených komponent a použité externí aplikace pro ukládání dat do databáze.

Klíčová slova

Niagara Framework, Entity Framework, Project Haystack, Hawk, Coach, Centraline, MySQL, Baja, JACE, sémantický model, integrovaný řídicí systém budovy

Abstract

This thesis deals with developing program components for collection of semantically labeled data from Honeywell's Hawk controller. The basic principles and capabilities of development using Niagara Framework, on which Hawk is based, are explained. Lastly, the specific components and external database application is described.

Keywords

Niagara Framework, Entity Framework, Project Haystack, Hawk, Coach, Centraline, MySQL, Baja, JACE, semantic model, integrated building automation system

Bibliografická citace:

DOSTÁL, J. *Systém pro zpracování dat z regulátoru HAWK firmy Honeywell*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2017. 59s. Vedoucí práce: Ing. Václav Kaczmarczyk, Ph.D.

Prohlášení

„Prohlašuji, že svou závěrečnou práci na téma Systém pro zpracování dat z regulátoru HAWK firmy Honeywell jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne **15. května 2017** ....

podpis autora

Děkuji vedoucímu diplomové práce Ing. Václavu Kaczmarczykovi, Ph.D.. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne **15. května 2017** …………………………
podpis autora(-ky)

Obsah

1 Úvod ... 8
2 Řídicí systémy budov ... 9
2.1 Historie a vývoj ... 9
2.2 Technická zařízení v automatizaci budov ... 9
2.3 Funkce řídicího systému budovy ... 10
2.4 Hierarchický model řídicího systému budovy ... 12
3 Integrovaný řídicí systém budov CentraLine ... 14
3.1 Úvod, platforma, stanice ... 14
3.2 Regulátor HAWK ... 15
3.3 Grafická centrála ARENA AX ... 17
3.4 Vývojový software COACH AX ... 18
4 Niagara framework ... 20
4.1 Historie a úvod ... 20
4.2 Architektura frameworku ... 21
4.3 Niagara objektový model ... 22
4.4 Stanice ... 24
4.5 Celkový přehled, protokoly ... 26
4.6 Niagara Framework z pohledu uživatele ... 27
5 Tvorba APLIKACE v Niagara frameworku ... 29
5.1 Úvod ... 29
5.2 Návrh řešení ... 29
5.3 Project Haystack ... 30
5.4 Nastavení vývojového prostředí ... 33
5.5 Vytvoření modulu ... 34
5.6 Vytvoření služby ... 37
5.7 Vytvoření rozšíření pro komponenty ... 39
5.8 HaystackSlot a vytvoření pohledu ... 40
6 Aplikace pro příjem dat ... 42
6.1 Úvod ... 42
6.2 Entity Framework ... 42
6.3 Třída DBContext ... 44
6.4 Návrh databáze ... 44
6.5 Vytvoření aplikace ... 46
7 Závěr ... 48
8 Použitá literatura ... 49
Seznam příloh ... 52
Příloha 1 - postup při ověření funkčnosti řešení ... 53

1 ÚVOD

Podle informací z literatury [21], dvacet až čtyřicet procent celkových výdajů za energie v Evropě představují výdaje za energetickou spotřebu budov. Typicky 50% - 60% této spotřebované energie je využito na systémy vytápění, větrání, a klimatizace (používaná souhrnná anglická zkratka HVAC - heating, ventilation, air conditioning), následuje osvětlení a ostatní zařízení.

V běžném životě se můžeme setkat s pojmem *inteligentní budova*. Pod tímto pojmem si lze představit budovu, která vyhovuje požadavkům na snížení provozních nákladů budovy a zároveň požadavkům na udržení, případně zvýšení uživatelského komfortu. Tyto požadavky plní řídicí systém budovy, který vhodně *propojuje a řídí* jednotlivé technické zařízení budov (jako vytápění, vzduchotechnika, osvětlení, atd).

Hlavním směrem v řídicích systémech budov je integrace. Z historického hlediska se funkční systémy budovy vyvíjely (a vyvíjejí) odděleně. To s sebou přináší použití různých komunikačních protokolů pro různé systémy, a od různých výrobců. Tento problém řeší například v této práci použitá integrační platforma Hawk řídicího systému CentraLineAX společnosti Honeywell, která je postavena na Niagara frameworku.

Cílem práce je vytvoření komponent v rámci Niagara frameworku, které umožňují sběr dat z vybraných datových bodů a jejich ukládání do mySQL databáze. Práce se dále zabývá tím, jak lze získaná data značit a přiřadit jim význam z hlediska řídicího celku. Pro značení dat je použito sémantického modelu vytvořeného opensourcovou iniciativou Project Haystack [15].

2 ŘÍDICÍ SYSTÉMY BUDOV

2.1 Historie a vývoj

V minulosti byly k řízení používány především pneumatické a elektro-mechanické prvky. S nástupem počítačů v řízení se situace změnila, základním prvkem řízení se stává počítač. Rychlý vývoj řídicích systémů nastupuje v 70-80 letech 20. století, poháněný rozvojem počítačové techniky a poptávce po šetření energií po ropné krizi v roce 1973 [16].

První systémy se objevily na trhu koncem 60. let 20. století. Jednalo se centralizované systémy, základem byl řídicí počítač a ovládací panel. Zpravidla byl řízen pouze jeden ze subsystémů budovy. Řízené vstupní a výstupní zařízení byly připojeny přes multiplexery adresovatelné z ovládacího panelu. Systém obsahoval základní funkce - časové programy pro jednotlivá zařízení, automatický reset výstupů, výstražná hlášení. Počítačová technika byla velmi drahá a z dnešního pohledu byla obsluha komplikovaná - např. program byl nahráván přes děrnou pásku, neexistovaly periferní zařízení klávesnice, myš.

Základní nevýhodou tohoto způsobu bylo, že všechny připojené zařízení byly neměnné a napevno "zadrátované", proto postupně docházelo k decentralizaci.V 70. letech začalo být používáno nové zařízení - panel pro sběr dat (DGP - data gathering panel). Funkce DGP je zajistit komunikaci mezi řídicím počítačem a IO zařízeními. DGP tak umožnil částečně decentralizovat řídicí systém. V 80. letech a později se začíná řízení decentralizovat.

Vzniká distribuovaná síť DDC (Direct digital control) modulů, autonomních řídicích jednotek nezávislých na centrálním počítači, a schopných komunikace s nadřazeným centrem. Současně vzniká i problém nekompatibility soukromých komunikačních protokolů od různých výrobců. Reakcí je snaha standardizovat komunikaci, vznikají protokoly BacNet, LonWorks, KNX.

Současný vývoj směřuje k integraci jednotlivých systémů budovy do jednoho celku. Mezi vlastnosti moderního řídícího systému budovy by měl patřit přístup k systému a zařízením přes internet [16, 17, 18].

2.2 Technická zařízení v automatizaci budov

Jak již bylo v úvodu napsáno, řídicí systém budovy propojuje a řídí jednotlivé technické zařízení budov (TZB). Pojem TZB pod sebe zahrnuje zařízení nezbytné pro provoz budov. Mezi hlavní patří zařízení, které zajišťují vnitřní prostředí budovy (dodávka tepla, vzduchu, vody, chlazení, elektrické energie, osvětlení atd.). Přehled běžně integrovaných TZB do řídicích systémů je v následující tabulce :

Tabulka 1: Technické zařízení budov v automatizaci budov [19]

přístroje (TZB)	TZB běžně integrovaná do automatizace budov	TZB dodatečně integrovaná do automatizace budov	TZB s DDC moduly nebo řízením a regulací prostřednictvím komponent techniky systémů budov
Vytápění	x		x
Ventilace	x		x
Chlazení, Klimatizace	x		x
Dodávky el. energie	x		
Řízení osvětlení	x		x
Řízení žaluzií/zastínění	x		x
Sanitární přístroje	x		
Detekce a ohlašování požáru	x		
Detekce a ohlašování neoprávněného vstupu		x	
Kontrola vstupu do budovy		x	
kamerový systém		x	
Technika sítě, zasíťování		x	
Multimédia		x	
Výtahy		x	
Telefonní přístroje		x	
Údržba		x	
Systém vyúčtování a úhrad		x	

2.3 Funkce řídicího systému budovy

2.3.1 Řídicí funkce

Řídicí funkce zajišťuje automatizaci systémů TZB tak, aby byly správně poskytovány základní služby na lokální úrovni - například v místnosti. Úkolem je aby výstup co nejpřesněji sledoval žádanou hodnotu (z hlediska aplikace se většinou jedná o konstantu - vyžadujeme stálou teplotu v místnosti, stálou vlhkost, stálý obsah CO2). Toho lze dosáhnout použitím zpětnovazebního řízení.

2.3.2 Funkce správy energií

Úspory na spotřebované energii jsou jedním z hlavních důvodů pro využití Řídicího systému budovy. Úspor je dosaženo vhodným řízením budovy, například

jedním ze způsobů je vypínat zařízení a provozovat je pouze podle časového plánu budovy. Toto je základní princip, který může být dále rozšířen. Pokud jsou brány v potaz například aktuální venkovní tepelné podmínky, může být odhadnut čas pro optimální zapnutí a optimální vypnutí tak, aby byla dodržena tepelná pohoda.

Druhý způsob spočívá v provozování systémů v energeticky optimálních podmínkách. Jako příklad slouží změna požadované teploty přívodu topného okruhu v závislosti na venkovních tepelných podmínkách. Posledním způsobem je spolupráce systémů, například zastavení chlazení a vytápění místnosti, pokud jsou otevřeny okna.

2.3.3 Monitorovací funkce

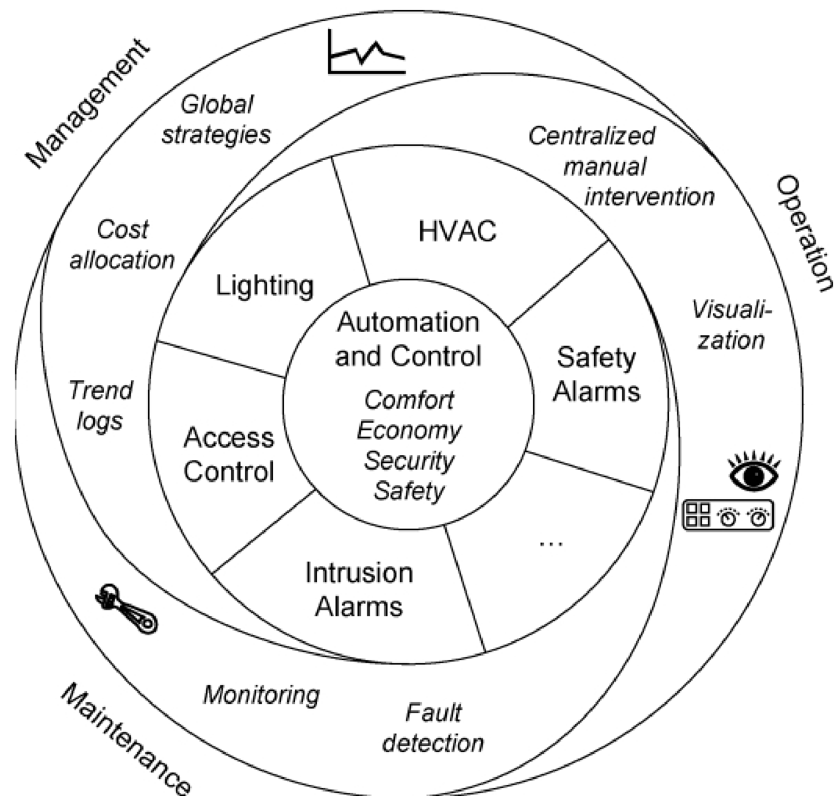
Řídicí systém budovy poskytuje všechny dostupné informace o budově v uceleném jednotném interaktivním grafickém prostředí pro obsluhu operátorské stanice. Obsluha má možnost ručně měnit parametry regulace, sledovat stavy zařízení, je informována o alarmních hlášeních a poruchách. Jsou shromažďovány časové závislosti vybraných veličin (trendy).

Pod monitorovací funkci lze zařadit i plánování údržby a detekci poruch. Zařízení většinou obsahují funkci hlášení poruchy. Údržba může být prováděna na základě doby provozu zařízení, popřípadě dalších měření (např. snímače diferenčního tlaku pro zjištění stavu filtrů).

2.3.4 Bezpečnostní funkce

Bezpečnostní funkce řídicího systému budovy zahrnují přístupová práva v budově, zabezpečení budovy proti cizímu vniknutí, monitorování prostorů budovy videokamerami, zabezpečení proti požáru. V případě požáru dojde automaticky například k řízení klapek a odtahových ventilátorů tak, aby nedocházelo k šíření kouře, odblokují se únikové trasy a dojde k světelnému vyznačení tras.

Celkové shrnutí funkcí řídicího systému budovy je na obrázku 1. Tento obrázek znázorňuje výše popsané funkce řídicího systému budovy a jejich návaznost na jednotlivé TZB. V jeho středu se nachází automatizace a řízení jako řídicí funkce systému.



Obrázek 1 - Funkce řídicího systému budovy [20]

2.4 Hierarchický model řídicího systému budovy

Model obecného řídicího systému lze podle funkce rozdělit na 3 základní úrovně.

2.4.1 Procesní úroveň

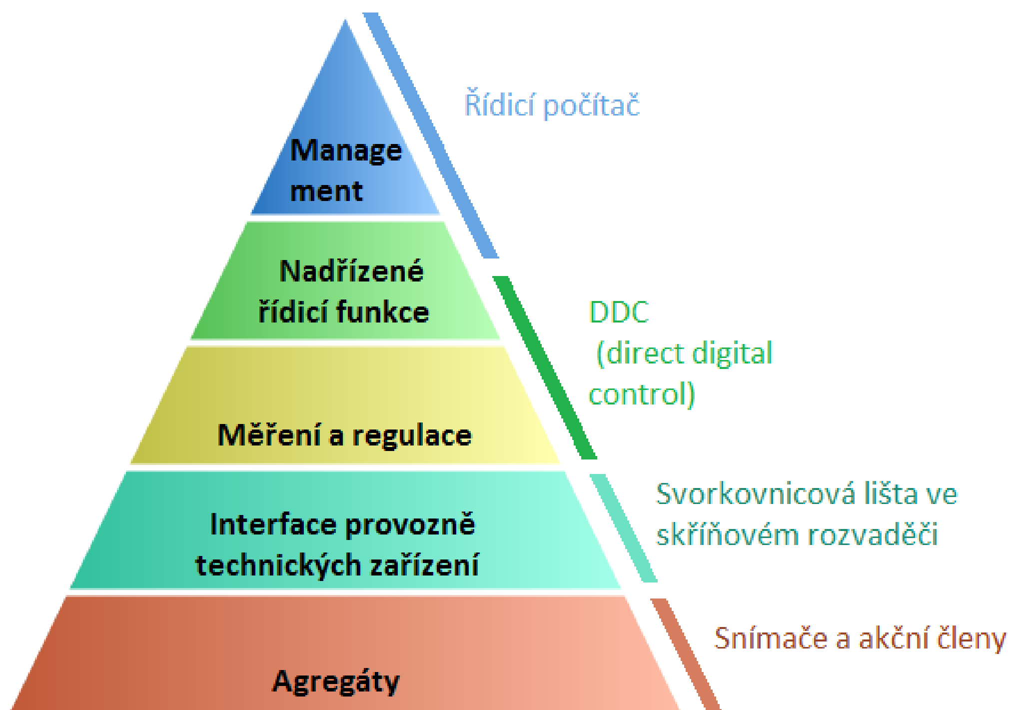
V této úrovní probíhá kontakt řídicího systému s reálným světem. Probíhá měření dat potřebných pro řízení procesu, a zároveň jsou fyzicky vykonávány příkazy z řídicího systému.V případě řídicího systému budovy se například jedná o snímače teploty, průtokoměry, ventily, tlakové snímače, klapky, ventilátory atd.

2.4.2 Úroveň řídicích buněk

Na této úrovni je využíváno dat získaných z procesní úrovně k automatickému řízení systémů budovy podle naprogramovaných řídicích smyček. Řídicí jednotky se umisťují na svorkovnice v rozvaděčových skříních. Skříně jsou zpravidla poblíž řízených TZB systémů, je tak snížena potřebná délka rozvodů.

2.4.3 Správcovská úroveň

Na této úrovni je zpravidla uživatelské rozhraní pro operátora který má možnost sledovat a případně zasahovat do jednotlivých řídicích procesů (např. časové plány, tvorba alarmů, dlouhodobý sběr dat). Jsou k dispozici všechna data z nižších úrovní.



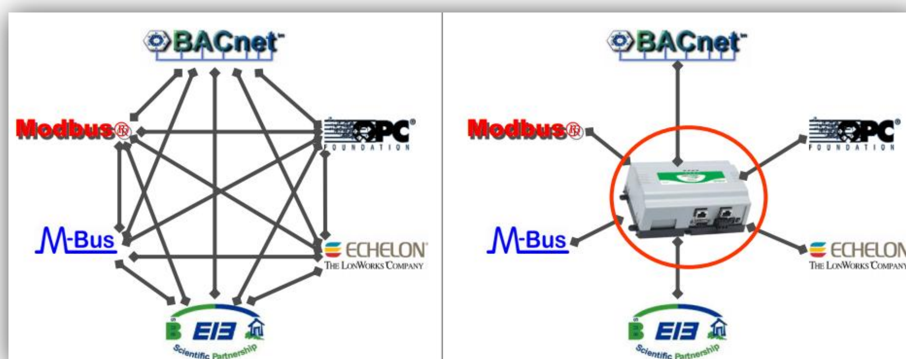
Obrázek 2 - hierarchický model [19, upraveno]

3 INTEGROVANÝ ŘÍDICÍ SYSTÉM BUDOV CENTRALINE

3.1 Úvod, platforma, stanice

Mezi hlavní požadavky na řídicí systém budovy patří interoperabilita mezi jednotlivými systémy TZB a integrace těchto systémů z hlediska řízení. Typická instalace zahrnuje několik různých protokolů, které je nutné integrovat do jednotného celku. V praxi se používají v zásadě 2 způsoby.

První způsob je využití převodních zařízení, které zajistí převod informace z jednoho protokolu na druhý. Druhým způsobem je využití integrační platformy. Takovou integrační platformou je například řídicí systém CentraLine AX.



Obrázek 3 - výhody integrovaného řídicího systému [2]

Integrovaný řídicí systém CentraLine je založen na Niagara frameworku. Hlavním cílem je zachování stávajících technických zařízení a jejich integrace do jednotného řídicího systému. Takto získaná data z různých podsystémů budovy - topení, ventilace, vzduchotechniky, osvětlení, měření spotřeby energií atd. mají sloužit k poskytování komplexnějších informací pro řídicí algoritmus a umožňovat řízení a monitoring budovy z jednoho místa.

Prakticky můžeme řídicí systém Centraline rozdělit na 3 části, kterými jsou vývojové prostředí COACH AX, embedded regulátor HAWK a grafická centrála ARENA AX. Pro úvod je nutné definovat pojmy platforma (platform) a stanice (station) tak, jak jsou chápány v rámci Niagara frameworku.

Pojem stanice je pouze softwarový koncept, který představuje databázi obsahující celý řídicí program, tzn. všechna řídicí logika, všechny datové body

které odpovídají reálným zařízením, a další funkce včetně sběru dat, nastavení alarmových hlášení a spravování síťového provozu [1].

Pojem platforma je používán pro skutečné zařízení, na němž je spouštěna stanice. Může se jednat o osobní počítače nebo o speciální regulátory, v našem případě HAWK. Všechny objekty a knihovny Niagara frameworku jsou nezávislé na zvolené platformě [1].

3.2 Regulátor HAWK

Regulátor HAWK slouží jako platforma pro spouštění Niagara stanice. Regulátor je prodáván v několika hardwarových provedení (HAWK 300E, HAWK 600E), hardwarové prostředky jsou uvolněny podle zakoupené licence. Operačním systémem regulátoru Hawk je QNX.

Obrázek 4 - regulátor HAWK [2, upraveno]

Řídicí program stanice je vytvářen v prostředí COACH AX, a poté nahrán do regulátoru. K regulátoru jsou dostupné i rozšiřovací moduly pro přímé připojení analogových nebo digitálních vstupů a výstupů, nebo karty s různými komunikačními rozhraními (RS232, RS485, LON interface). HAWK podporuje ovladače pro BacNet, LonWorks, EIB/KNX, M-Bus, Modbus a další [3].

Regulátor je určen pro montáž na DIN lištu. Ač není cílem práce zabývat se obchodní strategií prodeje regulátorů, je nutné zmínit jednotlivé dostupné licence podrobněji protože se k nim vážou jisté technické omezení. Následující tabulky slouží k porovnání technických parametrů HAWK 300E a HAWK 600E, tak jak je udává výrobce:

Tabulka 2: HAWK 300E, Procesor PowerPC 405ex 400 MHz, 256 MB DDR RAM, 128 MB Serial Flash

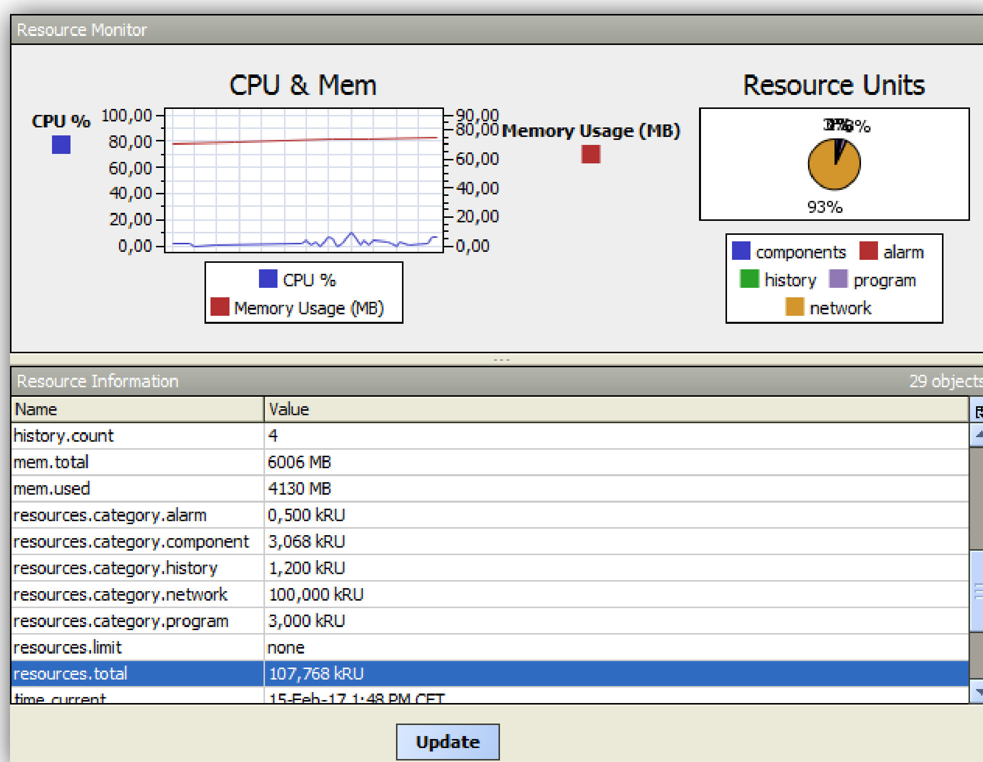
typ	Java Heap paměť [MB]	kResource jednotek	Doporučených bodů
HAWK350E	96	bez limitu	1000
HAWK340E	24	bez limitu	600
HAWK330E	24	500	400
HAWK320E	24	400	200

Tabulka 3: HAWK 600E, Procesor PowerPC 440 524 MHz, 256 MB DDR RAM, 128 MB Serial Flash

typ	Java Heap paměť [MB]	kResource jednotek	Doporučených bodů
HAWK660E	96	bez limitu	1200
HAWK650E	48	bez limitu	800
HAWK640E	48	1000	400
HAWK630E	48	450	200

Parametr "Resource jednotka" přiřazuje každému objektu (jako je datový bod, síť, historie, řídicí bloky atd.) ve stanici určitý počet z celkového počtu Resource jednotek. Tento počet je pevně daný, a nevykazuje v podstatě nic o vytížení hardwaru platformy (i když logicky více resource jednotek bude odpovídat větší zátěži, vytížení hardware se může pro stejné číslo resource jednotek lišit). Fakticky se tak jedná o umělý parametr, který podněcuje koncové zákazníky k nákupu dražších licencí, pokud jejich aplikace přesáhne určitý rozsah. Při překročení Resource limitu je uživatel na tuto skutečnost upozorněn, a při restartování platformy nemusí nahraná stanice vůbec nabootovat [4].

Počet doporučených bodů již lépe odpovídá technickému omezení, přičemž podle často kladených otázek je pro bezproblémový chod stanice nutné udržovat stálé využití procesoru pod 80 % a stálé využití java heap paměti pod 75 %. Pokud nejsou tyto podmínky dodrženy, je možné očekávat "nepředvídatelné chování" - jako příklad je uváděno špatné čtení dostupné paměti, nebo vynechávání zápisu u logovaných hodnot [4].



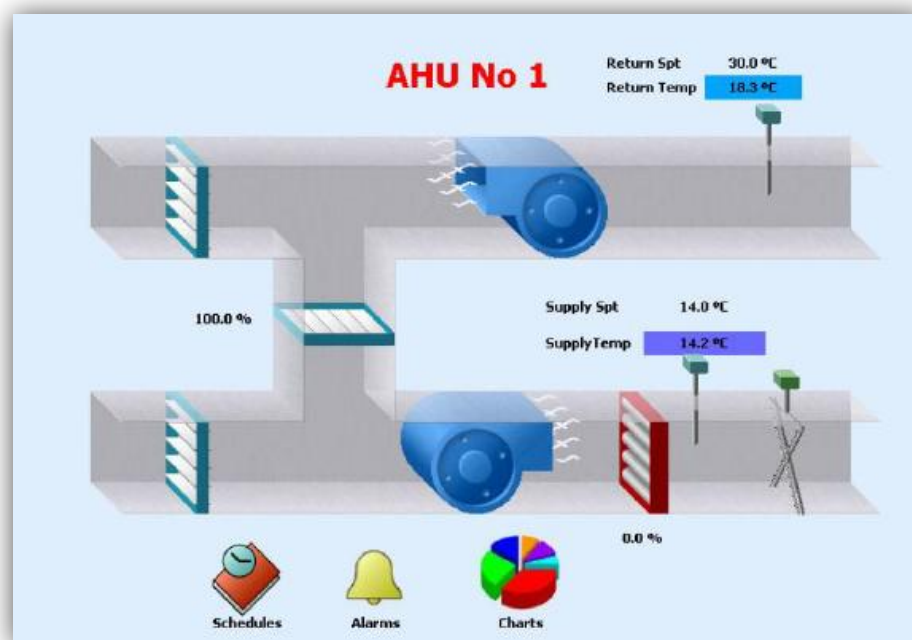
Obrázek 5 - Zobrazení resource jednotek a využití HW prostředků

3.3 Grafická centrála ARENA AX

ARENA AX je software pro monitorování řídicího systému budovy. Poskytuje uživatelské rozhraní pro správu a monitorování topení, vzduchotechniky a klimatizace, osvětlení, zabezpečení, požární signalizace a dalších systémů v rámci jedné nebo několika budov. Grafické rozhraní je vytvářeno opět v prostředí COACH AX.

Mezi hlavní funkce patří podpora webového serveru, takže uživatel má přístup k aktuálním datům pomocí webového prohlížeče, směrování a potvrzování alarmů pomocí emailu nebo sms, správa bezpečnosti pomocí nastavení uživatelských práv, a sběr a archivace dat pomocí databázových a textových formátů (například SQL, MySQL, CSV, DB2, HTML, XML).

Většinou je ARENA AX připojena do společné sítě s jedním nebo několika regulátory HAWK nebo EAGLE. Maximální počet připojitelných zařízení se liší podle zakoupené licence, stejně tak je k dispozici celá řada ovladačů pro rozšíření počtu připojitelných datových bodů například ze sítí LON, OBIX, nebo Modbus TCP. Zvlášť jsou dostupné i databázové ovladače pro archivaci dat a ovladače pro připojení bezpečnostních kamer. [6, 7]



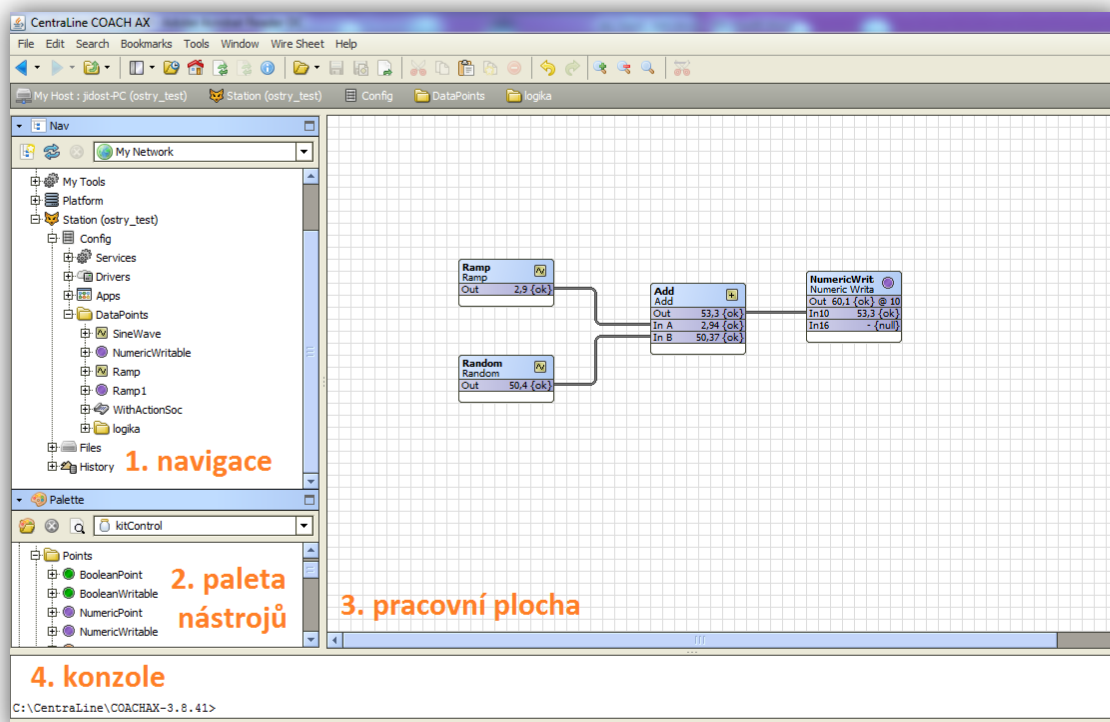
Obrázek 6 - Zobrazení vzduchotechnické jednotky v ARENA AX [2]

3.4 Vývojový software COACH AX

Coach AX je vývojový a inženýrský nástroj který slouží primárně k programování regulátoru HAWK a grafické centrály ARENA AX. Za použití tohoto nástroje je uživatel schopen nadefinovat datové body a řídicí logiku stanice, nastavovat alarmy a sběr dat a vytvářet grafickou vizualizaci procesu pro ARENA AX. Toto vývojové prostředí je v rámci Niagara frameworku nazýváno jako workbench.

Uživatel - programátor tvoří řídicí logiku za pomocí funkčních bloků (znázorněno na obrázku 7), programování probíhá intuitivně spojováním funkčních bloků do logických celků. Tyto funkční bloky jsou v rámci Niagara Frameworku nazývány jako komponenty.

Komponenty jsou distribuovány pomocí modulů. Každý modul obsahuje komponenty zaměřené na určitou funkci. Mezi nejpoužívanější moduly pro tvorbu řídicích algoritmů patří moduly control, kitControl, alarm, history a program.



Obrázek 7 - Rozložení pracovního prostředí COACH AX

Pracovní prostředí COACH AX se dá rozlišit na 4 hlavní oblasti - navigace slouží k pohybu v souborovém systému platformy a stanice, paleta nástrojů nabízí funkční bloky využitelné pro programování stanice, pracovní plocha lze přepínat do několika zobrazovacích režimů pro programování a konfiguraci jednotlivých bloků a stavový řádek konzole umožňuje ovládat COACH AX pomocí příkazů. Řídicí systém využívá již několikrát zmiňovaného frameworku Niagara, který bude dále podrobněji popsán.

4 NIAGARA FRAMEWORK

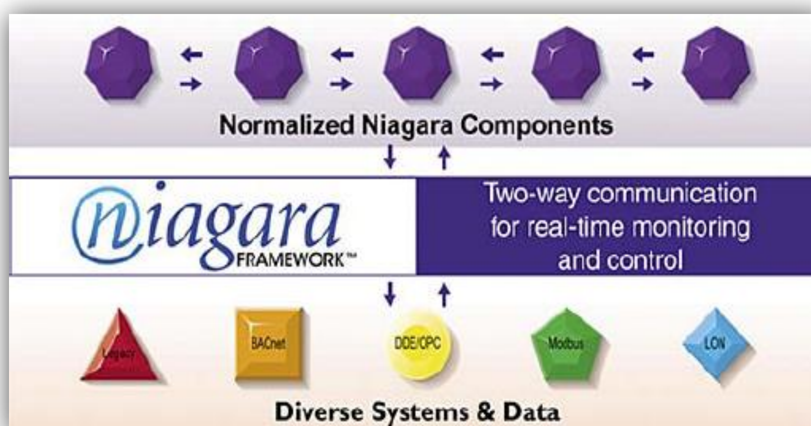
4.1 Historie a úvod

Niagara framework je od 90. let vyvíjen americkou společností Tridium. Cílem bylo vytvořit otevřený systém pro vývoj aplikací zajišťujících interoperabilitu mezi zařízeními od různých výrobců s využitím internetu pro vzdálený přístup a správu. V roce 1997 byl dokončen první funkční prototyp a v roce 1999 byla vydána první verze frameworku Niagara. V roce 2000 je Niagara framework poprvé dostupný na evropském trhu.

V dalších letech společnost Tridium začíná spolupracovat s významnými společnostmi v HVAC odvětví (Emerson, Carrier Corporation, Schneider Electric). V roce 2005 byla společnost Tridium akvizována společnostní Honeywell. V roce 2008 překročil niagara framework 100 tisíc použití, o čtyři roky později překročil 300 000. V roce 2015 byla vydána zatím nejnovější verze frameworku Niagara 4. [8]

Základní část frameworku byla vyvíjena jako otevřený standard pro automatizaci budov označovaný Baja (Building Automation Java Architecture), ovšem v roce 2016 bylo od této snahy z důvodu malého zájmu upuštěno. [9]

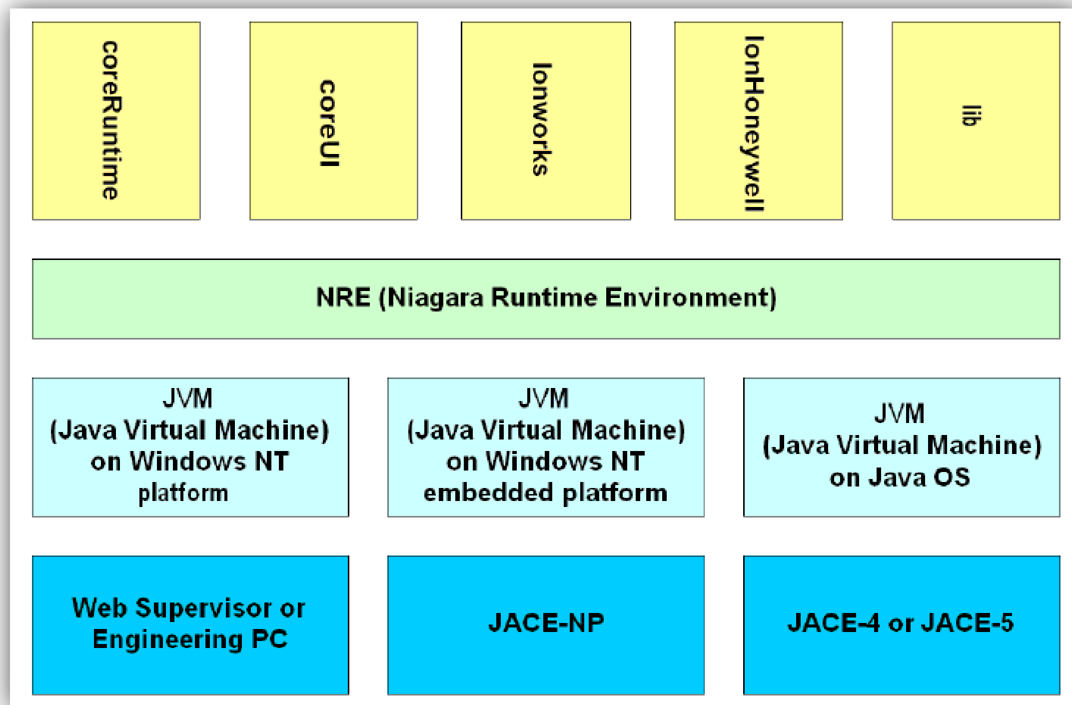
Snaha o integraci jednotlivých standardů vyústila ve vznik "společného objektového modelu" (Common object model), který slouží k normalizaci získaných dat z různých zařízení od různých výrobců. Získané data jsou převedeny podle společného modelu na odpovídající reprezentaci reálného zařízení v rámci Niagara frameworku. Tento model plní integrační funkci frameworku.



Obrázek 8 - Společný objektový model, Common object model [1]

4.2 Architektura frameworku

Niagara framework je napsán v jazyce Java, což znamená že může být spouštěn na různých zařízeních. Podmínkou je aby byl k dispozici interpret jazyka Java, tzv. JVM (Java Virtual Machine). Framework je určen pro J2ME platformy (potažmo J2SE v případě programování vizualizace pro grafické centrály).



Obrázek 9 - Architektura Niagara frameworku [1]

Architekturu můžeme rozdělit na 4 základní vrstvy. Nejnižší první vrstva představuje fyzické zařízení, na kterém je software spouštěn. Nejčastěji se jedná o osobní počítače nebo JACE (Java Application Control Engine) regulátory (v našem případě HAWK). Nad touto vrstvou je interpret jazyka Java JVM který se liší podle použité platformy. třetí vrstva představuje spuštěný Niagara software (NRE - Niagara Runtime enviroment). NRE vykazuje totožné chování napříč platformami.

Niagara framework je distribuován pomocí modulů, které tvoří nejvyšší čtvrtou vrstvu. Jednotlivé moduly zajišťují funkcionalitu frameworku. Mandatorní moduly jsou coreRuntime a coreUI. Moduly jsou distribuované jako soubory JAR (Java Archive - komprimovaný soubor který obsahuje zdrojové kódy, metadata a ostatní zdroje nutné pro spuštění programu). Moduly obsahují informaci o závislosti na jiných modulech a jejich verzích. Uživatel může zvolit, které moduly mají být na stanici nainstalovány.

4.3 Niagara objektový model

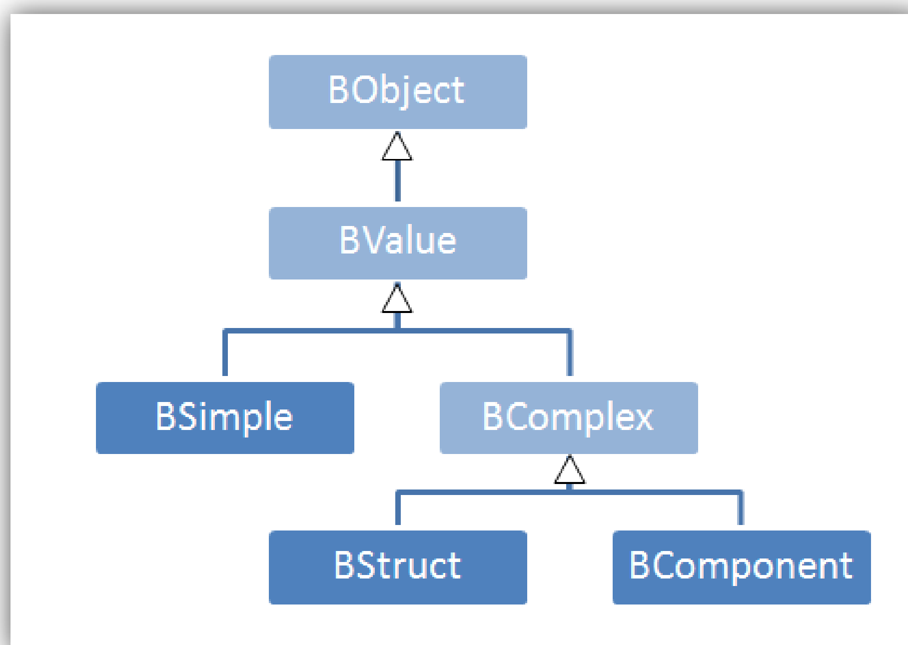
4.3.1 Objekty, Type, Registry

Moduly obsahují a definují Niagara objekty a jejich mapování na příslušné Java třídy. Každý Niagara objekt obsahuje prvek Type, který je mapován na Java třídu. Objekty představují upravenou verzi JavaBean objektů rozšířených o funkcionalitu pro použití v distribuovaném řídicím systému.

Každý Niagara objekt obsahuje definici svého Niagara Type. V porovnání s objektovým modelem je Niagara Type protějškem Java třídy. Souhrn všech Niagara Types je uchováván v databázi nazvané Registry (rejstřík). Tato databáze je obnovena, pokud je detekována změna v instalovaném modulu nebo pokud je přidán modul nový. Registry slouží hlavně k dotazování na nainstalované moduly a typy bez nutnosti opakovaného načítání do paměti.

Tabulka 4 : Porovnání Java a Niagara Objektů

Java	Niagara
`Java.lang.Object`	`Javax.baja.sys.BObject`
`Java.lang.Class`	`Javax.baja.sys.Type`



Obrázek 10 - Niagara objektový model

Mezi základní vlastnosti a funkce Niagara objektu patří:

Jednotné pojmenování - každý objekt má identifikátor ORD (object resolution descriptor) podobný URL adrese, díky kterému je možné se navigovat v sítí několika stanic.

Vlastnosti (Properties) - každý objekt má pojmenované vlastnosti které definují jeho stav. Tyto vlastnosti mohou být spojeny s jinými objekty. Jednotlivé vlastnosti slouží ke čtení nebo zápisu. Vlastnosti objektu bývají trvalého charakteru, a jsou ukládány na pevný disk (nebo flash paměť) platformy.

Serializace - z výše napsaného vyplývá že pro uložení instance objektu je nutné převést tento objekt na proud bytů při ukládání objektu a z proudu bytů zase zpět.

Vazby (Linking) - Niagara objekty mohou být spojeny z hlediska vstupních a výstupních vlastností (viz dále).

4.3.2 Komponenty a služby

Komponenty jsou speciálním případem Niagara objektů. Komponenty tvoří základní stavební kameny stanice a jsou definovány jako datové struktury s předdefinovaným chováním, které jsou využívány jako funkční bloky pro programování řídicích aplikací. Komponenty jsou definovány pomocí slotů, které udávají vlastnosti a chování komponent.

Sloty jsou v rámci komponenty identifikovány unikátním jménem. Každý slot má přiřazenou 32-bitovou masku flagů které definují další vlastnosti slotu (například zobrazení/skrytí, pouze ke čtení, atd). Sloty u komponenty můžeme rozdělit do 3 základních kategorií:

1) Property slot - ukládá vlastnosti komponenty. Příkladem může být měřená teplota u datového bodu.

2) Action slot - slot pro vyvolání určitého chování komponenty v případě přímého příkazu uživatele nebo jiné události. Příkladem může být akce přepsání žádané hodnoty uživatelem na předem danou dobu.

3) Topic slot - topic představuje předmět události. Nedefinují chování při události, ale pouze to že k události došlo. Příkladem může být vyvolání daného topicu při překročení limitu u měřené hodnoty.

Niagara služba je druh komponenty která globálně poskytuje funkcionalitu jiným komponentám. Je zajištěno, že je při startu stanice spuštěna před ostatními komponentami a při ukončení je ukončena až po ukončení ostatních komponent.

4.3.3 Vazby

Niagara framework je založený na zpracování událostí, a vazby (Links) slouží k přenášení těchto události z jedné komponenty na další. Vazby mohou být v aktivovaném nebo deaktivovaném stavu. Událost je v rámci Niagara frameworku definována jako:

- změna hodnoty property slotu u komponenty
- vyvolání action slotu o komponenty
- vyvolání topic slotu u komponenty

Link u komponent definuje vztah mezi jejich 2 sloty. Jednotlivé sloty u komponent mohou být propojeny podle této tabulky:

Zdroj	**Cíl**	**Popis**
Property	*Property*	Pokud se změnila zdrojová property, změň cílovou property
Property	*Action*	Pokud se změnila zdrojová property, vyvolej cílovou akci
Action	*Action*	Pokud byla vyvolána zdrojová akce, vyvolej cílovou akci
Action	*Topic*	Pokud byla vyvolána zdrojová akce, vyvolej cílový topic
Topic	*Action*	Pokud byl vyvolán zdrojový topic, vyvolej cílovou akci
Topic	*Topic*	Pokud byl vyvolán zdrojový topic, změň cílový topic

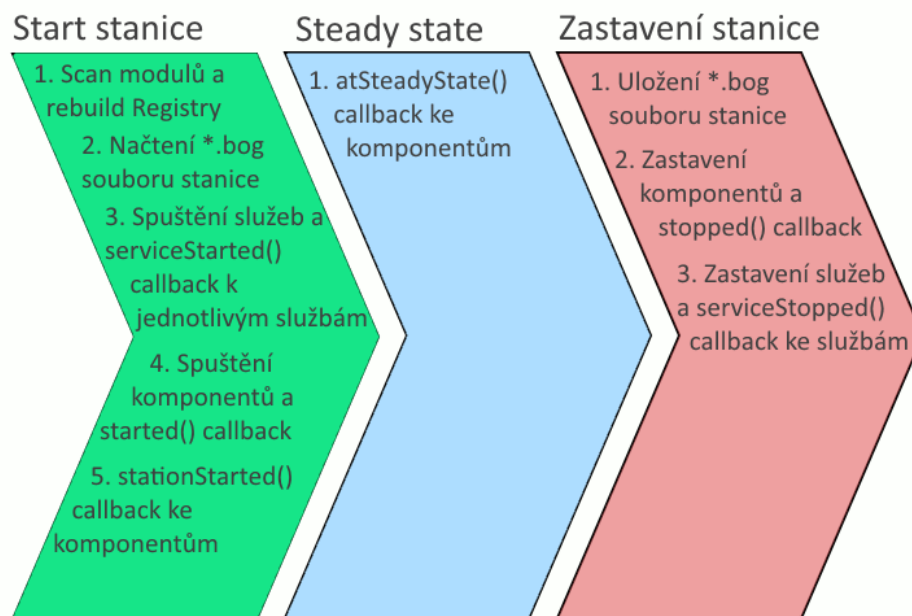
4.4 Stanice

Pojmem stanice je nazýván celkový řídicí program, který je spouštěný na platformě. Stanice je uložena jako komprimovaný *.bog soubor (BOG - Baja Object Graph). Pro vytvoření stanice slouží průvodce, v němž uživatel nastaví jméno stanice a administrátorský účet. Vytváření (i editace) stanice je prováděno v prostředí COACH AX. Zvolené jméno stanice je neměnné a je dále používané například při navigaci a v odkazech na obsah stanice.

Spouštění stanice lze rozlišit na oblast platformy - Start a Stop, a oblast stanice - Open a Close. Start stanice je proces, při kterém je konfigurační *.bog soubor stanice deserializován a nahrán do operační paměti platformy. Po startu přejde stanice do stavu Steady state. Na každé platformě může být spuštěna právě

jedna stanice, avšak na úložišti platformy (např. pevný disk PC) může být uloženo více stanic.

Naproti tomu Open a Close slouží k přístupu do databáze stanice za účelem její modifikace nebo zobrazení. V jednu chvíli může být ke stanici připojeno několik uživatelů. Připojování a odpojování uživatelů pomocí Open a Close tedy nemá vliv na celkový chod stanice. Životní cyklus stanice lze rozdělit na následující kroky:



Obrázek 11 - Životní cyklus stanice

Tento cyklus zajišťuje sekvenční spouštění pro komponenty a služby ve stanici. Prvním krokem je kontrola modulů a v případě jejich změny je vytvořena aktualizovaná databáze Registry. Následuje deserializace *.bog souboru stanice a spuštění služeb a komponent.

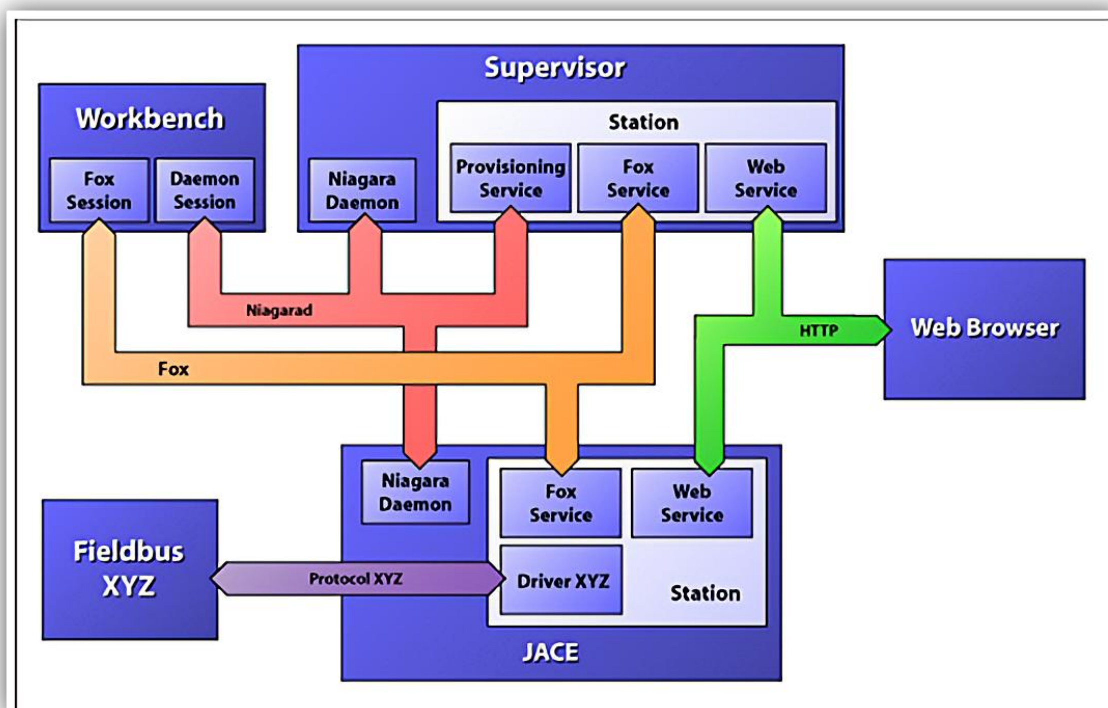
Každá služba a komponenta má vlastní callback metody, které jsou volány v různých stavech stanice. Tyto metody zajišťují například inicializaci nebo uvolnění prostředků. Při startu stanice je to `started()` metoda pro komponenty a `serviceStarted()` metoda pro služby.

Po startu stanice běží nastavený časovač (defaultně po dobu 10 s), a po uběhnutí této doby je volán `atSteadyState()` callback u komponent. Tato doba slouží jako prodleva u některých řídicích algoritmů. Uživatel může zjistit zda-li tato doba uběhla pomocí metody `Sys.atSteadyState()`. Doba trvání této prodlevy je nastavitelná parametrem *nre.steadystate*.

Při zastavení stanice dojde nejdříve k uložení *.bog souboru, poté jsou zastaveny komponenty a volán `stopped()` callback. Jako poslední jsou zastaveny služby a volán `serviceStopped()` callback.

4.5 Celkový přehled, protokoly

Výše byly popsány některé základní pojmy Niagara frameworku. Následující schéma slouží ke znázornění stanice a platformy a protokolů používaných v rámci Niagara frameworku.



Obrázek 12 - Přehled částí Niagara frameworku [1]

Z obrázku vidíme, že každá stanice je hostovaná uvnitř platformy. Na každé platformě je spuštěno několik daemonů (u Windows platforem se jedná o služby, u QNX platforem jsou to daemon procesy). K jednotlivým stanicím a platformám se uživatel může připojit přes vývojové prostředí Niagara Frameworku workbench. Ke stanicím se lze připojit i přes webový prohlížeč, pokud je spuštěna příslušná služba na stanici. Fieldbus znázorňuje sběrnici připojenou do stanice přes příslušný Driver, který poskytuje převod dat podle výše zmíněného společného objektového modelu.

Tři hlavní znázorněné protokoly jsou HTTP protokol, protokol Fox, a protokol Niagarad. HTTP je standardní protokol webových prohlížečů. Niagara poskytuje 2 technologie pro zobrazení stanice, první technologie využívá Java plugin v daném prohlížeči, druhá technologie nazvaná Hx využívá pro zobrazení

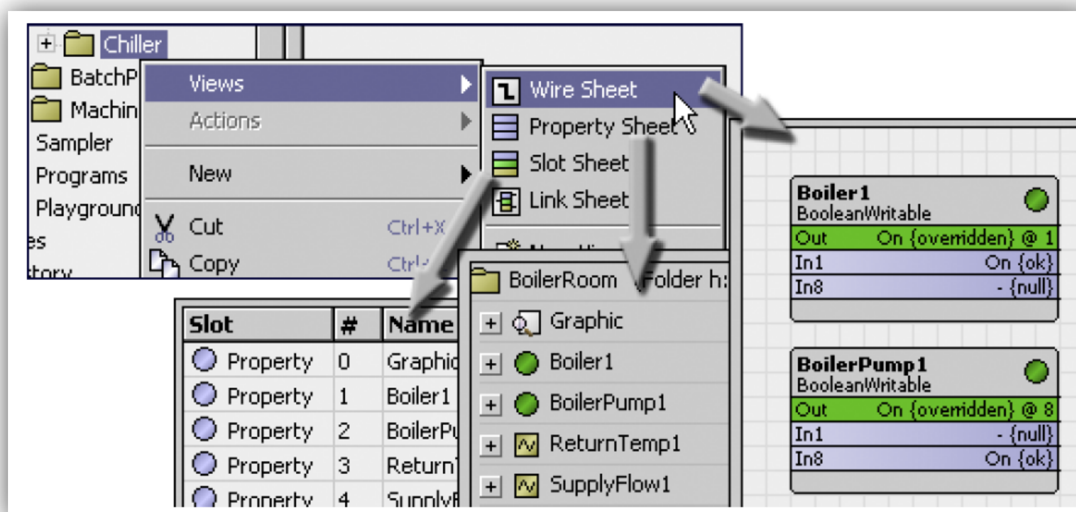
pouze HTML, CSS a JavaScript. Niagarad je proprietární protokol vyvinutý pro komunikaci mezi workbenchem a daemony dané platformy.

Fox je proprietární TCP/IP protokol, který je používán pro komunikaci mezi jednotlivými stanicemi, a mezi stanicemi a workbenchem. Defaultním portem je port 1911.

4.6 Niagara Framework z pohledu uživatele

V následující části práce bude čtenáři přiblíženo vývojové prostředí CoachAX a části Niagara frameworku, se kterými se běžný uživatel setká při práci v tomto vývojovém prostředí při tvorbě stanice.

Práce ve vývojovém prostředí CoachAX je orientovaná na grafické programování. Základní prvky při vytváření řídicí logiky jsou komponenty, které mohou být zobrazeny v několika pohledech. Každý pohled slouží k různým účelům při konfiguraci jednotlivých komponent.



Obrázek 13 - Pohledy ve vývojovém prostředí [22]

Wiresheet view - slouží pro tvorbu řídicí logiky, uživatel může systémem drag and drop vkládat a propojovat jednotlivé komponenty pomocí zobrazených slotů.

Property view - zobrazení vlastností jednotlivých komponent. Uživatel může nastavovat jednotlivé parametry pro dané komponenty.

Slot view - zobrazení všech slotů (včetně skrytých) a jejich flagů u dané komponenty.

Link view - zobrazení vazeb mezi jednotlivými komponenty.

Niagara framework také poskytuje možnost vytvoření vlastního pohledu vázaného k jakékoliv komponentě. Toho je v této práci využito pro vytvoření pohledu pro zadávání tagů k datovým bodům.

4.6.1 Základní datové typy Niagara frameworku

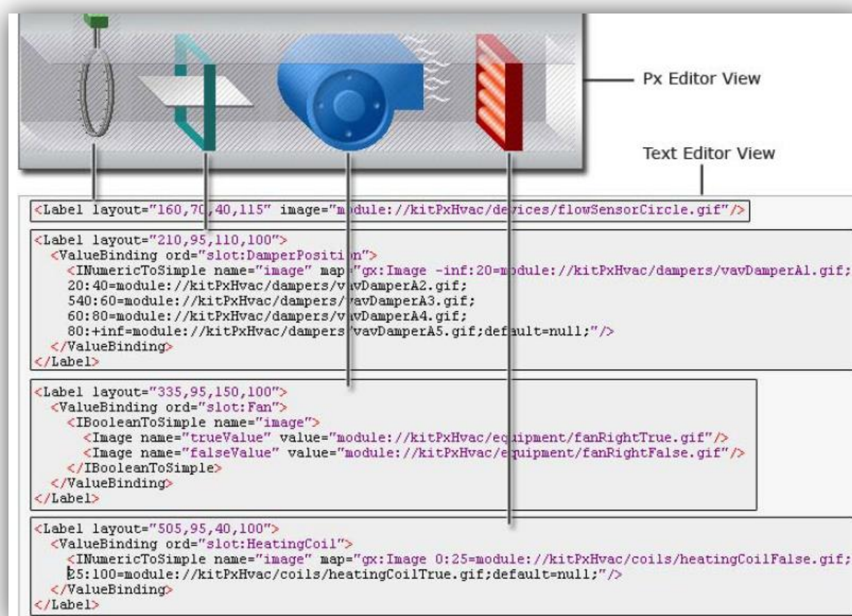
Data jsou reprezentována jako body (`points`). Tyto body mají různé datové typy. V Niagara frameworku jsou používány 4 základní datové typy - `Boolean`, `Numeric`, `String` a `Enum`.

U jednotlivých bodů může být nastaven režim zobrazení (`Facets`). Tím je určeno, jaká jednotka se bude s daným bodem používat. `Point extensions` jsou rozšíření pro jednotlivé body. Tyto rozšíření se dělí na `alarm`, `history`, a `control extensions`. Pomocí rozšíření uživatel definuje další vlastnosti, které ho u daného bodu zajímají.

Každý datový bod obsahuje také `status flag`. Tím je znázorněn stav datového bodu (například zařízení v poruše, komunikační chyba, alarm nebo převzetí kontroly operátorem).

4.6.2 Vizualizace

Vizualizace technických zařízení je realizována pomocí Px pohledů (views). Px pohledy jsou XML soubory, které definují vazby mezi zobrazenou grafikou a připojenými daty. K dispozici je grafický editor pro tvorbu těchto souborů (Px Editor).



*Obrázek 14 - Grafické a XML zobrazení *.Px souboru [22]*

5 TVORBA APLIKACE V NIAGARA FRAMEWORKU

5.1 Úvod

Niagara Framework poskytuje uživatelům možnost vývoje vlastních modulů. Uživatel využívá Java aplikačního rozhraní jednotlivých modulů Niagara frameworku pro implementaci vlastních řešení.

Tato kapitola má slouží jako názorný pracovní popis vývoje uživatelského modulu, který obsahuje komponenty pro sběr dat a službu pro jejich odesílání. Všechny dále uvedené informace jsou platné pro Niagara Framework verze 3.8.41.

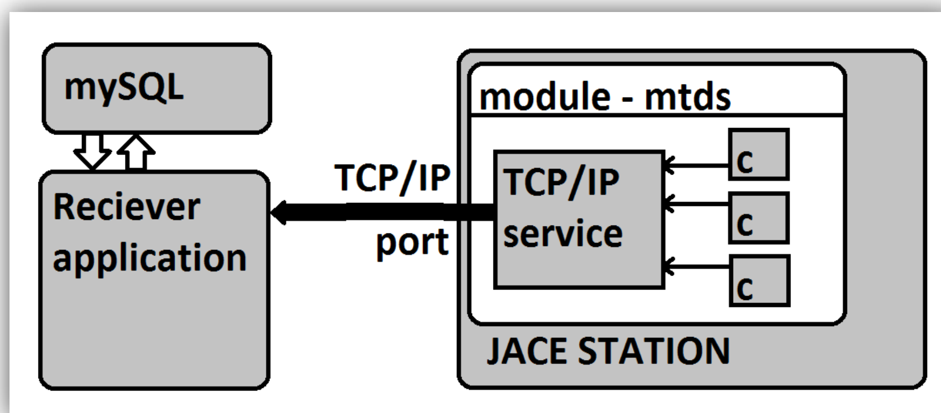
5.2 Návrh řešení

Původní záměr bylo vytvoření jednoho modulu, který bude obsahovat komponenty pro sběr dat z vybraných datových bodů, a službu která bude zajišťovat komunikaci Niagara Frameworku přímo s MySQL databází.

Tento záměr nebylo možné realizovat, protože při kompilaci modulu je vývojář omezen pouze na Java edice j2se 1.4 a j2me 1.4. Edice j2me, která je defaultní možností a je nutnou podmínkou pro použití modulu v JACE regulátoru, neumožňuje použití JDBC driveru pro přístup do databáze. Starší verze také znamenají drobné problémy, například chybějící utility třídy jazyka Java, ovšem jejich protějšky lze ve většině případů nalézt jako součást Niagara Frameworku.

Konečný návrh řešení, které splňuje zadanou funkcionalitu, je modul který obsahuje komponenty pro sběr dat z vybraných datových bodů, a služba která zajišťuje odesílání těchto dat do externí aplikace přes TCP/IP protokol. V externí aplikaci pro příjem dat jsou data ukládána do databáze.

Kromě časové značky a hodnoty jsou u jednotlivých datovým bodů ukládány i metadata o těchto bodech. Metadata jsou ukládány v podobě tagů, které jsou definované v opensourcové iniciativě *"Project Haystack".*



Obrázek 15 - Schematický návrh řešení

5.3 Project Haystack

5.3.1 Úvod

Project Haystack je opensourcová iniciativa založená v roce 2014. Snahou projektu je vytvoření a standardizace sémantických datových modelů (modelů, které popisují *význam* dat) za účelem snadnějšího získávání znalostí z dat generovaných senzory. Zaměřuje se na využití v řídicích systémech budov (umožňuje sémantický popis dat získaný ze systémů topení, chlazení, ventilace, osvětlení, větrání atd.).

V praxi je obvykle význam datového bodu určen z jeho názvu, což v menším měřítku postačuje, ale ve velkém rozsahu to komplikuje schopnost porozumět generovaným datům. Jednak musí takový název být čitelný a srozumitelný pro člověka, ale zároveň musí obsahovat co nejvíce dodatečných informací, což jsou protichůdné požadavky které vyžadují kompromisy.

Výsledkem takového postupu bývá vznik určitého názvosloví v pojmenovávání datových bodů. Tyto názvosloví zpravidla nejsou kompatibilní, protože můžou vznikat na úrovni firem (například ahu1-vtlatO-P a ahu1-fan-exh-C můžou představovat různá jména pro stejný datový bod kterým je povel odtahového ventilátoru vzduchotechnické jednotky AHU 1).

5.3.2 Základní pojmy

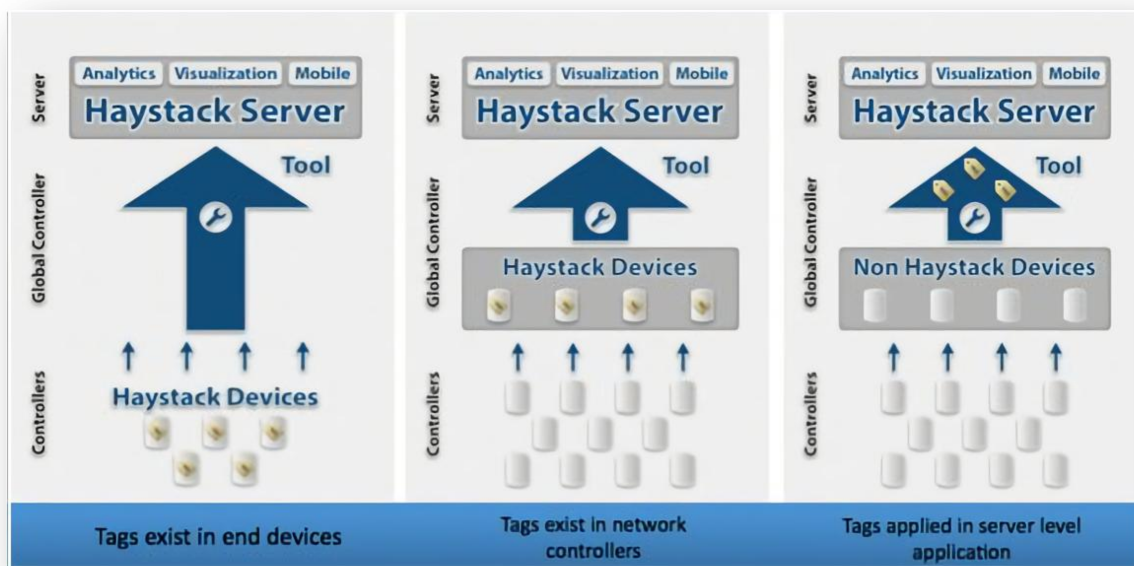
Project Haystack poskytuje možnost, jak pomocí sémantického modelu popsat význam jednotlivých datových bodů a jejich vzájemné vztahy. Mezi základní pojmy patří entity a tagy.

Za entitu je považována abstrakce fyzického objektu z reálného světa - jedná se například o budovu, technické zařízení budovy, snímač, atd. Project

Haystack se nezabývá tím, jak je entita reprezentovaná v softwarovém systému - může se např. jednat o řádek v tabulce nebo datový bod řídicího systému.

Haystack definuje, jak popisovat jednotlivé entity pomocí tagů. Tagy přiřazují entitám různé atributy, jsou párové a nepárové a dělí se na několik druhů. Mezi základní druhy tagů patří markery (označují příslušnost entity do určité skupiny, např. rozdělení podle zařízení), reference (definují vztahy mezi entitami), a několik dalších tagů pro označení geografické lokace, data, času nebo datového typu.

Tagy mohou být k entitám přidány na 3 úrovních, které víceméně odpovídají úrovním hiearchického modelu BAS. Nejnižší úrovní je úroveň zařízení. Prostřední úroveň je tvořena řídicími buňkami (v našem případě regulátor HAWK) a nejvyšší úroveň odpovídá správcovské úrovni (např. správcovská stanice ARENA).



Obrázek 16 - Přiřazení tagů na různých úrovních [13]

5.3.3 Struktura Haystack modelu

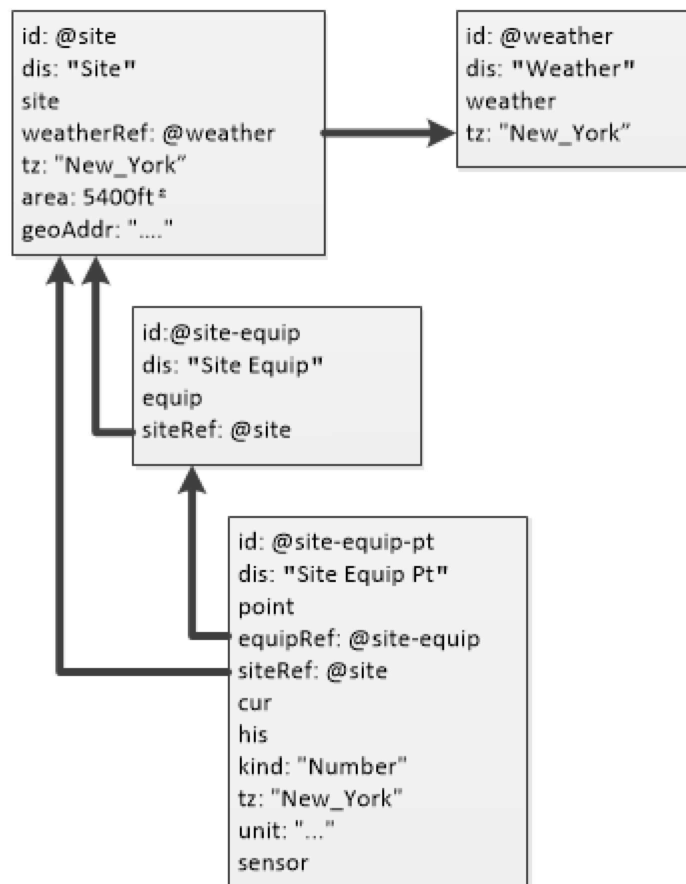
Struktura Haystack modelu je založena na hierarchickém rozlišení tří entit - site, equip a point. Entita s tagem site označuje budovu nebo umístění a je v hierarchii modelu nejvýše. Site je dále specifikována pomocí dalších tagů které označují např. geografické umístění, adresu, časovou zónu nebo plochu budovy.

Equip je označení pro fyzické nebo logické zařízení v rámci site - např. vzduchotechnická jednotka. Equip entity by měly obsahovat referenční tag s odkazem na site, do které náleží.

Point je označení pro digitální nebo analogové senzory, akční členy nebo setpointy v rámci zařízení (*equip*). Pointy jsou přiřazeny referenčním tagem ke svému zařízení (*equip*) a umístění (*site*). Pointy jsou označeny markery, které jednoznačně určují kde (*discharge, return, exhaust*), jaké médium (*air, water, steam*), a co (*humidity, temp, pressure, flow*) je měřeno nebo ovládáno.

Haystack definuje 13 zařízení, které obsahují sady markerů pro jejich body (*points*). Seznam zařízení a jejich tagů je k dispozici na [14]. Tyto sady byly použity v diplomové práci pro generování uživatelského rozhraní pro zadávání tagů u datových bodů v Niagara frameworku (uživatel zvolí typ zařízení, vyplní referenci, a zvolí si jednu možnost z předpřipravených markerů).

Tato implementace není úplná, protože se nezabývá umístěním (*site*) a referencemi na umístění, ale i bez toho přidává dostatečné informace o sbíraných datech z řídicího systému.

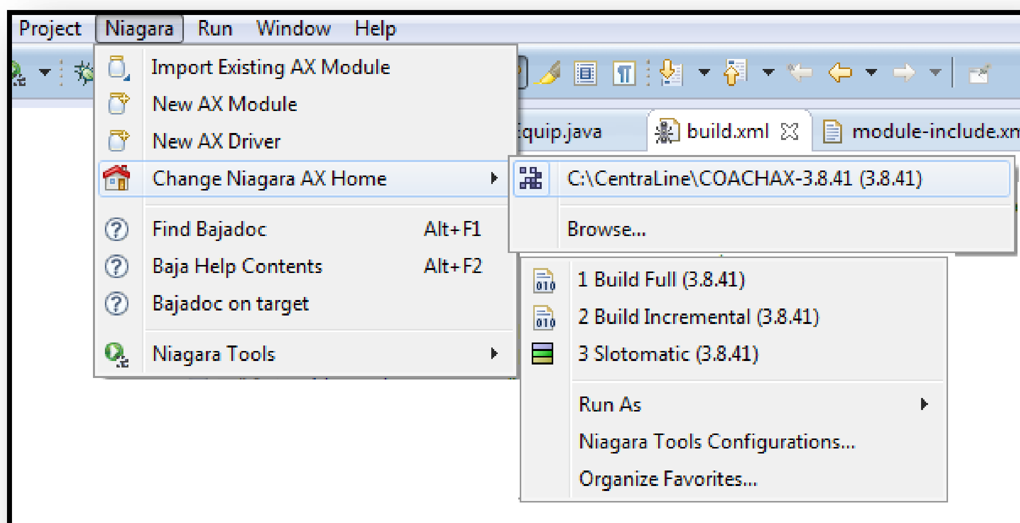


Obrázek 17 - Haystack model s referencemi [15]

5.4 Nastavení vývojového prostředí

Pro usnadnění práce vývojářů je k dispozici Niagara plugin pro známé vývojové prostředí Eclipse. Tento plugin je volně dostupný ke stažení z [11], a rozšiřuje vývojové prostředí Eclipse o nástroje Niagara Frameworku. Plugin je používán s Eclipse Juno 4.2.2 dostupného ke stažení z [12].

Instalace samotného pluginu je velmi jednoduchá, stačí stažený soubor zkopírovat do adresáře eclipse/dropins/ a spustit vývojové prostředí Eclipse. Po instalaci a spuštění Eclipse přibude v panelu nástrojů položka "Niagara". Aby mohl být plugin využíván, musí být na počítači nainstalován Niagara Framework s platnou licencí. Na jednom počítači může být nainstalováno i více instancí Niagara frameworku - v tomto případě musí uživatel zvolit cílový adresář dané instalace Niagara frameworku.



Obrázek 18 - nabídka Niagara pluginu

Nabídka Niagara pluginu umožňuje uživateli vytvoření nebo import modulu, vytvoření nového driveru, a zobrazení nápovědy pro moduly Niagara Frameworku. Dále jsou k dispozici nástroje pro inkrementální nebo celkové sestavení vytvořeného projektu. Výše zmíněné funkce jsou běžné pro většinu vývojových prostředí.

Za zmínku stojí nástroj Slotomatic. Jedná se o preprocesor který generuje Java kód pro Sloty komponent. Zdrojový kód pro Slotomatic je umístěn jako předdefinovaný komentář v hlavičce zdrojového souboru. Celý komentář se dá rozdělit do 3 bloků podle 3 typů Slotů v Niagara frameworku (Property, Action, Topic). Slotomatic umožňuje jednoduše nastavit komentář ke Slotu, defaultní

hodnotu a flagy slotu. Konkrétní příklad použití nástroje Slotomatic je uveden níže v textu.

5.5 Vytvoření modulu

Prvním krokem při tvorbě vlastní komponenty v Niagara frameworku je vytvoření modulu. Jak již bylo napsáno, modulem je souhrn Java balíčků a tříd a ostatních zdrojových souborů nutných pro funkci zkompilovaného kódu.

Pro vytvoření modulu použijeme průvodce vytvořením nového modulu dostupného v Niagara pluginu (*Obrázek 18*). V průvodci uživatel nastaví název, symbol, popis, autora modulu a umístění modulu (pokud je v umístění ponechán "!", je modul vytvořen v nastaveném adresáři Niagara frameworku).

V dalších krocích může uživatel nastavit závislost na ostatních modulech a případně specifikovat které Java balíčky mají být součástí modulu. Vytvořený modul se kromě zdrojových Java souborů skládá z dalších 4 konfiguračních souborů, které definují chování v rámci Niagara frameworku. Tyto soubory jsou `build.xml`, `module-include.xml`, a volitelné soubory `module.lexicon` a `module.palette`.

5.5.1 Build.xml

Tento soubor definuje, jak má být modul sestaven. Jedná se o XML strukturu s hlavním tagem `module`, který obsahuje další elementy `dependency`, `package` a `resource`.

```
<!-- Module Build File -->
<module
  name = "myTcpDataSender"
  bajaVersion = "0"
  preferredSymbol = "mtds"
  description = "Module with controlpoint interval history extensions
  vendor = "xdosta37"
>
  <!-- Dependencies -->
  <dependency name="baja" vendor="Tridium" vendorVersion="3.8" />
  <dependency name="history" vendor="Tridium" vendorVersion="3.8" />
  <dependency name="control" vendor="Tridium" vendorVersion="3.8" />
  <dependency name="bajaui" vendor="Tridium" vendorVersion="3.8" />
  <dependency name="workbench" vendor="Tridium" vendorVersion="3.8" />
  <dependency name="gx" vendor="Tridium" vendorVersion="3.8" />

<package name="com.talker" />
<package name="com.hayslot" />
<package name="com.hayslot.ui" />

<resources name="com/haystack/*.txt" />
```

Obrázek 19 - Soubor build.xml

Element module - obsahuje povinné atributy `name, bajaVersion, preferredSymbol, description` a `vendor` které jsou uživatelem vyplněny v průvodci vytvořením nového modulu.

Element depedency - obsahuje povinné atributy `name` a `vendor`, které udávají závislost modulu na ostatních modulech. Nepovinný atribut `vendorVersion` specifikuje minimální verzi požadovaného modulu.

Element package - obsahuje povinný atribute `name` pro označení Java balíčků, které jsou součástí modulu. Mezi nepovinné atributy patří atribut `edition`, který určuje v jaké edici bude Java kód kompilován (defaultně j2me, volitelné možnosti j2se a j2se-5.0). Dalším nepovinným atributem je atribut `compile`, který určuje jestli bude balíček kompilován (nastavení true nebo false). Posledním nepovinným atributem je atribut `install` který umožňuje odebrat balíčky modulu podle instalovaného prostředí Niagara stanice (možnosti *ui, doc, runtime*).

Element resource - zdrojový kód modulu může obsahovat např. odkazy na obrázky. Tyto soubory musí být uvedené jako resource element modulu. Obsahuje povinný atribut `name` (specifikuje relativní umístění vůči adresáři "src", je možné využívat speciální znaky pro označení více souborů stejného typu) a nepovinný atribut `install.`

5.5.2 module-include.xml

Soubor `module-include.xml` slouží k mapování Niagara Type na odpovídající Java třídu. Všechny objekty v Niagara frameworku jsou potomky z třídy `BObject`. Všechny třídy musejí obsahovat statickou proměnou typu *Type*. *Type* slouží k identifikaci Niagara Objektu v rámci frameworku, a to ve formátu (název modulu):(název *Type*).

```xml
<types>

  <type name="CommonQueueService" class="com.talker.BCommonQueueService"/>

  <type name="WorkerQueue" class="com.talker.BWorkerQueue"/>
  <type name="BooleanIntervalTcpExt" class="com.talker.BBooleanIntervalTcpExt"/>
  <type name="NumericIntervalTcpExt" class="com.talker.BNumericIntervalTcpExt"/>
  <type name="EnumIntervalTcpExt" class="com.talker.BEnumIntervalTcpExt"/>
  <type name="StringIntervalTcpExt" class="com.talker.BStringIntervalTcpExt"/>


  <type name="Equip" class="com.hayslot.BEquip"/>
  <type name="HaystackSlot" class="com.hayslot.BHaystackSlot"/>
  <type name="HaystackTags" class="com.hayslot.BHaystackTags"/>

   <type name="HaystackSlotFE" class="com.hayslot.ui.BHaystackSlotFE">
    <agent>
        <on type="myTcpDataSender:HaystackSlot" />
    </agent>
  </type>
```

Obrázek 20 - Soubor module-include.xml

Jedná se o xml strukturu s hlavním tagem `types`, který obsahuje elementy `type`. Niagara Type je specifikovaný atributem `name`, zdrojová Java třída je specifikovaná atributem `class`. Je pravidlem, že při pojmenovávání třídy volíme jako první písmeno "B" (odkazuje na baja standard), v názvu typu pak toto písmeno vynecháváme.

Mezi Niagara *Types* může uživatel explicitně definovat vzájemné vazby pomocí Niagara *Agents. Agent* se obvykle využívá pro tvorbu různých workbench zobrazení. Tato vazba je nastavena elementem `agent` ,ve kterém je specifikován cílový *Type*. Informace o Agentech je uložena v Registry, cílové *Types* tedy o svých Agentech "neví" (je to z důvodu umožnění vytváření Agentů i na *Types*, k jejichž zdrojovým kódům nemá uživatel přímý přístup).

5.5.3 module.palette a module.lexicon

Tyto soubory jsou volitelné soubory Niagara modulu. V `module.palette` jsou specifikovány Niagara *Types,* které jsou pro uživatele přístupné v paletě nástrojů ve vývojovém prostředí (například v COACH AX). Záznamy obsahují atribut `n` který odpovídá zobrazovanému jménu v paletě nástrojů a atribut `t` který udává Niagara *Type* včetně modulu. Soubor `module.lexicon` slouží k lokalizaci určitých částí modulu (pojmenování slotů v interface, statusy, chybové hlášky atd).

```
<?xml version="1.0" encoding="UTF-8"?>
<bajaObjectGraph version="1.0">
<p m="b=baja" t="b:UnrestrictedFolder">

 <p m="mtds=myTcpDataSender" n="CommonQueueService"
    t="mtds:CommonQueueService" />


 <p n="BooleanIntervalTcp" t="mtds:BooleanIntervalTcpExt" />
 <p n="EnumIntervalTcp" t="mtds:EnumIntervalTcpExt" />
 <p n="NumericIntervalTcp" t="mtds:NumericIntervalTcpExt" />
 <p n="StringIntervalTcp" t="mtds:StringIntervalTcpExt" />

  </p>
</bajaObjectGraph>
```

Obrázek 21 - Soubor module.palette

5.6 Vytvoření služby

5.6.1 Úvod

Služba v rámci Niagara frameworku poskytuje globální funkcionalitu pro další komponenty. Oproti obyčejným komponentám se služba liší v tom, že při startu stanice jsou vždy služby spouštěny pomocí `serviceStarted()` callbacku před komponentami a při zastavení stanice jsou zastaveny jako poslední pomocí `serviceStopped()` callbacku. Služby jsou spouštěny v pořadí odpovídajícímu pořadí shora dolů v jejich component space stanice (tzn. tak, jak jsou umístěny v adresáři services ve stanici).

Dalším rozdílem je *Type* u služeb. Služba může být, na rozdíl od komponent, registrována pod více než jedním *Type*. Je to z důvodu zpětné kompatibility, kdy k novým službám které nahrazují své starší verze může být přiřazen i *Type* původní služby. Každá služba musí obsahovat metodu `getServiceTypes()`, která vrací pole jednotlivých *Type*. Je důležité aby návratová hodnota byla pole datového typu *Type*, jinak může dojít k problémům při registraci služby a následně s jejím spouštěním. Z pohledu Frameworku jsou služby komponenty, které implementují rozhraní `BIService` nebo jsou potomky třídy `BAbstractService`.

5.6.2 Realizace služby pro odesílání dat

Služba `BCommonQueueService` obsahuje statické metody pro uložení záznamů z jednotlivých komponent do pole bytů. Zděděné jsou property sloty *enabled* a *status*, *enabled* umožňuje vypnutí a zapnutí služby a *status* zobrazuje stav služby. Přidané jsou *sloty portNumber, queueSize* a *hostIP*. Slot *portNumber* a *hostIP* specifikuje port a IP (verze 4) adresu, na kterou jsou posílány shromážděná data. Slot *queueSize* umožňuje nastavit počet záznamů, které budou shromážděny před odesláním.

Zvolený způsob hromadného odesílání dat slouží k šetření prostředků. Odesílání dat je prováděno na vedlejším vlákně. Je to z toho důvodu, aby nedocházelo ke zpomalování hlavního vlákna (*Control Engine Thread*). Hlavní vlákno je monitorováno systémovým watchdogem, a pokud každých 60 sekund nepřijde odpověď z hlavního vlákna, tak dojde k vynucenému restartu stanice.

Ve statické metodě `postWork()` je obsah bytového pole, adresy, a portu použitý k vytvoření nové instance třídy `RunnableTcpWork` která implementuje rozhraní `Runnable`. V metodě run třídy `RunnableTcpWork` je vytvořen nový TCP/IP socket s nastavenou adresou a portem. Pokud se nepodaří data odeslat, je služba `BCommonQueueService` vypnuta a zobrazena chybová hláška ve statusu. Instance třídy `RunnableTcpWork` jsou shromažďovány do fronty třídy `BWorkerQueue`, která spouští vedlejší pracovní vlákno.

5.6.3 Poznámka k řešení

Statický přístup k celé třídě přináší z pohledu Niagara Frameworku určité problémy. Ty vycházejí z implementace Property slotů (a slotů obecně). Sloty jsou totiž definované jako statické členy, ale přístupové gettery a settery jsou instančního charakteru. Například slot *portNumber*:

```
public static final Property portNumber = newProperty(Flags.SUMMARY, 6667,null);
```

obsahuje instanční přístupové metody:

```
public int getPortNumber() { return getInt(portNumber); }
public void setPortNumber(int v) { setInt(portNumber,v,null); }
```

Když uživatel přidá službu do stanice, vytvoří se její nová instance. To znamená, že například ze statické metody `postWork()` nemůžeme přistupovat k vlastnosti *PortNumber*. Toto je vyřešeno vytvořením statických proměnných, do kterých se při spouštění služby a při každé změně některého slotu ukládají nové

hodnoty. K ukládání hodnot dochází v metodě `serviceStarted()` která je volána při spouštění stanice, a v metodě `changed()`, která je volána pokud dojde ke změně hodnoty v některém ze slotů služby.

5.7 Vytvoření rozšíření pro komponenty

Doplňující funkce k datovým bodům jsou v Niagara Frameworku řešeny pomocí rozšíření. Existují rozšíření pro alarmy, integrátory a čítače, a trendy. Všechny typy rozšíření dědí ze společné třídy `BPointExtension`. Rozšíření mohou být použity pouze na komponenty které jsou potomky třídy `BControlPoint`.

Vytvořené rozšíření jsou potomky rozšíření pro trendy. Celkem byly vytvořeny 4 rozšíření pro 4 základní datové typy Niagara frameworku.

```
public class BBooleanIntervalTcpExt
    extends BBooleanIntervalHistoryExt

public class BEnumIntervalTcpExt
    extends BEnumIntervalHistoryExt

public class BNumericIntervalTcpExt
    extends BNumericIntervalHistoryExt

public class BStringIntervalTcpExt
    extends BStringIntervalHistoryExt
```

Kromě slotů zděděných z třídy předka byly přidány property sloty typu *HaystackSlot* a *Uuid*. *Uuid* přiřazuje každému vytvořenému rozšíření unikátní identifikátor. *HaystackSlot* ukládá zadaná metadata pro dané rozšíření.

Všechny vytvořené rozšíření obsahují zděděnou metodu `protected void writeRecord(BAbsTime arg0, BStatusValue arg1)`. Je to implementace abstraktní metody z třídy předka `BHistoryIntervalExt`. Metoda je volána po uplynutí uživatelem zadaného časového intervalu pro sběr dat v property pohledu u daného rozšíření.

Tato metoda je v rozšířeních přepsána. Při jejím volání je nejdříve zkontrolováno, jestli je stanice ve *Steady State* a jestli je rozšíření povoleno a běží vytvořená služba `BCommonQueueService`. Pokud jsou tyto podmínky splněny, je zkontrolováno jestli nedošlo ke změně v metadatech uložených v *HaystackSlot* slotu. Pokud ano, jsou nová metadata přidány do bytového pole statickou metodou služby `enqueConfig()`. Naposledy je přidán nový záznam obsahující časovou značku a hodnotu do bytového pole statickou metodou služby `enqueVal()`.

5.8 HaystackSlot a vytvoření pohledu

Každé rozšíření obsahuje property slot typu *HaystackSlot*, ve kterém jsou ukládány uživatelem přiřazená metadata podle Haystack modelu pro jednotlivé zařízení. Pro typ *HaystackSlot* byl vytvořen workbench pohled, který uživateli umožňuje snadno vybrat metadata z předpřipravených sad markerů pro jednotlivé zařízení definovaných podle Haystack modelu.

Byla vytvořena pomocná třída `BEquip` která obsahuje enumeraci pro jednotlivé typy zařízení a třída `BHaystackTags` která obsahuje enumeraci pro všechny Haystack tagy. Tyto třídy jsou potomky třídy `BFrozenEnum`. Sady tagů jsou uloženy jako textové soubory, a jsou definované jako resources pro daný modul v souboru `build.xml`. Byly použity sady tagů dostupné z [14].

Typ *HaystackSlot* obsahuje 3 property sloty. Sloty *Devicename* a *Devicetags* slouží k uložení reference zařízení a markerů u daného datového bodu. Slot *DeviceIndex* ukládá index který odpovídá jednomu z 13 typů zařízení definovaných v Haystack modelu. Dále obsahuje metodu `getCurrentTags(),` která vrací String v předdefinovaném formátu pro odesílání metadat.

Uživatel nemá možnost přímo zadávat data do jednotlivých property slotů. K zadávání metadat slouží vytvořený pohled typu HaystackSlotFE, který je agentem na typu HaystackSlot (definované v souboru module-include.xml).

5.8.1 Vytvoření workbench pohledu

Byl vytvořen workbench pohled pro zadávání metadat. Pro vytvoření pohledu byla vytvořena nová třída `BHaystackSlotFE`. Předkem je třída `BWbFieldEditor`, která na rozdíl od klasického workbench pohledu nevyplní celou pracovní plochu, a proto se typicky používá například jako součást property pohledu kde slouží k vytvoření uživatelského rozhraní pro zadávání dat například do určité struktury.

Třída `BWbFieldEditor` definuje 3 metody které jsou využívány k řízení pohledu. Metoda `doLoadValue(BObject value, Context cx)` a `doSaveValue(BObject value, Context cx)` slouží načtení pohledu z objektu a uložení hodnot do objektu z pohledu. Tento objekt odpovídá *Type*, na kterém je daný pohled zaregistrovaný jako Agent. Třetí metoda `setModified()` vyvolá upozornění o provedených změnách a umožní uložení pohledu.

Samotný pohled *HaystackSlotFE* je vytvořen z prvků uživatelského rozhraní obsažených v modulu *bajaui*. Mezi tyto prvky patří panely (které slouží pro umístění dalších prvků), tlačítka, textová pole, nabídky, atd. Následující popis třídy `BHaystackSlotFE` obsahuje kroky pro vytvoření pohledu:

1. Vytvoření proměnných pro uložení prvků uživatelského rozhraní.
2. Do nadřazeného prvku třídy `BGridPane` jsou umístěny další prvky pro ovládání uživatelského rozhraní (třídy `BButton, BLabel, BTextField, BListDropDown`).
3. Vytvoření Action slotů pomocí nástroje Slotomatic pro obsluhu událostí na prvcích uživatelského rozhraní

```
public class BHaystackSlotFE
    extends BWbFieldEditor
{

  /*-
  class BHaystackSlotFE
  {
  actions
   {
     dropSelect()
     tagsSetSelect()
     addTagAction()
     removeTagsAction()
     buttonClicked(event: BMouseEvent) default{[ new BMouseEvent() ]}
   }
  }
      -*/
```

Obrázek 22 - Vytvoření Action slotů pomocí nástroje Slotomatic

4. Vytvoření callback metod pro jednotlivé Action sloty. Název metody musí obsahovat předponu "do" (například akce `addTagAction()` - metoda `doAddTagAction()`).
5. Vytvoření Action-Action a Topic-Action vazeb (Links) pro jednotlivé prvky uživatelského rozhraní.

```
// link UI elements to their actions
this.linkTo(dropequip, BListDropDown.listActionPerformed, dropSelect );
this.linkTo(tagsSet, BListDropDown.listActionPerformed, tagsSetSelect);
this.linkTo(devicename, BTextField.textModified, BWbPlugin.setModified);
this.linkTo(addTag, BButton.actionPerformed, addTagAction);
this.linkTo(clearTags, BButton.actionPerformed, removeTagsAction);
```

Obrázek 23 - Vytvoření vazeb

6. Přepsání callback metod `doLoadValue(BObject value, Context cx)` a `doSaveValue(BObject value, Context cx)` pro uložení a načtení pohledu z připojeného typu *HaystackSlot*.

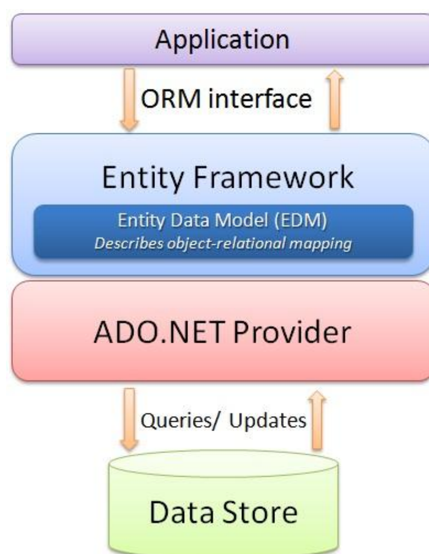
6 APLIKACE PRO PŘÍJEM DAT

6.1 Úvod

Funkce aplikace pro příjem dat spočívá v naslouchání na zvoleném TCP portu a zařazení přijatých dat do mySQL databáze. Aplikace je vytvořena v programovacím jazyku C#, pro práci s databází je použito Entity Frameworku který uživateli umožňuje pracovat pouze s třídami a objekty a Entity se stará o generování SQL dotazů.

6.2 Entity Framework

Entity Framework poskytuje ORM (objektově relační mapování) pro automatickou konverzi dat z objektové reprezentace do relační databáze. Využívá technologie ADO.NET pro přístup k datům.



Obrázek 24 - Schéma Entity Frameworku [23]

Entity Data Model (EDM) je klíčovou součástí Entity Frameworku, protože specifikuje vztahy mezi objekty a relační databází. Vytvořením modelu dochází k oddělení aplikace od konkrétní databáze. Výsledkem je koncepční model, který popisuje jednotlivé entity a vztahy mezi nimi. Model lze vytvořit třemi základními způsoby:

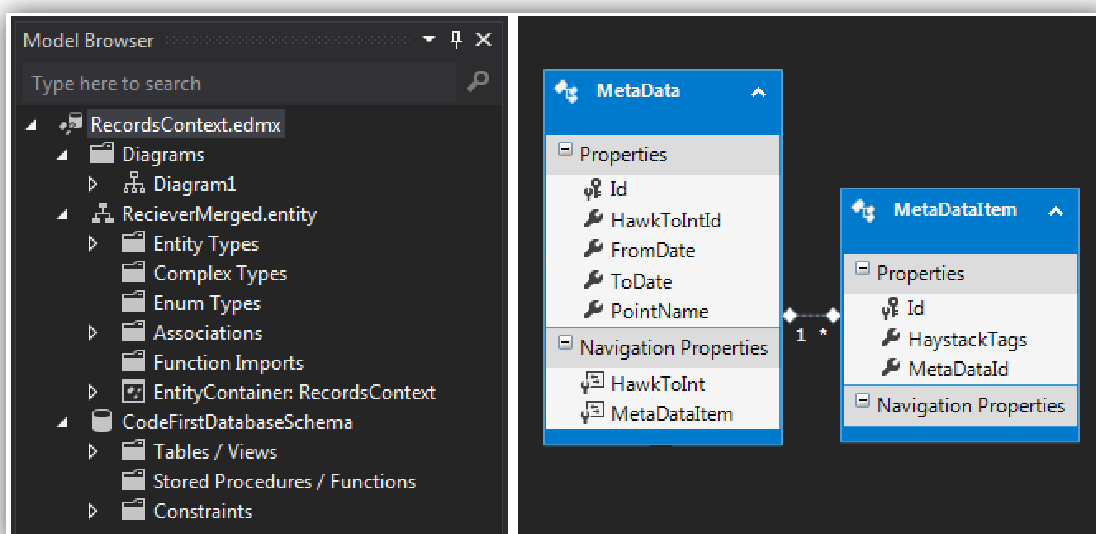
1. **Database First** - model je vytvořen z existující databáze.

2. **Model First** - uživatel ručně vytvoří model, definuje jednotlivé entity a vztahy mezi nimi. Tento přístup je vhodný pokud cílová databáze ještě nebyla vytvořena.

3. **Code First** - model je vytvořen z uživatelem definovaných tříd.

Entity Data Model může být prohlížen a editován z vývojového prostředí pomocí prohlížeče modelu. Složka Diagrams obsahuje grafické zobrazení diagramu entit a vztahů mezi nimi. Složka Entity Types obsahuje třídy, které jsou mapované jako entity. Složka Associations obsahuje definice cizích klíčů pro jednotlivé entity. Databázové schéma definuje ukládání entit do tabulek databáze.

Kompatibilitu datového modelu EDM a databáze zajišťují migrace. Uživatel v určité fázi projektu vytvoří migraci. Entity framework vytvoří ve složce "Migrations" příslušnou třídu, která obsahuje metody `Up()` a `Down()`. Tyto metody obsahují seznam změn, které budou v databázi provedeny. Po připojení k databázi je možné aplikovat některou z migrací, což znamená že jsou v databázi vytvořeny tabulky a klíče odpovídající modelu pro nahranou migraci. Mezi jednotlivými verzemi migrací lze jednoduše přecházet v obou směrech, tedy od starší verze k novější a naopak.



Obrázek 25 - Prohlížení modelu, Entity a vztahy

6.3 Třída DBContext

Další důležitou součástí Entity Frameworku je třída `DBContext.` Při vytváření Database First nebo Model First je kontext vygenerován, při vytváření Code First je tato třída vytvořena programátorem.

Kontext obsahuje proměnné typu DBset, které jsou mapované na jednotlivé tabulky v databázi. Kontext se využívá pro základní operace nad jednotlivými entitami (vytvoření, smazání, editace, čtení). Provedené změny v jednotlivých entitách jsou ukládány do mezipaměti kontextu (po celou dobu života kontextu). Kvůli stálému ukládání změn do mezipaměti je vhodné používat kontexty s klíčovým slovem using, které zajistí odstranění objektu kontextu.

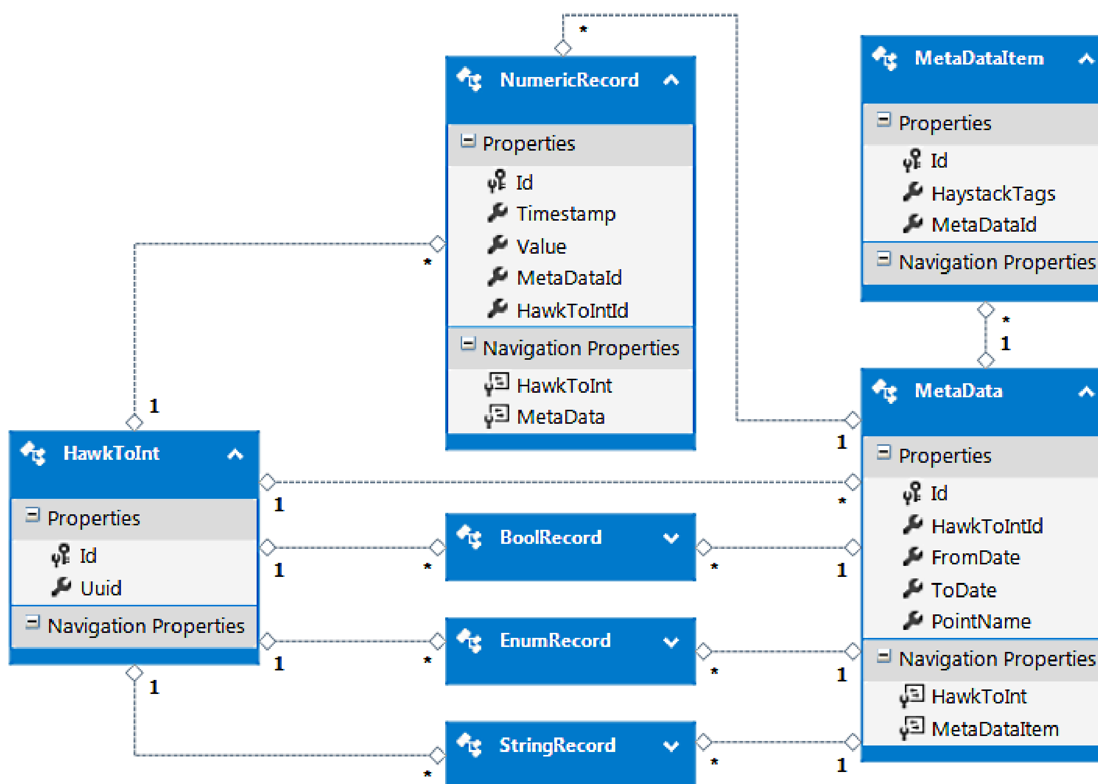
6.4 Návrh databáze

Model relační databáze v Entity Frameworku pro ukládání dat z řídicího systému byl navržen metodou Code First. Byly vytvořeny následující třídy:

- Třídy `EnumRecord, BoolRecord, StringRecord` a `NumericRecord` pro uložení časové značky a hodnoty.
- Třída `HawkToInt` pro převod identifikačního stringu na integer.
- Třída `MetaData` pro uložení platných metadat k jednotlivým bodům.
- Třída `MetaDataItem` pro uložení konkrétních tagů

V těchto třídách jsou definované vzájemné vztahy mezi entitami. Jednotlivé entity odpovídají jednotlivým tabulkám v databázi, vlastnosti entit odpovídají jednotlivým sloupcům tabulky. Každý nový záznam je novým řádkem tabulky.

Při návrhu databáze pro ukládání přijatých dat byla snaha oddělit uložení měřených dat z jednotlivých datových bodů od metadat těchto bodů. Zvolený způsob ukládání dat šetří počet zápisů, protože každému záznamu (třídy Records) jsou přiřazeny pouze jedny (v ten okamžik platná) metadata. Jednotlivé tagy jsou již odděleně ukládány do tabulky MetaDataItems, a jsou navázány k metadatům. Vztahy mezi vytvořenými entitami znázorňuje nejlépe následující diagram:



Obrázek 26 - Entity model, vztahy tabulek databáze

Entita HawkToInt je využívána k přiřazení klíče Id typu Integer k dlouhému (pro práci s databází nevhodnému) identifikačnímu stringu Uuid. Unikátní identifikační string je získaný z jednotlivých rozšíření pro sběr dat. Každý řádek v tabulce entity HawkToInt tak odpovídá jednomu unikátnímu rozšíření pro sběr dat.

K tomuto nově získanému Id jsou přiřazeny vztahem one-to-many entity záznamů měřených dat pro jednotlivé datové typy (tabulky entit NumericRecord, BoolRecord, EnumRecord, StringRecord). Každý řádek v tabulce záznamů ukládá časovou značku, hodnotu, odkaz na cizí klíč Id v tabulce HawkToInt, a odkaz na cizí klíč Id v tabulce MetaData.

Tabulka entity MetaData slouží k ukládání platných metadat u jednotlivých záznamů. Je ve vztahu více Records k jednomu MetaData záznamu. Tabulka MetaData neukládá přímo jednotlivé tagy, ale slouží k uložení časového intervalu, na kterém jsou daná metadata platná. Každý řádek obsahuje klíč Id metadat, název datového bodu, na kterém je rozšíření pro sběr dat umístěno, dobu začátku a konce platnosti metadat FromDate a ToDate, a odkaz na cizí klíč Id v tabulce HawkToInt.

Jednotlivé tagy jsou uloženy v tabulce entity MetaDataItem. S tabulkou entity MetaData je ve vztahu one MetaData to many MetaDataItem. Každý řádek ukládá tag HaystackTags, odkaz na Id tabulky MetaData a klíč Id.

Pro přístup k zmíněným tabulkám byla vytvořena třída `RecordContext`, která dědí z třídy `DBContext`. Jsou zde definované jednotlivé tabulky databáze podle vytvořených tříd.

```
class RecordsContext : DbContext
    {
        public RecordsContext() : base("mysqllocalconnection") {
}

        public DbSet<NumericRecord> NumericRecords { get; set; }
        public DbSet<BoolRecord> BoolRecords { get; set; }
        public DbSet<EnumRecord> EnumRecords { get; set; }
        public DbSet<StringRecord> StringRecords { get; set; }
        public DbSet<MetaData> MetaDatas { get; set; }
        public DbSet<MetaDataItem> MetaDataItems { get; set; }
        public DbSet<HawkToInt> HawkToIDs { get; set; }
 }
```

6.5 Vytvoření aplikace

Pro vytvoření aplikace bylo využito technologie WPF. Takto bylo vytvořeno základní uživatelské rozhraní, ve kterém uživatel nejdříve spustí naslouchání na zvoleném portu a poté se připojí k databázi. Adresa, název a přístupové údaje k databázi jsou nastavitelné v konfiguračním souboru.

Byly vytvořeny třídy `DbWriter` a `CAsyncCom`. Třída `CAsyncCom` zajišťuje otevření asynchronního TCP socketu pro poslouchání na zvoleném portu. Poslouchání je spuštěné na novém vlákně, aby při spuštění nedošlo k zamrznutí uživatelského rozhraní. Přijaté data jsou řazeny do thread-safe fronty `dbdata`.

Třída `DbWriter` zabaluje třídu `RecordContext` do několika funkcí, která slouží převážně k ukládání dat do databáze. Mezi tyto funkce patří:

```
public bool containsHawkId(HawkToInt hawk)
public void addHawkId(HawkToInt bset)
public void addMetaData(MetaData bMetaData)
public void addMetaDataItems(string bname, string bhawkId,
string btag)
public void addNumericRecord(string bhawkId, string bname, long
btimestamp, double bvalue)
```

```
public void addBoolRecord(string bhawkId, string bname, long btimestamp, bool bvalue)
public void addEnumRecord(string bhawkId, string bname, long btimestamp, int bvalue)
public void addStringRecord(string bhawkId, string bname, long btimestamp, string bvalue)
```

V každé funkci pro přidání dat je vytvořena lokální instance třídy `RecordContext`, která po vykonání funkce zaniká. Při spuštění připojení k databází je vytvořeno nové vlákno. Pokud se databáze neshoduje s aktuální migrací Entity Frameworku, je aplikována nejnovější migrace. Na tomto vlákně jsou v nekonečné while smyčce kontrolovány nové data ve frontě `dbdata.` Pokud jsou k dispozici nové data, je určen jejich typ (podle datových Niagara typů Enum, Bool, Numeric nebo String, v případě změny metadat vytvořený typ Config), a jsou uloženy do databáze.

7 ZÁVĚR

Hlavním cílem práce bylo realizovaní sběru dat z regulátoru a integrátoru HAWK, který je součástí integrovaného řídicího systému pro automatizaci budov CentraLine. Sbíraná data měla být ukládána do externí databáze, k tomu účelu byl zvolen databázový systém MySQL.

Prvotním záměrem bylo vytvořit komponenty v Niagara Frameworku (na němž je řídicí systém CentraLine postaven), které budou realizovat sběr a odesílání dat do externí MySQL databáze. Po nastudování možností práce a vývoje v Niagara Frameworku muselo být od tohoto záměru z technických důvodů upuštěno. Komponenty vytvořené v Niagara Frameworku jsou totiž primárně určeny pro platformu Javy Java2ME. Tato platforma Javy neumožňuje použití JDBC driveru pro MySQL databázi. Niagara Framework umožňuje kompilovat komponenty i pro platformu Java2SE, ale tyto komponenty nemohou být spuštěny v regulátoru HAWK, který podporuje pouze Java2ME aplikace.

Z tohoto důvodu bylo nakonec řešení rozděleno na 2 části. V první části byl s použitím aplikačního rozhraní Niagara Frameworku vytvořen programový modul, který obsahuje komponenty pro sběr dat z datových bodů pro každý datový typ Niagara Frameworku, a službu pro odesílání shromážděných dat přes zvolený TCP socket.

Pro jednotlivé komponenty pro sběr dat byl dále vytvořen uživatelský pohled, který umožňuje zadávání tagů k jednotlivým datovým bodům. Tyto tagy jsou převzaty z opensourcové iniciativy Project Haystack, která usiluje o vytvoření a standardizaci sémantických modelů (zejména pro HVAC odvětví). Postup tvorby modulu, komponent, a služby je uveden v kapitole 5, v předcházející kapitole 4 jsou vysvětleny základní pojmy, se kterými se uživatel při práci v Niagara Frameworku setkává.

V druhé části řešení byla vytvořena aplikace pro příjem dat a jejich ukládání do databáze. Aplikace byla vytvořena v programovacím jazyce C# s využitím Entity Frameworku, který poskytuje ORM (Objektově-relační mapování) pro automatickou konverzi dat z objektové do databázové reprezentace.

8 POUŽITÁ LITERATURA

[1] TRIDIUM. *Niagara Concepts Guide*. Richmond, Virginia, 2002. Dostupné také z: http://www.hvacc.net/pdf/tridium/docs_2.3.522/JDE/Niagara_Concepts_Guide/NiagaraConceptsGuide.pdf

[2] MALTZ, Václav. *Nová generace řídicích systémů Honeywell CentraLineAX* [online prezentace]. [cit. 2017-01-02]. Dostupné z: http://www.ib.cvut.cz/sites/default/files/Honeywell_prednasky/3a_prezentace.pdf

[3] TRIDIUM. *Tridium and Third Party Drivers*. 2014. Dostupné také z: http://www.tridium.com/~/media/tridium/niagara%20ax/documents/tridium_driverslist_1413477589.ashx?la=en

[4] *Niagara AX best practices FAQ*. 2008. Dostupné také z: http://tech.ct-supply.com/Training%20Material/NiagaraAx-Best-Practices-FAQ.pdf

[5] HONEYWELL. *Engineering manual of automatic control for commercial buildings*. Minneapolis, Minnesota, 1997.

[6] *Grafická centrála ARENA AX* [online]. [cit. 2017-04-03]. Dostupné z: http://products.centraline.com/cz/ecatdata/pg_clarenaax.html

[7] *ARENA AX – BACnet centrála: Katalogový list*. 2014. Dostupné také z: http://products.centraline.com/cz/pdf/en0z0945-cz01r0314.pdf

[8] *Corporate info: About us* [online]. [cit. 2017-04-03]. Dostupné z: https://www.tridium.com/en/about-us/corporate-info

[9] *JSR 60: Building Automation Java API* [online]. [cit. 2017-04-03]. Dostupné z: https://jcp.org/en/jsr/detail?id=60

[10] TRIDIUM. *NiagaraAX-3.2 Developer Guide*. 2007. Dostupné také z: http://www.hvacc.net/pdf/tridium/docs_3.2.16/docDeveloperPDF/docDeveloperPDF_3_2.pdf

[11] *Eclipse Niagara Plugin: Eclipse Niagara Plugin adds support for the 3.7U1 environment* [online]. [cit. 2017-04-03]. Dostupné z: http://niagara-3-8-38.software.informer.com/

[12] *Eclipse Classic 4.2.2* [online]. [cit. 2017-04-03]. Dostupné z: http://www.eclipse.org/downloads/packages/eclipse-classic-422/junosr2

[13] PETZE, John. *Project Haystack Extends Outreach with Educational Webinars* [online]. [cit. 2017-05-03]. Dostupné z: http://www.automatedbuildings.com/news/dec13/articles/haystack/131126113003haystack.html

[14] *Downloads - Project Haystack* [online]. [cit. 2017-05-03]. Dostupné z: http://project-haystack.org/download

[15] Structure - Project Haystack. *Project Haystack* [online]. [cit. 2017-05-03]. Dostupné z: http://project-haystack.org/doc/Structure

[16] NEWMAN, Mike. *Oral and Personal Histories of Computing at Cornell: Recollections* [online]. [cit. 2016-01-05]. Dostupné z: http://www2.cit.cornell.edu/computer/history/Newman.html

[17] Shengwei Wang. *Intelligent Buildings and Building Automation.* London: Spon Press, 2010. ISBN 0-203-89081-7.

[18] NEWMAN, H. Michael a M. D MORRIS. *Direct digital control of building systems: theory and practice*. Wiley, 1994. ISBN 0471516961.

[19] MERZ, Hermann, Thomas HANSEMANN a Christof HÜBNER. *Automatizované systémy budov: sdělovací systémy KNX/EIB, LON a BACnet*. 1. vyd. Praha: Grada, 2008, 261 s. : il. ; 24 cm. ISBN 978-80-247-2367-9.

[20] KASTNER, W., G. NEUGSCHWANDTNER, S. SOUCEK a H.M. NEWMANN. Communication Systems for Building Automation and Control. *Proceedings of the IEEE* [online]. USA: IEEE, 0050n. l., 93(6): 1178-1203 [cit. 2016-01-02]. DOI: 10.1109/JPROC.2005.849726. ISSN 0018-9219.

[21] PÉREZ-LOMBARD, Luis, José ORTIZ a Christine POUT. A review on buildings energy consumption information.*Energy & Buildings* [online]. 2008, 40(3): 394-398 [cit. 2017-01-02]. DOI: 10.1016/j.enbuild.2007.03.007. ISSN 0378-7788.

[22] TRIDIUM. *Niagara AX User Guide. Richmond, Virginia, 2007.* Dostupné také z: http://www.hvacc.net/pdf/tridium/docs_3.2.16/docUser/docUser.pdf

[23] *Entity Framework 6* [online]. *Copyright © 2016 by EntityFrameworkTutorial.net. All Rights Reserved, 2016 [cit. 2017-04-13].* Dostupné z: http://www.entityframeworktutorial.net/entityframework6/introduction.aspx

Seznam příloh

Příloha 1. Postup při ověření funkčnosti řešení
Příloha 2. CD/DVD s modulem pro Niagara Framework, aplikací pro příjem dat a zdrojovými soubory

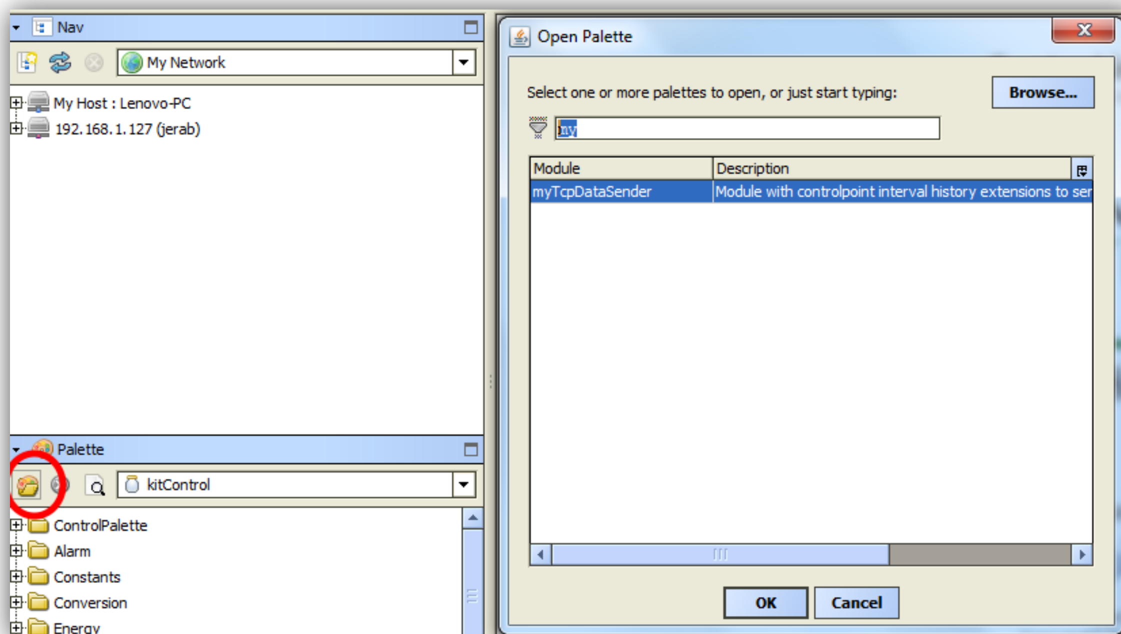
Příloha 1 - postup při ověření funkčnosti řešení

Tato příloha obsahuje pracovní postup pro ověření funkčnosti vytvořeného Niagara modulu myTcpDataSender a ověření funkčnosti aplikace pro příjem dat.

Testování modulu myTcpDataSender

Vytvořený modul obsahuje komponenty rozšíření pro sběr dat z datových bodů a službu pro odesílání těchto dat přes zvolený TCP port do aplikace pro příjem dat. Tento modul byl použit v nově vytvořené stanici spuštěné na platformě osobního počítače (Lenovo ThinkPad 540). Níže jsou uvedeny jednotlivé kroky při testování s vysvětlujícími obrázky z pracovního prostředí COACH AX.

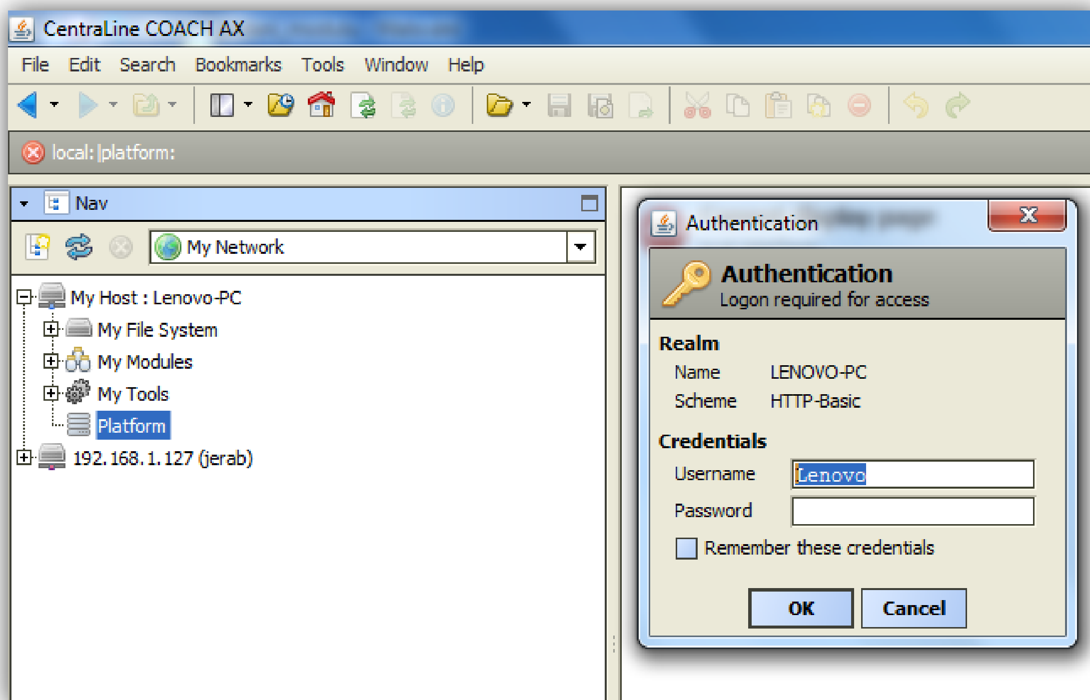
1. Zkopírování zdrojového souboru *myTcpDataSender.jar* do instalační složky /CentraLine/COACHAX-3.8.38/modules
2. Spustíme vývojové prostředí CoachAX. Spouštění může trvat déle než obvykle, dochází k novému sestavení Registry
3. Přidání vytvořeného modulu na paletu zobrazovaných modulů. Na paletě modulů vybereme ikonu Open Palette. Dojde k otevření nového okna, kde je vybrán modul myTcpDataSender.



Obrázek 27 - přidání modulu na paletu

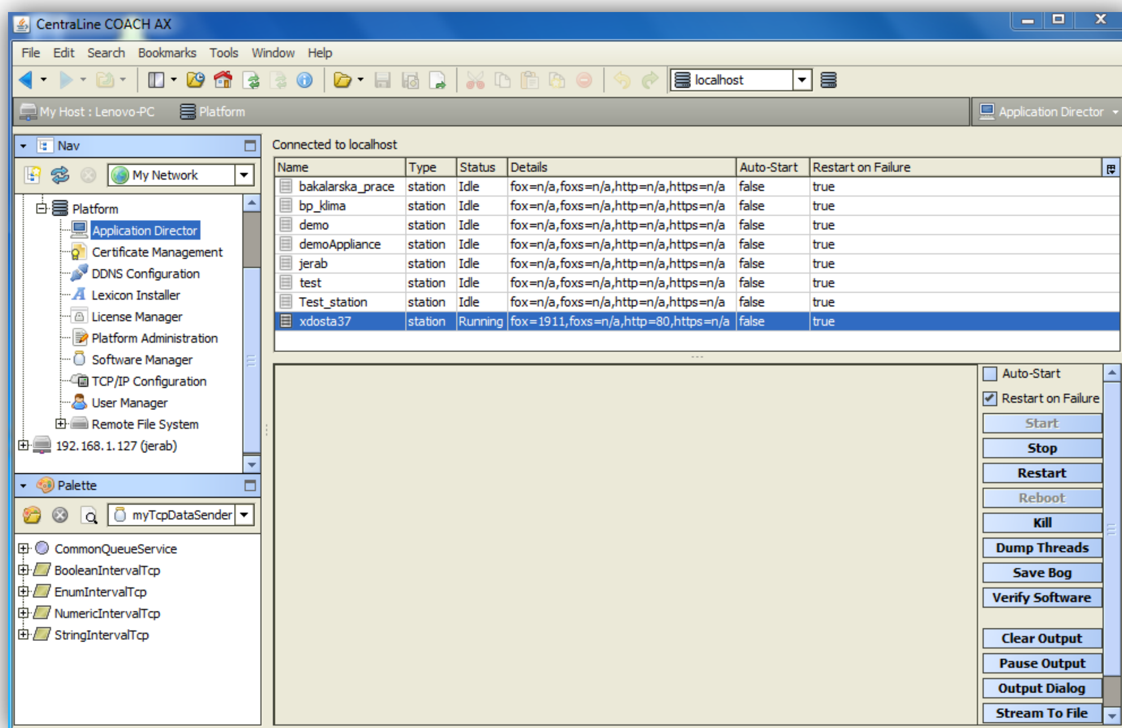
4. Připojení k platformě. V navigaci pracovního prostředí CoachAX otevřeme složku místní platformy (v tomto případě My Host:Lenovo-PC). Dvojklikem na záložku Platform se otevře dialog pro přihlášení k platformě. Jako

uživatelské jméno volíme jméno uživatele PC, heslo odpovídá heslu k PC. V případě připojování k regulátoru Hawk použijeme nastavené jméno a heslo (defaultní je uživatel tridium a heslo Niagara).



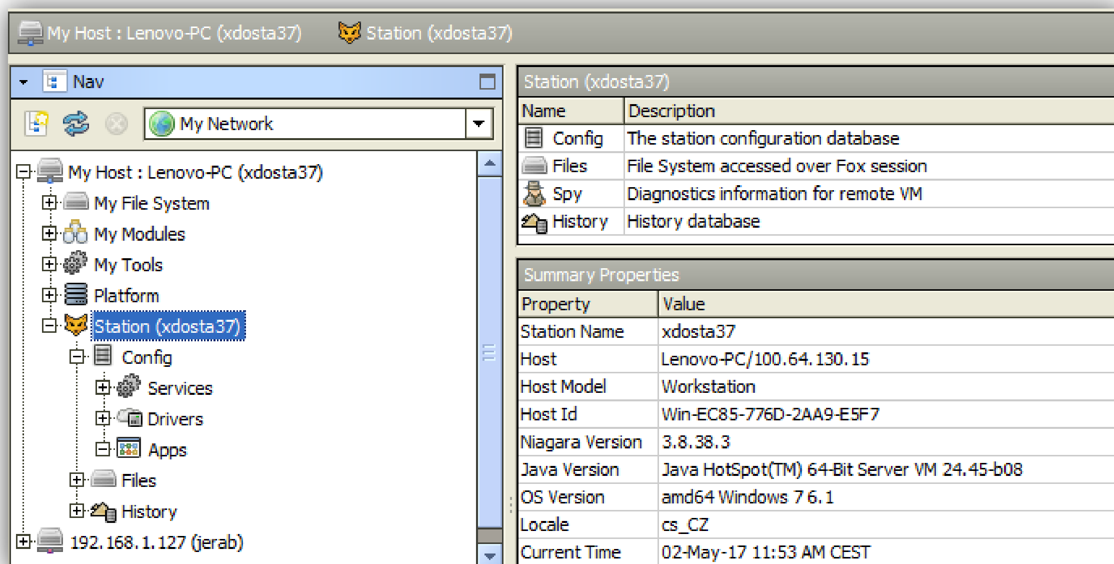
Obrázek 28 - Připojení k platformě

5. Vytvoření nové stanice. Z panelu nástrojů vybereme Tools. Zvolíme položku New Station a následujeme průvodce vytvoření stanice. Zadáme název stanice a heslo.
6. Spuštění nové stanice. V navigaci vybereme a otevřeme záložku Platform a vybereme záložku Application Director. Dojde k zobrazení všech stanic, které jsou na platformě k dispozici. Vybereme vytvořenou stanici (v našem případě xdosta37) a spustíme ji.



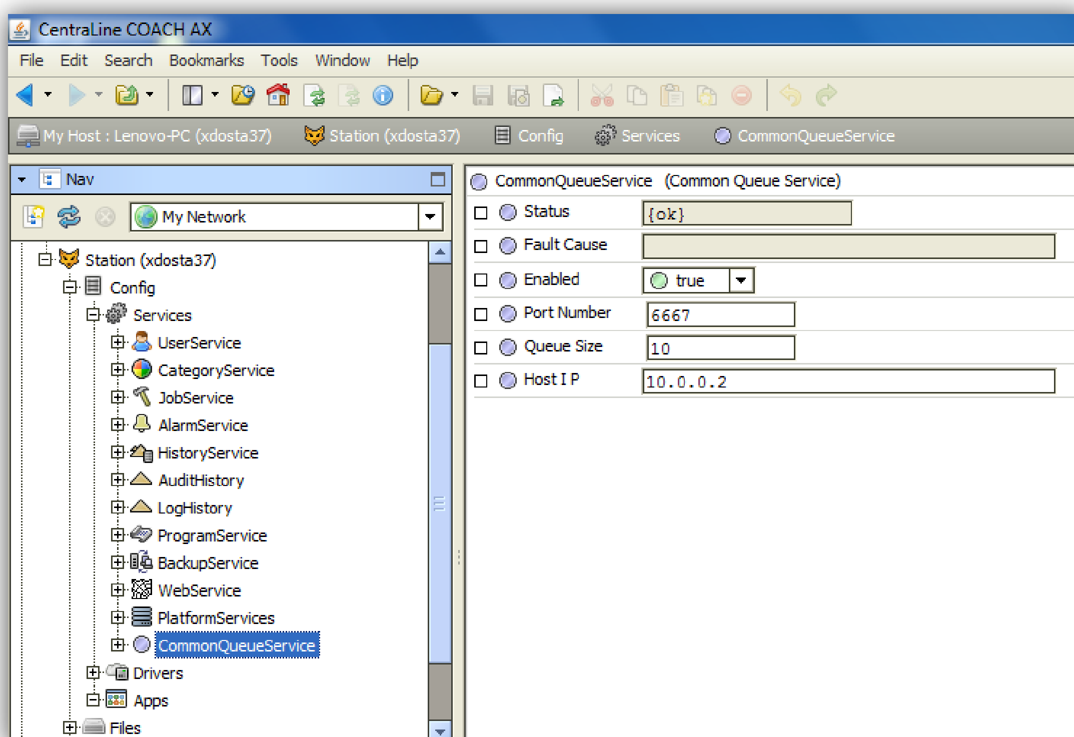
Obrázek 29 - spuštění stanice

7. Otevření připojení k nové stanici. Z panelu nástrojů vybereme File, Open, Open Station. V otevřeném okně vyplníme IP adresu stanice (v našem případě stačí použít localhost). Po zadání jména a hesla je otevřeno spojení se stanicí. Stanice je zobrazena v navigaci vývojového prostředí.



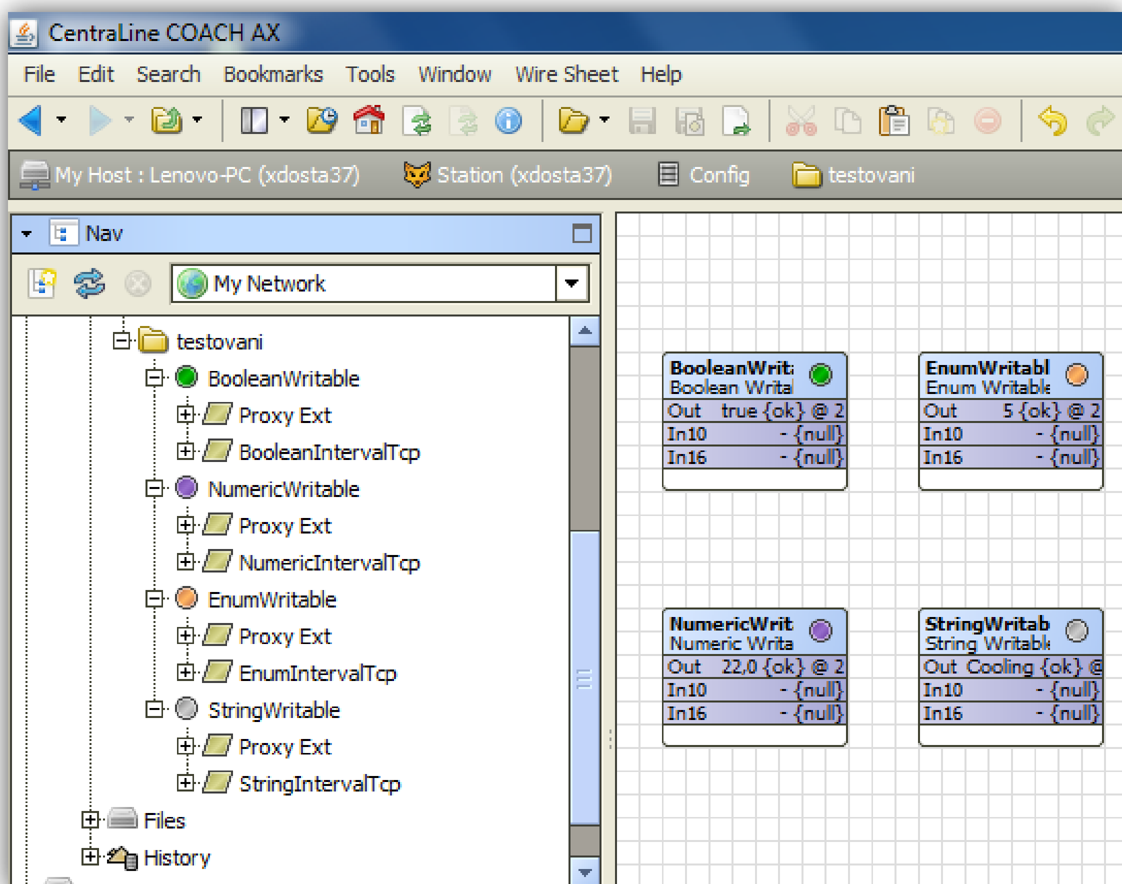
Obrázek 30 - Otevřená nová stanice

8. Přidání služby CommonQueueService do stanice. Z palety vybereme službu CommonQueueService a přetáhneme službu do složky Services v námi vytvořené stanici. Jméno služby ponecháme defaultní. Dvojklikem na přidanou službu se otevře Property view služby, kde uživatel může službu ručně vypnout nebo zapnout, nakonfigurovat port a IP adresu kam budou data odesílána, a nakonfigurovat velikost fronty.



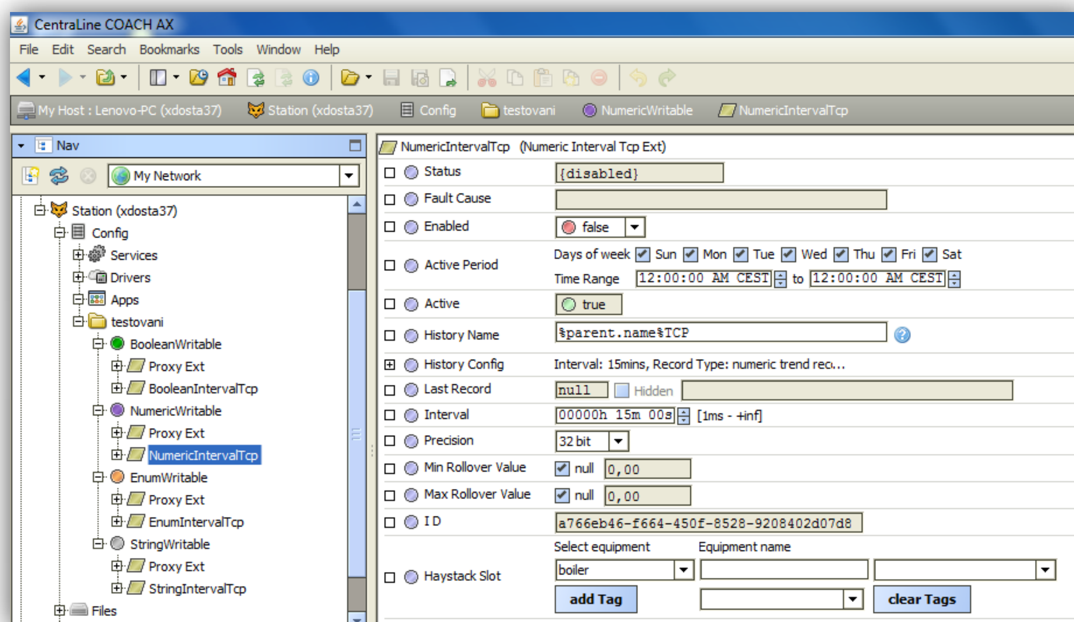
Obrázek 31 - Přidaná služba CommonQueueService, Property view

9. Pro testovací účely je nutné vytvořit datové body. Pod položkou Config ve stanici byla vytvořena složka (testovani). Do této složky umístíme datové body. V paletě vybereme modul control, a ze složky points umístíme pro každý datový typ požadovaný počet bodů (pro testování jsou použity body umožňující přímý zápis označené writable).
10. K vytvořeným datovým bodům přidáme komponenty rozšíření z modulu myTcpDataSender. To je opět provedeno přetažením jednotlivých rozšíření na vytvořené datové body. Rozšíření jsou zobrazeny v navigaci jako součást datového bodu.



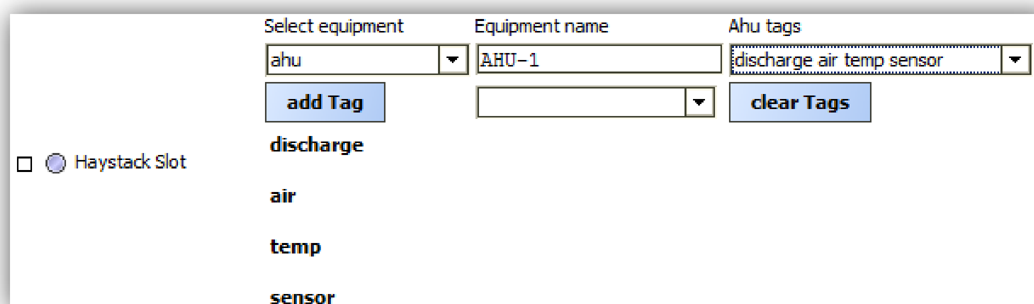
Obrázek 32 - Zobrazení rozšíření v navigaci

11. Konfigurace jednotlivých rozšíření je prováděna v Property view. Uživatel může nastavit interval a časový plán záznamu hodnot. Další nastavitelné parametry se liší podle datového typu bodu.



Obrázek 33 - Konfigurace rozšíření

12. Konfigurace tagů je prováděna pomocí vytvořeného pohledu. Tento pohled je u všech rozšíření stejný, zobrazuje se jako HaystackSlot v Property view. Uživatel může zvolit z předem připravených tagů rozdělených do skupin podle zařízení. Po zvolení skupiny tagů jsou tagy zobrazeny jako tlačítka v prvním sloupci. Tagy lze odstranit jednotlivě klepnutím na tyto tlačítka. Tagy lze také jednotlivě přidat, k čemuž slouží druhý řádek s tlačítkem "add Tag" které přidá vybraný tag. K hromadnému odstranění všech tagů slouží tlačítko "clear Tags".



Obrázek 34 - Pohled HaystackSlotFE na typu HaystackSlot

13. Po konfiguraci všech rozšíření můžeme jednotlivé rozšíření aktivovat nastavením vlastnosti Enabled na true. V této chvíli jsou sbíraná data ukládány do pole bytů ve službě CommonQueueService.

Testování aplikace pro příjem dat a jejich ukládání do databáze

Aplikace obsahuje jednoduché uživatelské rozhraní pro zadání TCP adresy a portu, a tlačítka pro spuštění naslouchání a připojení k databázi. Je spuštěna konzole kde jsou vypisovány dodatečné informace.

1. Před spuštěním aplikace je nejdříve nutné v konfiguračním souboru (název RecieverMerged.exe.config) změnit například v textovém editoru parametry v připojovacím řetězci. Uživatel zadá adresu a port databázového serveru, a uživatelské jméno a heslo pro toto připojení. Uživatel musí mít práva s plným přístupem.
2. Po spuštění aplikace uživatel zadá IP adresu a port na které je spuštěno naslouchání. Pokud jsou údaje zadané špatně, je pro pokračování nutné zadat platné údaje.
3. Po spuštění naslouchání je odemčeno tlačítko k připojení do databáze. (připojovací řetězec je editován externě). Pokud jsou údaje v připojovacím řetězci správné, je uživatel informován zprávou. V této chvíli již není možné do spuštěné aplikace zasahovat.
4. Při každém přijetí dat je uživatel informován zprávou v konzoli spuštěné aplikace.