SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: TODO: number FEI-XXXX-YYYYY

PREDIKTÍVNE METÓDY RIADENIA V PROSTREDÍ IOT

DIZERTAČNÁ PRÁCA

2016 **Anton Pytel**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: TODO: number FEI-XXXX-YYYYY

PREDIKTÍVNE METÓDY RIADENIA V PROSTREDÍ IOT

DIZERTAČNÁ PRÁCA

Študijný program:	TODO: program kybernetika/mechatronika
Číslo študijného odboru:	TODO: XXX cislo
Názov študijného odboru:	TODO: cislo a nazov programu
Školiace pracovisko:	Ústav automobilovej mechatroniky
Vedúci záverečnej práce:	prof. Ing. Štefan Kozák, PhD.

Bratislava 2016 **Anton Pytel**

ANOTÁCIA

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	TODO: program kybernetika/mechatronika
Autor:	Anton Pytel
Dizertačná práca:	Prediktívne metódy riadenia v prostredí IoT
Vedúci záverečnej práce:	prof. Ing. Štefan Kozák, PhD.
Miesto a rok predloženia práce:	Bratislava 2016

Hlavná myšlienka dizertačnej práce je aplikovať moderné metódy riadenia do IoT systému. Práca sa v prvej časti venuje matematickému opisu prediktívneho riadenia. Ďalej opisu podporného programu na simuláciu prediktívneho riadenia, ktorý bol vytvorený v prvej časti štúdia. Podporný program je vytvorený v prostredí Matlab-Guide a umožňuje testovať, ladiť a konfigurovať prediktívne algoritmy riadenia. Po objasnení prediktívneho riadenia sa práca zaoberá využitím znalostí z oblasti softvérovej architektúry na definovanie IoT systému, trendy v tejto oblasti a hlavne uskutočnenie hlavnej myšlienky a teda nájdenie optimálneho využitia IoT komponentov na umiestnenie prediktívneho regulátora do nich. Takto vzniká myšlienka regulátor ako služba (Controller as a service - CaaS), ktorá je v práci vysvetlená. Popísané sú výhody a nevýhody tohto prístupu riadenia. Myšlienka je overená v praxi na IoT systéme, ktorého súčasti sú v práci podrobne popísané. Po overení CaaS myšlienky reálnou implementáciou, je v práci ako vedľajšia myšlienka zhrnutie rozdielov, ktoré nové IoT systémy prinášajú oproti klasickým, čisto softvérovým systémom. Tieto postrehy vychádzajú z návrhu, implementácie súčastí a prevádzkovania IoT systému v praxi. V práci je využitý IoT systém inteligentnej budovy.

Kľúčové slová: MPC, REST, IoT, CaaS

Obsah

Zoznam obrázkov

Zoznam skratiek a značiek

CRM - Customer Relationship Management - aplikácia na správu vzťahov so zákazníkom
ELK - Elasticsearch, Logstash, Kibana - sada nástrojov na prácu s Big Data
ERP - Enterprise Resource Planning - plánovanie zdrojov v podniku
ESB - Enterprise Service Bus - servisná zbernica podniku
FPGA - Field-programmable gate array - programovateľné hradlové pole
GPIO - General purpose input/output
HTML - HyperText Markup Language - značkovací jazyk používaný na vytváranie www stránok
HTTP - HyperText Transfer Protocol - protokol na prenos štrukturovaných informácií.
IEEE - Institute of Electrical and Electronics Engineers - Inštitút elektrotechnických a elektronických inžinierov
IoT - Internet of Things - internet vecí
IT - Information Technology
JSON - JavaScript Object Notation - Notácia na definovanie Javascript objektov
KPIs - Key Performance Indicators - kľúčove ukazovatele výkonnosti
MIMO - multiple input multiple output - viacrozmerný systéme
MPC - Model predictive controller - prediktivný regulátor
MVC - Model-View-Controller štruktúra aplikácie
NoSQL - non SQL - nie relačná databáza
OT - Operationals Technology
PAN - Personal Area Network
PID - regulátor s Proporčnou, Integračnou a Derivačnou zložkou
RAM - Random access memory - operačná pamäť
REST - Representational state transfer architectural style
RFID - Radio Frequency IDentification - identifikácia pomocou rádiovej frekvencie
RPC - Remote Procedure Call - volanie vzdialenej procedúry
RSS - Rich Site Summary
SISO - single input single output - jednorozmerný systém
SOA - service oriented architecture - architektúra so službou ako základným prvkom
SOAP - Simple Object Access Protocol - protokol výmenu štrukturovanej informácie
TCP/IP - Transmission Contro Protocol/Internet protocol
TICK - Telegraf, InfluxDB, Chronograf, Kapacitor sada nástrojov na prácu s Big Data
UML - Unified Modeling Language - unifikovnaý modelovací jazyk

UPnP - Universal Plug aNd Play
VPN - Virtual Private Network
Wifi - bezdrôtové protokoly rodiny 802.11
WSDL - Web Service Description Language - jazyak na opísanie webovej služby
WWW - World Wide Web - Celosvetová sieť (stránok s HTML obsahom)
XML - eXtensible Markup Language - rozšíriteľný značkovací jazyk

Zoznam algoritmov

Úvod

V posledných rokoch môžeme sledovať veľký rozmach pripájania do siete internet, či už mobilných zariadení, počítačov nehovoriac a sú prípady pripájania už aj chladničiek. Rovnako môžeme sledovať zvýšenie dostupnosti elektronických súčiastok a lacných zariadení typu Raspberry Pi. Okrem toho vidíme kontinuálny rozvoj v oblasti návrhu a vývoja softvéru nové jazyky, nové štandardy, nové spôsoby integrácie. Všeobecne môžeme povedať, že informačné technológie idú dopredu rýchlym tempom. Výsledkom rozvoja informačných technológii a spomínaných oblastí je vznik nových typov informačných systémov s názvom internet vecí - IoT (internet of things) systémy. Aktuálne je pojem IoT čoraz častejšie skloňovaný na konferenciách akademickej a rovnako aj komerčnej sféry. Rôzne popredné spoločnosti ako IDC, Gartner ai. zaoberajúce sa výskumnými a poradnými činnosťami v oblasti informačných a komunikačných technológií robia odhady využitia. Tvrdenie portálu www.crn.com: „Júnový prieskum trhu spoločnosti IDC predikoval, že výdavky na IoT dosiahnú v roku 2020 sumu vo výške 1,7 biliónov dolárov, zatiaľ čo Gartner predpovedal, že v tom istom roku bude pripojených 21 miliárd zariadení.“[1] V oblasti regulátorov určite v percentuálnom nasadení stále prevládajú konvenčné metódy typu PID, avšak dostávajú sa do praxe aj moderné metódy ako prediktívne alebo adaptívne riadenie. Zvýšiť mieru nasadenia moderných metód riadenia je možné práve overením ich funkčnosti a potvrdením ich lepších vlastností v praxi a následnou propagáciou týchto výsledkov.

Cieľom práce je spojenie novovznikajúcich IoT systémov s aplikovaním prediktívnej metódy riadenia (MPC - model predictive controller) do nich. Ak chceme hlavný cieľ prace špecifikovať do detailu, tak ide o overenie zavedenia myšlienky regulátor ako služba (CaaS - controller as a service) do praxe. V práci sa overuje konkrétne online forma prediktívneho riadenia s obmedzeniami a sledovaním referenčnej hodnoty, čo je vysvetlené v prvej časti práce. Myšlienka regulátor ako služba predstavuje koncept, v ktorom je celá zložitosť výpočtu akčného zásahu regulátora, ktorú MPC prináša, na serveri a akčný člen len vykonáva poskytnutý akčný zásah riadenej veličiny. V práci sú popísané detaily tejto myšlienky, implementácie, výhody a nevýhody tohoto prístupu. Regulátor sa v práci implementuje do reálneho IoT systému inteligentnej domácnosti. Prechod od opisu prediktívneho regulátora ku jeho overeniu tvorí časť popisujúca trendy softvérových architektúr. V tejto časti sú vysvetlené architektúry a architektonické princípy, ktoré sú význačné pre IoT systémy. V tejto časti je aj detailne vysvetlený pojem IoT. Keďže myšlienka CaaS by bez IoT systému nebola realizovateľná, rovnaké architektonické princípy sú použité pri

implementácii riešenia CaaS. Po opise implementácie prediktívneho regulátora do IoT systému je najdôležitejšia časť v práci a to overenie na konkrétnych fyzických zariadeniach. Overenie je vykonané na systéme udržiavania požadovanej hladiny intenzity osvetlenia. Na konci práce je posledná časť venujúca sa rozdielom medzi klasickým softvérovým systémom a IoT systémom z viacerých uhlov pohľadu, vychádzajúc z praktických skúsenosti návrhu, vývoja a prevádzkovania IoT systému.

1 Prediktívne metódy riadenia

Na začiatok, si povieme, čo je prediktívne riadenie a čo je základ prediktívneho riadenia.

Prediktívne riadenie (MPC, model predictive controller) je pokročilá metóda riadenia založená na optimalizácii, ktorá bola využívaná na aplikáciu v systémoch s pomalou dynamikou, napríklad v chemických, či petrochemických procesoch. Na rozdiel od lineárno-kvadratického regulátora, MPC na optimálne riadenie ponúka explicitné ošetrenie procesných obmedzení, ktoré vznikajú z prirodzených požiadaviek, napríklad efektivita nákladov, bezpečnostné obmedzenia akčných členov a iné.[42]

Hlavná výhoda prediktívneho regulátora je, že je riešený ako optimalizačný problém, takže sa snaží minimalizovať okrem iného hlavne potrebný riadiaci zásah na dosiahnutie žiadaného výstupu riadeného systému. Ďalšia výhoda pri riešení optimalizačného problému je, že sa jednoducho môžu zadať ohraničenia systému. Rôzne obmedzenia môžu byť aplikované na riadený systém a napriek tomu môže byť systém riaditeľný s minimálnym riadiacim zásahom. Táto metóda riadenia môže byť prirovnaná k prepočítavaniu ťahov v šachu. Pri prepočítavaní sa ráta s tým, čo sa môže udiať v čase v závislosti od poznania procesu, pri šachovej hre podľa jej pravidiel a podľa toho optimalizovať riadiace zásahy, ťahy v šachu, aby sa dosiahol najlepší výsledok z dlhodobého hľadiska, v prípade šachu, to je vyhratá partia. Pri MPC regulátoroch sa dá predísť javom, ktoré sa vyskytujú pri konvenčných regulátoroch ako PID, ako sú riadiace zásahy, ktoré dosahujú dobré výsledky z krátkodobého hľadiska, ale v konečnom dôsledku sa prejavia ako vysoko nákladné. Tento jav môže byť pomenovaný ako „vyhrať bitku, ale prehrať vojnu“.

Ďalšie výhody MPC regulátora sú také, že jednoducho vedia riadiť viac premenné systémy a ako už bolo spomenuté, zahrnúť obmedzenia systému – výstupu, riadiaceho zásahu, stavu systému, už pri návrhu regulátora. MPC regulátor obsahuje viacero premenných, pomocou, ktorých je možné ho vyladiť takmer pre každý proces.

Medzi nevýhody MPC regulátora patrí napríklad to, že niektoré MPC modely sú limitované len na stabilný proces v otvorenej slučke. Často vyžadujú veľký počet koeficientov modelu na opis odozvy systému. Niektoré MPC modely zase sú formulované na rušenie na výstupe a tie by ťažko mohli zvládnuť poruchy na vstupe. Niektoré MPC modely sú zase upravované na výstupe, pretože model nie je totožný s reálnym systémom. Tieto modely sú zvyčajne upravené o konštantu, podľa nameraných údajov, nerátajú však s tým, že táto zmena na výstupe sa môže v budúcnosti zmeniť, čo môže mať za následok, že finálny výsledok nebude optimálny. Taktiež ak horizont predikcie nie je zvolený správne, tak riadenie

nebude optimálne aj keď model systému správny bude. Niektoré systémy majú širokú škálu prevádzkových podmienok, ktoré sa často menia. Medzi príklady patria exotermické reaktory, procesy na dávkové spracovanie a tiež systémy, kde rôzny spotrebitelia majú rôzne špecifikácie produktov. Lineárne modely MPC regulátorov nie sú schopné zvládnuť dynamické správanie týchto procesov, preto musí byť použitý nelineárny model pre lepšie riadenie.[37]

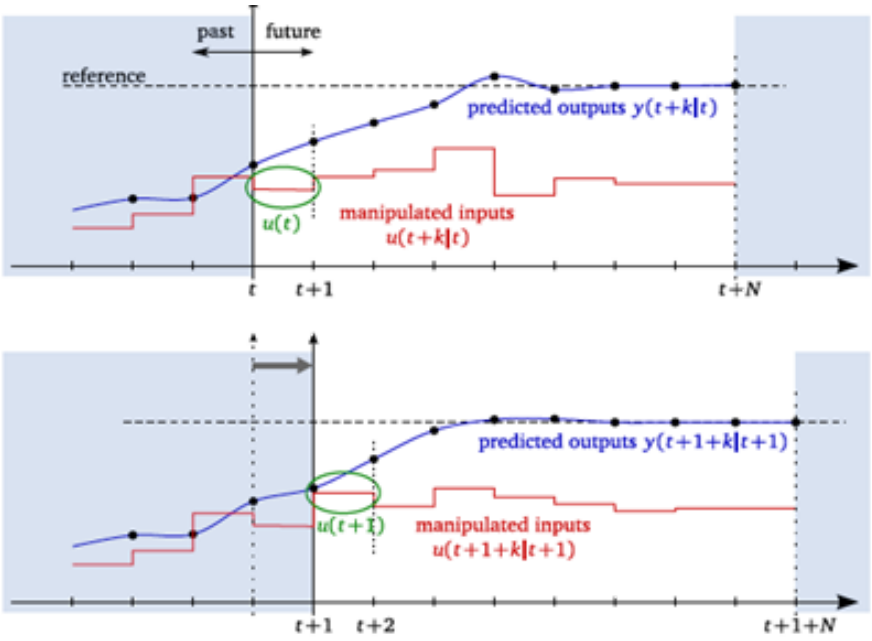
1.1 Teoretický základ MPC

V tejto časti sa vysvetlí história, matematický základ a variácie MPC regulátora.

1.1.1 On-line MPC

Prediktívny regulátor je, na najnižšej úrovni, metóda riadenia dynamických systémov, ktorá využíva nástroje matematickej optimalizácie. Spoločné črty všetkých prístupov riešenia problému prediktívnych regulátorov je vypočítať on-line, v každej časovej vzorke, optimálny riadiaci zásah v konečnom horizonte predikcie pre dynamický model systému, kde aktuálny stav je počiatočný stav. Iba prvý element z vypočítanej sekvencie predikovaných riadiacich zásahov je potom aplikovaný na systém. V ďalšom okamihu vzorkovania je horizont predikcie posunutý a výpočet optimálneho riadiaceho zásahu je vykonávaný znovu pre novonadobudnutý stav. Táto myšlienka nie je nová, už v článku od Lee a Markus (1967), je možné nájsť nasledujúce tvrdenie: „Jedna technika na návrh regulátora so spätnou väzbou zo znalosti regulátora pre otvorenú slučku je merať aktuálny stav riadenia procesu a veľmi rýchlo vypočítať funkciu riadenia pre otvorenú slučku. Prvá časť tejto funkcie je potom využitá počas krátkeho intervalu, po ktorom je opäť zmeraný stav procesu a k nemu vypočítaná riadiaca funkcia pre otvorenú slučku. Táto procedúra je potom opakovaná.“ Technika popisovaná v článku od Lee a Markus (1967) je zvyčajne označovaná ako „Receding Horizon Control“ (RHC) – „Postupujúci horizont riadenia“ a dnes je viac-menej používaný ako synonymum k pojmu „Model Predictive Control“ – „Prediktívne riadenie“. [35] Popisovaný princíp je znázornený na obrázku 1. Nech existuje lineárny dynamický model s časovo invariantnými parametrami, vyjadrený diskrétnou stavovou rovnicou:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) + Du(t), \\ x(t) \in R^n, y(t) &\in R^P, u(t) \in R^m, \\ A \in R^{(n\times n)}, B &\in R^{(n\times m)}, C \in R^{(p\times n)} \end{aligned} \tag{1}$$



Obrázok 1: Postupujúci horizont riadenia

x, y, u sú stavy systému, výstupy systému a riadiace zásahy v uvedenom poradí v čase alebo lepšie povedané vo vzorke t.

n – predstavuje počet stavov systému

p – predstavuje počet výstupov

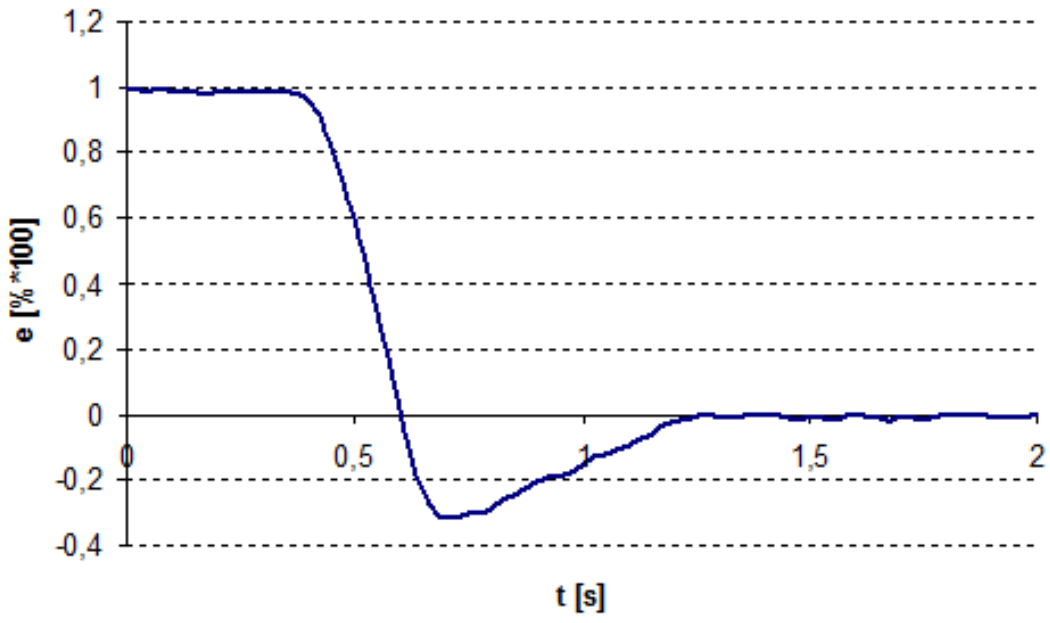
m – predstavuje počet vstupov

Matice A, B, C sú systémová matica, vstupná matica a výstupná matica v uvedenom poradí. System 1 musí spĺňať nasledujúce obmedzenia na stav a vstup:

$$\begin{aligned} x(t) \in X, u(t) \in U \\ U \subset R^m \\ X \subset R^n \end{aligned} \tag{2}$$

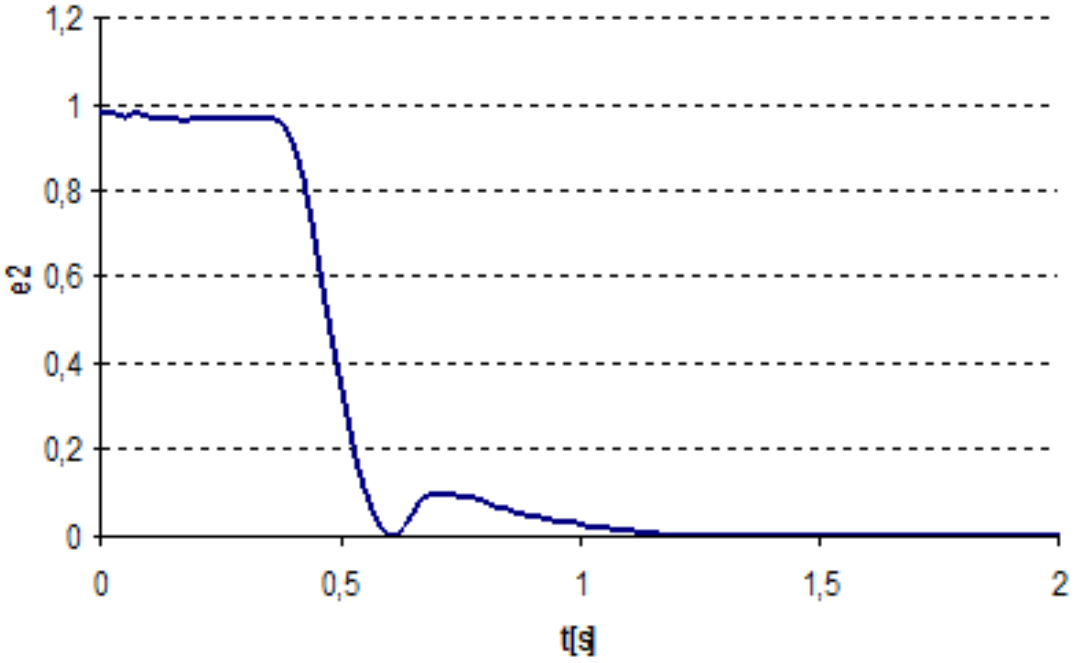
kde obmedzujúca množina riadiacich zásahov U je konvexná, kompaktná (uzavretá a ohraničená) a obmedzujúca množina stavov X je konvexná a uzavretá. Pre obidve množiny U a X sa predpokladá, že obsahujú počiatok v ich vnútri.[36] Najskôr je jednoduchšie vyriešiť optimálne riadenie bez obmedzení. Pozornosť bude upriamená na nájdenie takej optimálnej postupnosti u^*(k), ..., u^*(k+N – 1), ktorá bude minimalizovať zvolené kvadratické kritérium a zároveň rešpektovať dynamiku systému. Čo inými slovami znamená, že tento regulátor sa bude snažiť minimalizovať stav a rovnako riadiaci zásah, ešte lepšie povedané ich kvadrát. Ak by teda nebola špecifikovaná referenčná hodnota, ktorú má výstup regulátora sledovať, tak implicitne výstupná veličina prediktívneho regulátora

konverguje k hodnote 0 alebo pri systémoch s viacerými výstupmi k nulovému vektoru. N – predstavuje horizont predikcie. Ak je teda systém v stave x_0, ktorý poznáme a horizont predikcie je 3, tak do optimalizačnej funkcie vstupujú premenné x_1, x_2, x_3 a u_1, u_2, u_3 kde u_1 zabezpečí prechod do stavu x_1, u_2 do stavu x_2 atď. každá premenná x je odvoditeľná z predošlého stavu a prvý stav x_0 je známy. Preto jedinou „neznámou“ ostáva sekvencia riadiacich zásahov. Jedna z otázok by mohla byť, prečo sa využíva kvadratické kritérium. Ako odpoveď je možné použiť príklad kvality riadenia. Pri hodnotení kvality riadenia sa používajú integrálne kritéria. Existuje jednoduché kritérium, ktoré spraví integrál pod krivkou regulačnej odchýlky. Nevýhodou tohto je, že ak dochádza k preregulovaniu a regulačná odchýlka je záporná, veľkosť plochy je zmenšovaná o tie časti, ktoré sú záporné. Toto sa rieši absolútnym integrálnym kritériom, ktoré spraví zo zápornej regulačnej odchýlky kladnú a teda plocha pod krivkou sa zväčšuje aj pri preregulovaní. Navyše existuje kvadratické kritérium, ktoré okrem toho, že odstraňuje problém so zápornou regulačnou odchýlkou navyše viac penalizuje hodnoty odchýlky väčšie ako 1. Inými slovami, ak je hodnota viac ako 1 o to horšiu kvalitu regulácie bude toto kritérium indikovať. Problém so zápornou regulačnou odchýlkou je znázornený na obrázkoch 2, 3 a 4.

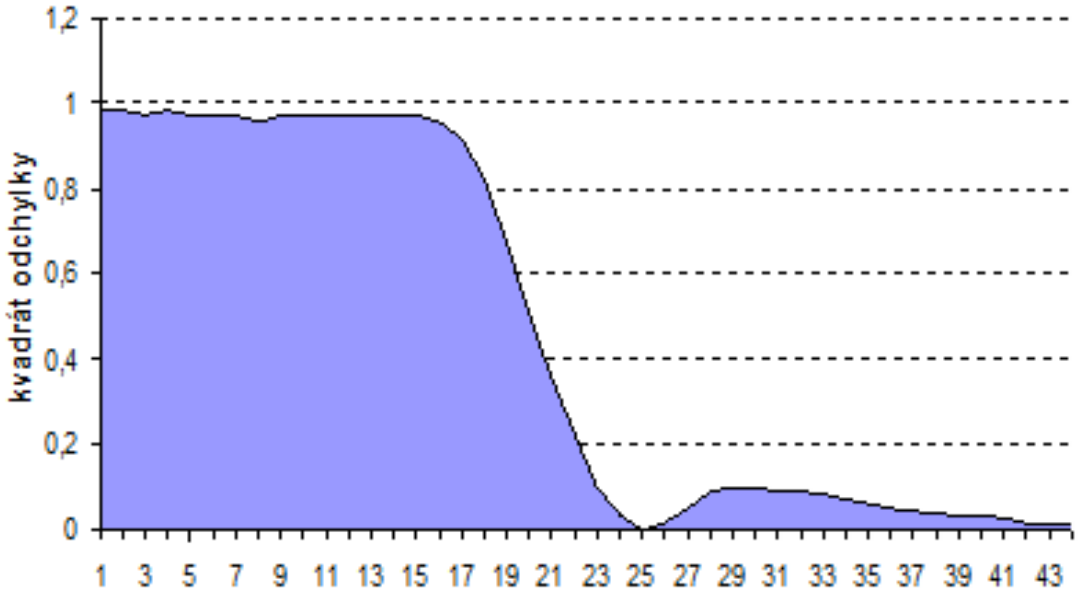


Obrázok 2: Hodnota regulačnej odchýlky v čase

Rovnako ako pri minimalizácii regulačnej odchýlky, tak aj pri minimalizácii riadiaceho zásahu je kvadratické kritérium najlepším ukazovateľom.



Obrázok 3: Časová závislosť kvadrátu regulačnej odchýlky



Obrázok 4: Plocha kvadrátu regulačnej odchýlky

Pre porozumenie ďalších vzťahov si je potrebné uvedomiť, že: $x(t+k) = x_{(t+k)}$

- Pre vzorku k = 0 (aktuálny stav systému): x(t)= x_t,
- pre vzorku k = 1 x(t+1)=$x_{(t+1)}$,
- atď.

Kvadratické kritérium pre konečný horizont predikcie dĺžky N je:

$$J(x(t), u(t), \ldots, u(N-1)) = \frac{1}{2}\sum_{k=0}^{N-1}[x_k^T Q x_k + u_k^T R u_k] + \frac{1}{2}x_N^T Q_N x_N \tag{3}$$

kde:

$$\begin{aligned} x_(k+1) &= Ax_k + Bu_k \\ x_0 &= x(t), \\ Q = Q^T \succcurlyeq 0, Q_N = Q_N^T \succcurlyeq 0, R = R^T \succ 0 & \\ Q_N &\in R^{n\times n} \\ R &\in R^{m\times m} \end{aligned} \tag{4}$$

Matice, Q, Q_N a R sú nazývané váhové matice a spolu s horizontom predikcie N sú nazývané parametrami na ladenie prediktívneho regulátora. Nájdenie optimálneho riadenia, ktoré by minimalizovalo kvadratické kritérium predstavuje dynamickú optimalizáciu a má v tomto prípade aj analytické riešenie: [38]

$$\begin{aligned} x_{t+k} &= A^k x_0 + \sum_{i=0}^{k-1} A^i B u_{k-1-i} \\ u_{t,N} &= [u_t^T, \ldots, u_{t+N-1}^T] \end{aligned} \tag{5}$$

Vyjadrenie predikovaného stavu je potom:

$$[x_{t+1}^T, \ldots, x_{t+N}^T]^T = Vx_0 + Tu_{t,N} \tag{6}$$

Táto rovnica je jedna z najdôležitejších pri pochopení fungovania on-line prediktívneho algoritmu. Preto je vhodné ukázať ako matica V a T vyzerajú pre konkrétny

jednoduchý systém s horizontom predikcie N = 3 zadaný v stavovom priestore:

$$\begin{aligned} A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0,5 \end{bmatrix} \\ C = \begin{bmatrix} 0 & 1 \end{bmatrix}, D = 0 \\ x_0 = \begin{bmatrix} 4 \\ 5 \end{bmatrix} = \begin{bmatrix} x_{01} \\ x_{02} \end{bmatrix} \end{aligned} \tag{7}$$

Matice V a T budú vyzerať:

$$\begin{aligned} V = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \\ 2 & 1 \\ 1 & 0 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} A^1 \\ A^2 \\ A^3 \end{bmatrix} \\ T = \begin{bmatrix} 1 & 0 & 0 \\ 0,5 & 0 & 0 \\ 1 & 1 & 0 \\ 1,5 & 0,5 & 0 \\ 1 & 1 & 1 \\ 2,5 & 1,5 & 0,5 \end{bmatrix} = \begin{bmatrix} A^0B & 0 & 0 \\ A^1B & A^0B & 0 \\ A^2B & A^1B & A^0B \end{bmatrix} \end{aligned} \tag{8}$$

Výsledný vzťah:

$$\left[\begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix}, \begin{bmatrix} x_{21} \\ x_{22} \end{bmatrix}, \begin{bmatrix} x_{31} \\ x_{32} \end{bmatrix} \right]^T = \begin{bmatrix} A^1 \\ A^2 \\ A^3 \end{bmatrix} \begin{bmatrix} x_{01} \\ x_{02} \end{bmatrix} + \begin{bmatrix} A^0B & 0 & 0 \\ A^1B & A^0B & 0 \\ A^2B & A^1B & A^0B \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{9}$$

Dôležitý krok pri hľadaní optimálneho riadenia je zo vzťahu 3 „odstrániť“ sumu. Finálny vzťah vyzerá:

$$J\left(x\left(t\right), U_t\right) = \frac{1}{2} {u_{t,N}}^T H u_{t,N} + x^T\left(t\right) F U_t + \frac{1}{2} x^T\left(t\right) \mathrm{Yx}\left(t\right), \tag{10}$$

Matica $H \in R^{(m \bullet N \times m \bullet N)}$, $F \in R^{(n \times m \bullet N)}$, $Y \in R^{(n \times n)}$. Ak sa symbol $\otimes$ označí ako kroneckerovo násobenie matíc a $I_j \in R^{(j \times j)}$, jednotková matica s príslušnou dimenziou,

tak platí:

$$H = T^T \tilde{Q} T + I_N \bigotimes R, \quad F = V^T \tilde{Q} T, \quad Y = V^T \tilde{Q} V$$
$$\tilde{Q} = \begin{bmatrix} I_{N-1} \bigotimes Q & 0 \\ 0 & Q_N \end{bmatrix} \tag{11}$$

Pre systém zo vzťahu 7 a váhy $Q = Q_N = I_2,\ R = 0,1$ budú jednotlivé matice vyzerať:

$$H = \begin{bmatrix} 11,85 & 6,5 & 2,25 \\ 6,5 & 4,6 & 1,75 \\ 2,25 & 1,75 & 1,35 \end{bmatrix}, F = \begin{bmatrix} 14 & 7,5 & 2,5 \\ 4,5 & 2 & 0,5 \end{bmatrix}, \quad Y = \begin{bmatrix} 17 & 6 \\ 6 & 3 \end{bmatrix} \tag{12}$$

Optimálnu riadiacu sekvenciu je možné dostať minimalizáciou kvadratickej formy 10:

$$u^*_{t,N}\left(x\left(t\right)\right) = \arg\min_{u_t,N} \left\{ J(x\left(t\right), u_{t,N}\ \right\} = -H^{-1}F^T x(t) \tag{13}$$

Optimálna hodnota kritéria je:

$$J^*\left(x\left(t\right)\right) = \min_{u_t,N} \left\{ J(x\left(t\right), u_{t,N}\ \right\} = \frac{1}{2} x^T(t)(Y - FH^{-1}F^T)x(t) \tag{14}$$

Na to aby sa zabezpečila spätná väzba, je potrebné v každom kroku použiť iba prvú hodnotu zo sekvencie riadiacich zásahov a potom zmerať stav a na základe tej hodnoty znovu vypočítať optimálnu sekvenciu riadiacich zásahov. Bez merania stavu, by to bola regulácia v otvorenom regulačnom obvode. Meranie stavu a jeho použitie pri výpočte nasledujúceho riadiaceho zásahu v každom kroku zabezpečí reguláciu v uzavretom regulačnom obvode.

1.1.2 On-line MPC s obmedzením

Doteraz sa reguloval systém, v ktorom nebolo žiadne obmedzenie. V realite ich však býva mnoho. Pokiaľ sa pri návrhu regulátora začnú brať do úvahy rovnice 2, bude ich treba zapracovať do výpočtu. Pri návrhu prediktívneho regulátora s obmedzením sa využíva kvadratické programovanie. Úloha kvadratického programovania je úlohou nelineárneho

programovania, v ktorej sústava ohraničení je lineárna a účelová funkcia je kvadratická. Všeobecná formulácia kvadratickej úlohy je:

$$\begin{aligned} f(x) &= \min_x \left\{ x^T H x + F^T \mathrm{x} \right\} \\ x \in D &= \{ \mathrm{x} | \quad Lx \leq m_c,\ x \geq 0 \} \end{aligned} \tag{15}$$

Pre návrh regulátora je premenná, ktorej minimum sa hľadá, sekvencia riadiacich zásahov $u^*_{t,N}$. Rovnice pre prediktívny regulátor s obmedzením následne vyzerajú:

$$J(x(t),\ u(t), \ldots,\ u(N-1)) = \frac{1}{2} \sum_{k=0}^{N-1} \left[x_k^T Q x_k + u_k^T R u_k \right] +\ \frac{1}{2} x_N^T Q_N x_N \tag{16}$$

kde:

$$\begin{aligned} x_{k+1} &= A x_k + B u_k, \\ x_0 &= x(t), \\ E_c x_k + G_c u_k &\leq m_c, \quad k = 1 \ldots N \\ Q = Q^T \succcurlyeq 0,\ Q_N &= Q_N^T \succcurlyeq 0,\ R = R^T \succ 0 \end{aligned} \tag{17}$$

Aby bolo možné sústavu rovníc 16 a 17 zapracovať do kvadratického programovania, previesť do maticového tvaru. Prvá časť rovnice ostáva nezmenená, pridá sa k nej druhá časť:

$$\begin{aligned} J(x(t), U_t) = \ & \frac{1}{2} {u_{t,N}}^T H u_{t,N} + x^T(t) F U_t + \frac{1}{2} x^T(t)\, \mathrm{Yx}(t), \\ & G u_{t,N} \leq w + E x(t) \end{aligned} \tag{18}$$

Kde matice H, F, Y sú určené rovnako ako vo vzťahu 11 a matice G, E, a vektor w majú tvar:

$$G = \begin{bmatrix} -I_N \\ I_N \\ -(I_N \otimes CT) \\ (I_N \otimes CT) \\ \vdots \end{bmatrix},\ E = \begin{bmatrix} 0 \\ 0 \\ -(I_N \otimes CV) \\ (I_N \otimes CV) \\ \vdots \end{bmatrix},\ w = \begin{bmatrix} -(1_N \otimes u_{\min}) \\ (1_N \otimes u_{\max}) \\ -(1_N \otimes y_{\min}) \\ (1_N \otimes y_{\max}) \\ \vdots \end{bmatrix} \tag{19}$$

V rovnici 19 sú znázornené základné systémové obmedzenia. Môže existovať ďaleko viac. 1_N predstavuje jednotkový vektor o veľkosti N. Prvé dva riadky matice G, $G_{1,2} \in$

$R^{N\times N}$, E, $E_{1,2} \in R^{N\times n}$ a vektor w, $w_{1,2} \in R^N$ v (19) predstavujú obmedzenia vstupu. Druhé dva riadky matice G, $G_{3,4} \in R^{N\times N}$, E, $E_{3,4} \in R^{N\times n}$ a vektor w, $w_{3,4} \in R^N$ v 19 predstavujú obmedzenia výstupu. Ďalšie obmedzenia, by museli nadobúdať rovnaké rozmery ako predošlé riadky príslušných matíc a vektora.

Ak sa pridajú do systému 7 obmedzenia na vstup a výstup:

$$-1 \leq u\,(t) \leq 1, \quad -10 \leq y\,(t) \leq \ 10 \tag{20}$$

matice G, E a vektor w budú vyzerať:

$$G = \begin{bmatrix} -I_3 \\ I_3 \\ -(I_3 \bigotimes CT) \\ (I_3 \bigotimes CT) \end{bmatrix}, \ E = \begin{bmatrix} 0 \\ 0 \\ -(I_3 \bigotimes CV) \\ (I_3 \bigotimes CV) \\ \vdots \end{bmatrix}, \ m_c = \begin{bmatrix} -(1_3 \bigotimes u_{\min}) \\ (1_3 \bigotimes u_{\max}) \\ -(1_3 \bigotimes y_{\min}) \\ (1_3 \bigotimes y_{\max}) \end{bmatrix} \tag{21}$$

$$G = \begin{bmatrix} -\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ -\begin{pmatrix} 0,5 & 0 & 0 \\ 1,5 & 0,5 & 0 \\ 2,5 & 1,5 & 0,5 \end{pmatrix} \\ \begin{pmatrix} 0,5 & 0 & 0 \\ 1,5 & 0,5 & 0 \\ 2,5 & 1,5 & 0,5 \end{pmatrix} \end{bmatrix}, \ E = \begin{bmatrix} -\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \\ -\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix} \end{bmatrix}, \ w = \begin{bmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 10 \\ 10 \end{pmatrix} \\ \begin{pmatrix} 10 \\ 10 \end{pmatrix} \end{bmatrix} \tag{22}$$

Vznikne sústava lineárnych rovníc, ktoré tvoria obmedzenia pre kvadratický problém.

$$\begin{bmatrix} -\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ -\begin{pmatrix} 0,5 & 0 & 0 \\ 1,5 & 0,5 & 0 \\ 2,5 & 1,5 & 0,5 \end{pmatrix} \\ \begin{pmatrix} 0,5 & 0 & 0 \\ 1,5 & 0,5 & 0 \\ 2,5 & 1,5 & 0,5 \end{pmatrix} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \leq \begin{bmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 10 \\ 10 \end{pmatrix} \\ \begin{pmatrix} 10 \\ 10 \end{pmatrix} \end{bmatrix} + \begin{bmatrix} -\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ -\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix} \end{bmatrix} \begin{bmatrix} x_{01} \\ x_{02} \end{bmatrix} \tag{23}$$

Ak sú obmedzenia na systém príliš striktné, môže sa stať, že kvadratický problém nemá riešenie. Takáto situácia je znázornená na obrázku 5.

Ak by obmedzenia na výstup boli napríklad $-13 \leq y\,(t) \leq\ 13$, kvadratický problém by bol riešiteľný a jeho riešenie pre horizont predikcie 3 je zobrazene na obrázku 6.

1.1.3 On-line MPC so sledovaním referenčného signálu

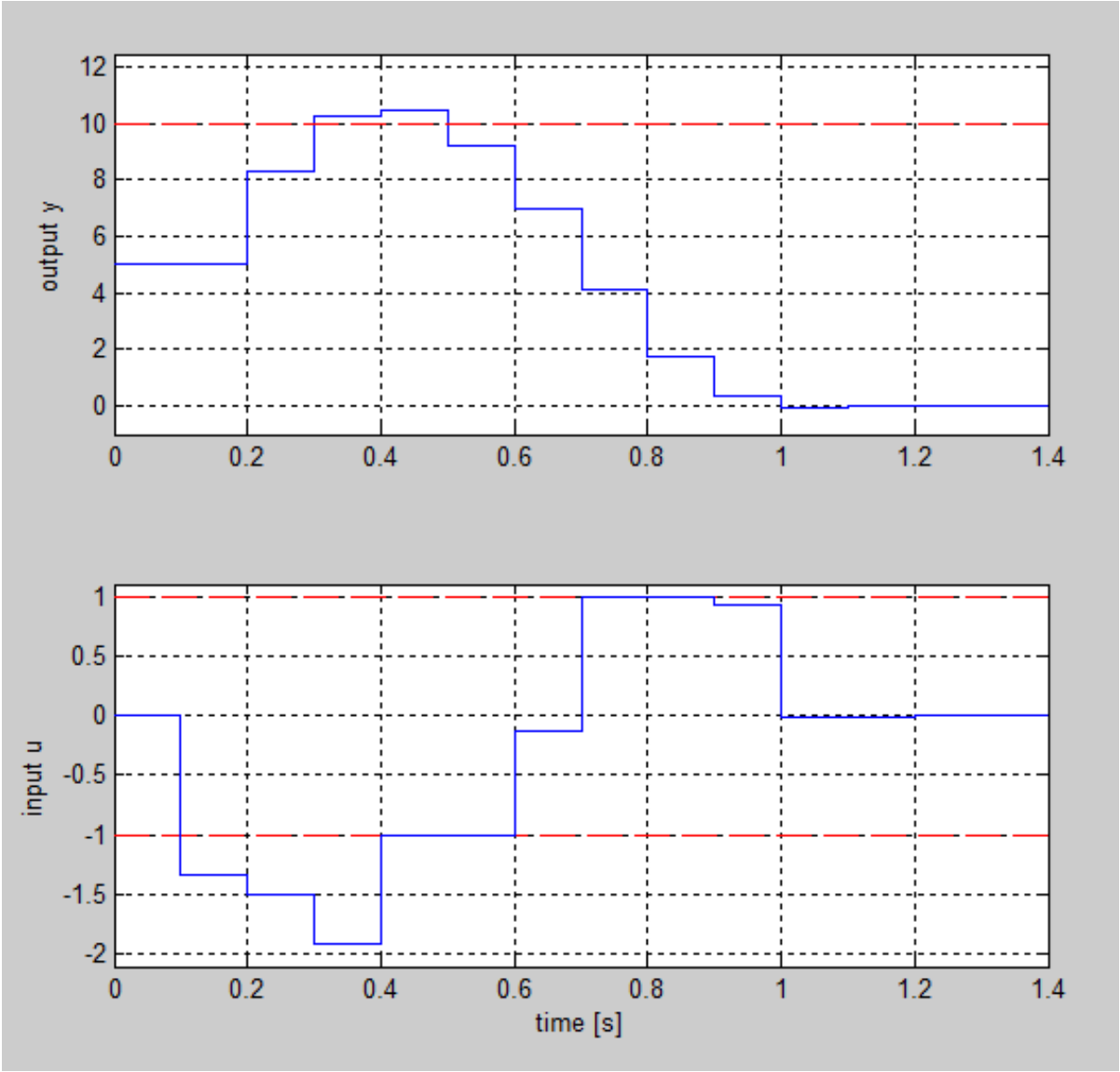
Doteraz bol riešený regulátor, ktorý reguloval hodnotu k „počiatku“ k hodnote 0 alebo nulovému vektoru. Táto kapitola obsahuje návrh MPC regulátora, ktorý bude sledovať po častiach konštantný referenčný signál r(t). Stále sa berie do úvahy systém 1. Najprv je potrebné nahradiť člen $x_k^T Q x_k$ v pôvodnom kritériu 17 odchýlkou

$$(z_k - r_k)^T Q_y (z_k - r_k) \tag{24}$$

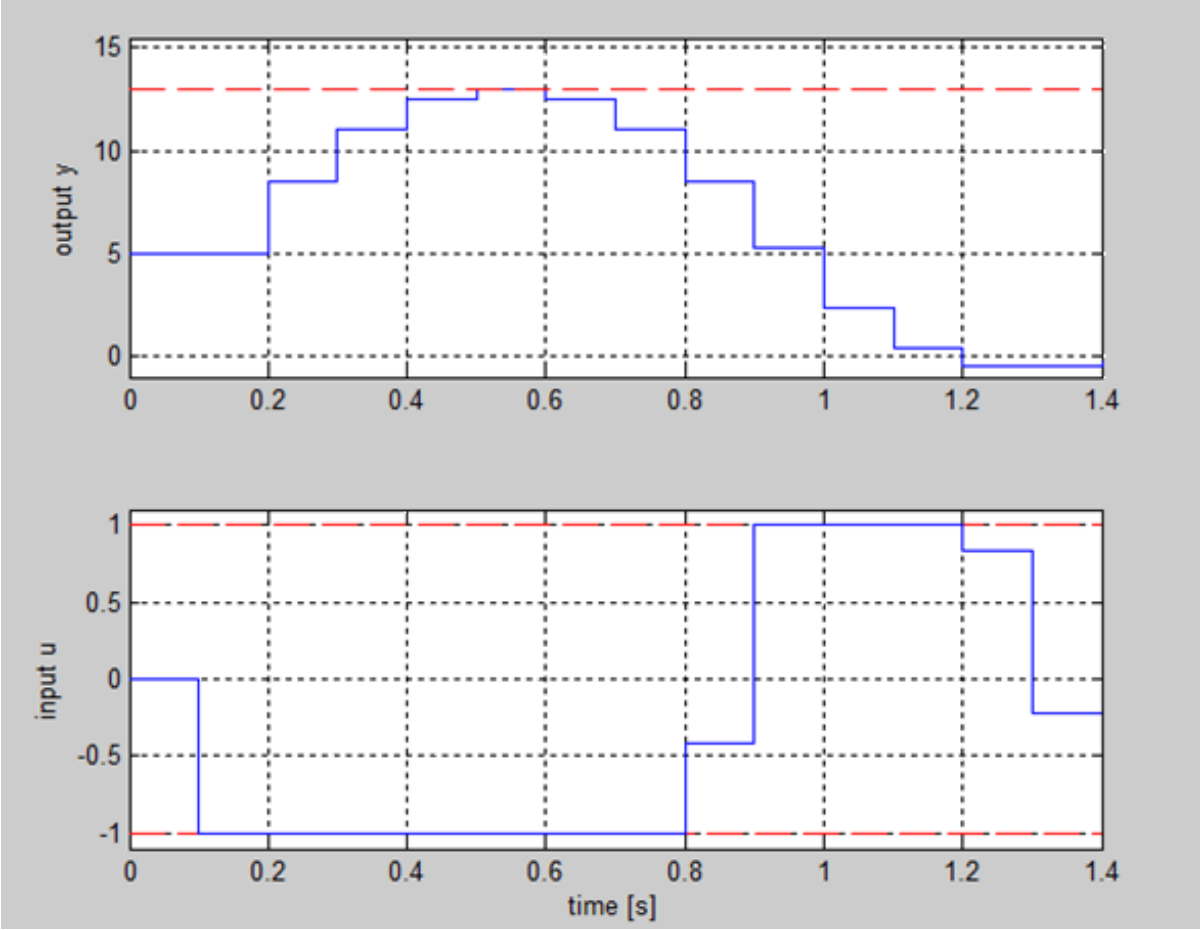
kde

$$z\,(t) = Z y\,(t)\,,\ z\,(t) \in R^{p_z},\ p_z \leq p,\ r \leq m \tag{25}$$

Aby to bolo možné, je potrebné poznať predikovanú hodnotu referenčného signálu. Ta môže byť definovaná rôznymi spôsobmi jedna z možností: r(t+1)=r(t).



Obrázok 5: Časový priebeh výstupnej veličiny a riadiaceho zásahu pre neriešiteľný kvadratický problém. Červená čiara predstavuje obmedzenie na danú veličinu



Obrázok 6: Časový priebeh výstupnej veličiny a riadiaceho zásahu riešiteľného kvadratického problému

V ustálenom stave je potrebné, aby platila rovnosť $z_u = r_u$. Takže:

$$\begin{bmatrix} I_n - A & -B \\ \text{ZC} & 0 \end{bmatrix} \begin{bmatrix} x_u \\ u_u \end{bmatrix} = \begin{bmatrix} 0 \\ r_u \end{bmatrix} \tag{26}$$

Ak ma predchádzajúca matica plnú riadkovú hodnosť, existuje riešenie u_u a x_u.. V dôsledku nenulového ustáleného vstupu sa v praxi často používa odchýlka $u_k = u_k - u_{k-1}$. Ak sústava obsahuje integrátor, je lepšie použiť hodnotu u. [38]

Kritérium pre sledovanie referencie má tvar:

$$J\left(x\left(t\right),\ u\left(t\right),\ldots,\ u\left(N-1\right)\right) = \frac{1}{2}\sum_{k=0}^{N-1}\left[e_k^T Q_e e_k + u_k^T R u_k\right] \tag{27}$$

kde

$$e_k = y_k - r_k \tag{28}$$

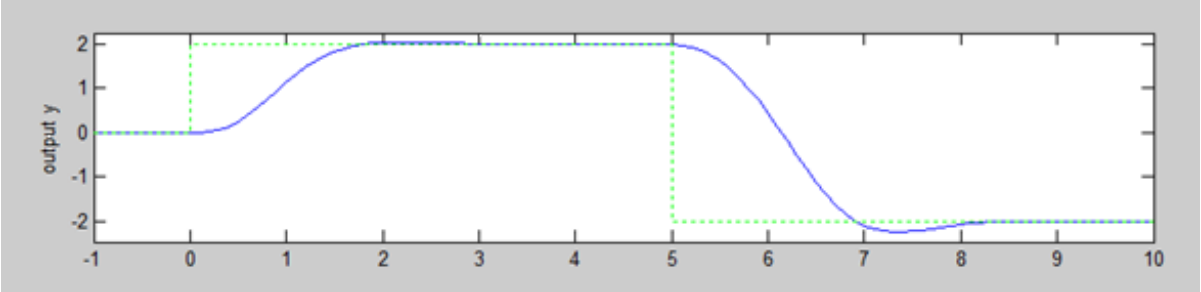
je regulačná odchýlka a

$$\Delta u_k = u_k - u_{k-1} \tag{29}$$

je optimalizovaná premenná. Systém rozšírený o históriu riadiaceho zásahu u_k a generátor referencie r_k je formulovaný:

$$\begin{bmatrix} x_{k+1} \\ u_k \\ r_{k+1} \end{bmatrix} = \begin{bmatrix} A & B & 0 \\ 0 & I_m & 0 \\ 0 & 0 & I_{n_y} \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \\ r_k \end{bmatrix} + \begin{bmatrix} B \\ I_m \\ 0 \end{bmatrix} u_k = \begin{array}{l} A_m \hat{x_k} + B_m u_k \\ \\ e_k = \begin{bmatrix} C & 0 & -I_{n_y} \end{bmatrix} \hat{x_k} = C_m \hat{x_k} \end{array} \tag{30}$$

Po upravení algoritmu na sledovanie po častiach spojitého referenčného signálu už MPC regulátor neriadi výstupnú veličinu k 0 hodnote alebo nulovému vektoru, ale ku aktuálnej hodnote prípadne vektoru referenčného signálu v kroku k, ..., k + N - 1, kde N predstavuje horizont predikcie. Preto referenčný signál vstupujúci do výpočtu je je buď vektor pre jednorozmerné systémy (SISO) alebo matica pre viacrozmerné systémy (MIMO). Časový priebeh výstupnej veličiny SISO systému regulovaného MPC regulátorom so sledovaním referencie je znázornený na obrázku 7.



Obrázok 7: Sledovanie referencie, časové priebehy - zelená referencia, modrá výstup systému

1.1.4 Explicitné riešenie - offline MPC

Je zrejmé, že optimálna hodnota kritéria 14 a optimálna riadiaca sekvencia 13 (aj s obmedzeniami) sú funkciou stavu x(t). Túto úlohu je možné formulovať pomocou **multiparametrického kvadratického programovania** (mp-QP), ktoré sa snaží nájsť optimálne riešenie pre všetky možné hodnoty parametra x(t) vopred. Ak sa spraví substitúcia rovnice

$$J^*\left(x\left(t\right)\right) = \min_{u_t, N} \left\{J\left(x\left(t\right), u_t\right) \middle| Gu_t \leq W + Ex(t)\right\} \tag{31}$$

pomocou

$$u_t = \tilde{u_t} - H^{-1}F^T x(t) \tag{32}$$

vznikne:

$$J^*\left(x\left(t\right)\right) = \min_{u_t, N} \left\{J\left(\tilde{u_t}, x\left(t\right)\right) = \frac{1}{2}\tilde{u_t}^T H \tilde{u_t} + \beta \middle| G\tilde{u_t} \leq W + Sx(t)\right\} \tag{33}$$

Kde $S = E + GH^{-1}F^T$ a $\beta = \frac{1}{2}x\left(t\right)^T \left(FH^{-1}F^T + Y\right) x(t)$. Ešte je potrebné zaviesť množinu indexov I= $\{1, \ldots, q\}$, ktorá zodpovedá riadkom matice G, W a S.
Kritický región CR je taká polytopická oblasť v priestore parametrov x(t), ktorá má v optimálnej hodnote $J^*\left(x\left(t\right)\right)$, $u^*\left(x\left(t\right)\right)$ aktívne rovnaké obmedzenia $A(x(t)) \subset I$. Takže platí

$$G_A \tilde{u_t}^*\left(x\left(t\right)\right) = W_A + S_A x\left(t\right) \;\; pre \; x(t) \in CR \tag{34}$$

Nech $H \succ 0$ a nech $G_A$majú lineárne nezávisle riadky, potom optimálna sekvencia $\tilde{u_t}^*(x(t))$ je jednoznačne definovaná afinnou funkciou stavu x(t) na danom kritickom regióne CR.

$$\tilde{u_t}^*(x(t)) = H^{-1}G_A^T(G_A H^{-1} G_A^T)(W_A + S_A x(t)) \tag{35}$$

Ak sa preformuluje optimalizačný problém 13 (s obmedzeniami) ako mp-QP a $H \succ 0$, potom optimálna riadiaca sekvencia $u_t{}^*(x(t)) : X_{\text{feas}} \to R^m$ je spojitá, po častiach afinná funkcia na polyedry a optimálna hodnota $J^*(x(t))$ je spojitá, konvexná a po častiach kvadratická funkcia na polyedry.

Algoritmus mp-QP najskôr spočíta riešenie (13) (s obmedzeniami) pre vhodne zvolenú počiatočnú podmienku ($x_0 \in X_{\text{feas}}$) a vytvorí sa príslušný kritický región CR_0. Potom rekurzívne prehľadá okolie a vytvára nové regióny. Výsledkom je rozdelenie oblasti X_{feas}do kritických regiónov $CR_i = \{x|\, P_i x \leq p_i\}$, nad ktorými je definovaná spojitá afinná funkcia:

$$u_t{}^*(x(t)) = F_i x(t) + G_i \tag{36}$$

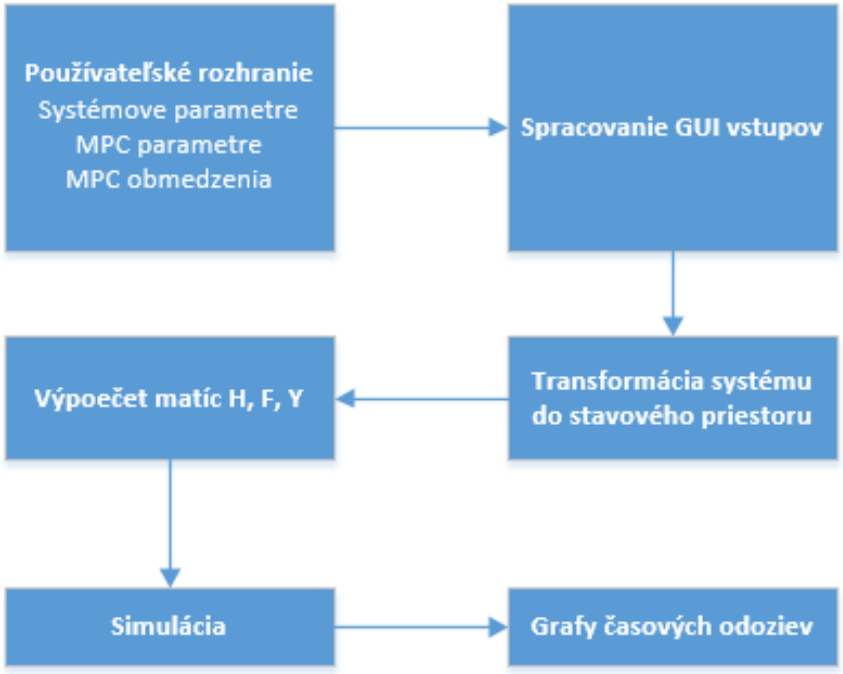
a spojitá kvadratická funkcia

$$J_t{}^*(x(t)) = x^T(t) A_i x(t) + B_i x(t) + C_i \tag{37}$$

Týmto sa presunula numerická výpočtová náročnosť optimalizácie 13 k off-line výpočtom. V priebehu riadenia stačí identifikovať región CR_i, obsahujúci aktuálny stav x(t) a aplikovať príslušný zákon riadenia . [38]
Takýmto spôsobom je možné rozdeliť výpočtovú zložitosť na dve časti. Prvá, výpočtovo zložitejšia, časť je hľadanie kritických regiónov, ktorá sa môže uskutočniť pred zavedením regulátora do behu. Tuto nie je čas kritický, pretože regulačný proces nebeží. Druhá časť je počas behu regulačného procesu. Tá predstavuje časovo nenáročnú operáciu vyhľadania kritického regiónu v pamäti a podľa neho vrátiť akčný zásah. Hlavná nevýhoda offline - explicitného riešenia je, že systém je jednorázovo daný a už počas behu nevstupuje do výpočtu na rozdiel od online riešenia, kde do každého kroku matematický model systému vstupuje a teda je možné matematický model dynamicky adaptovať, aby čo najviac zodpovedal reálnemu systému. Pri voľbe spôsobu implementácie praktickej časti pre výučbu aj neskôr v IoT prostredí, tento fakt zavážil a zvolila sa online metóda implementácie.

1.2 Popis funkčnosti On-line MPC algoritmu

Vytvorený algoritmus sa skladá skladá z viacerých modulov - funkcií naprogramovaných v prostredí Matlab a prispôsobených na fungovanie v open source verzii Octave. Program bol vytvorený pre účely využitia v pedagogickom procese. Grafické rozhranie bolo vytvorené pomocou komponent Matlabu - Guide, ktorý slúži práve na vytváranie grafických rozhraní. Základné komponenty, z ktorých sa program skladá sú znázornené na obrázku 8.



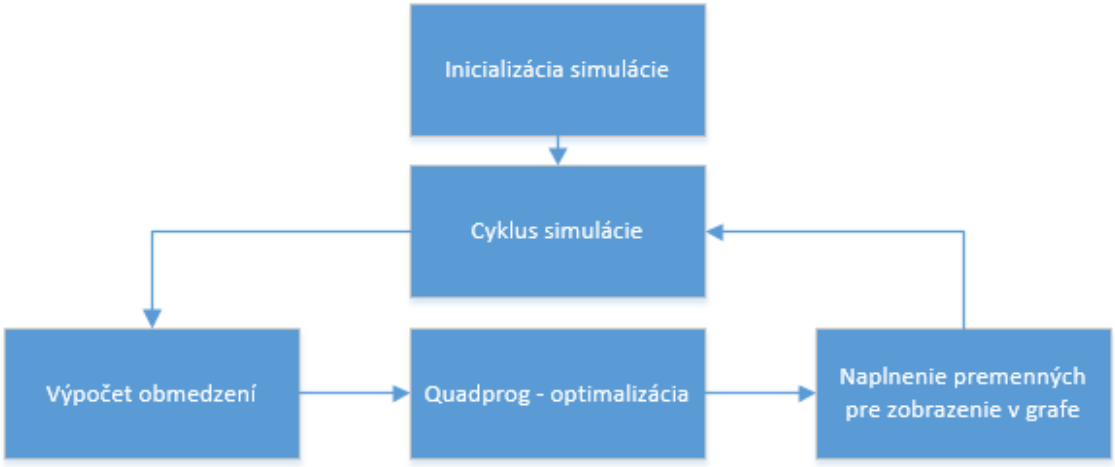
Obrázok 8: Základné komponenty

1. Používateľské rozhranie je popísane v kapitole 1.3
2. Spracovanie vstupov zabezpečí správnu konverziu reťazcov na čísla a výsledné čísla priradí do správnych premenných potrebných na vstupe do ďalšej časti.
3. MPC regulátor pre svoje fungovanie vždy potrebuje mať na vstupe matematický model systému podľa rovnice 1, ktorý sa označuje ako systém zadaný v stavovom priestore. Táto časť zabezpečí konverziu z prenosovej funkcie na stavový priestor.

4. Nasleduje výpočet matíc H, F, Y podľa rovníc 11, ktoré sú produktom matíc A, B, C z rovnice 1 a váhových matíc, Q, Q_n a R z rovnice 11, ktoré si používateľ volí.

5. Následne prebieha simulácia, ktorej komponenty sú znázornené na obrázku 9 a popísane nižšie.

6. Produktom simulácie sú dáta, ktoré vytvoria grafy časových odoziev popísaných v kapitole 1.3.



Obrázok 9: Detail komponentu simulácia

1. Počas inicializácie simulácie sa pripravuje referenčný signál, počiatočný stav systému a počet krokov simulácie.

2. Počet krokov simulácie je nastavený na podiel času simulácie a periódy vzorkovania.

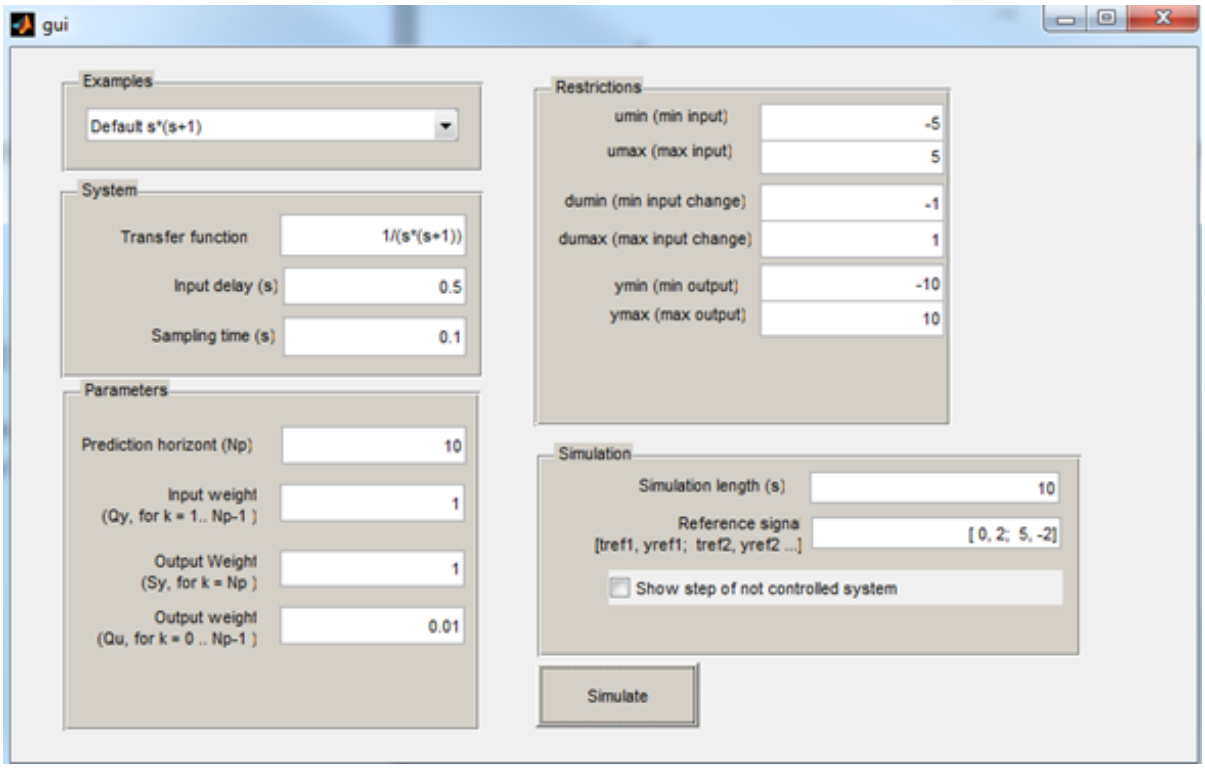
3. V prvej fáze simulačného cyklu sa prepočítajú obmedzenia systému podľa rovnice 19.

4. Následne prebehne samotná optimalizácia spustením Matlab súčasti *quadprog*, prípade Octave súčasti *qp*. Bez obmedzení by to bol výpočet podľa rovnice 13 spomenuté súčasti kvadratického programovania k tomu pridajú sústavu obmedzení.

5. Po optimalizácii sa vyberie prvý akčný zásah z vektora akčných zásahov, ktorý má dĺžku horizontu predikcie, čo je vlastne výsledok optimalizácie. Keďže v programe beží simulačný cyklus tak sa vypočíta výstup, a stav, ktoré sa zapamätajú a následne začína ďalší krok simulačného cyklu.

1.3 Overenie On-line MPC algoritmu

Po teoretickom základe a popise vytvoreného algoritmu nasleduje popis grafického rozhrania a jeho overenie. Grafické rozhranie vytvoreného programu je znázornené na obrázku 10



Obrázok 10: Grafické rozhranie k programu na testovanie MPC algoritmu.

a ponúka pre používateľa možnosti zadania:

- Systémových nastavení:
  - Automatický vyplniť určité druhy systému
  - Zadať akúkoľvek prechodovú funkciu v s alebo z oblasti systému, ktorý má byť riadený
  - Zadať dopravné oneskorenie systému
  - periódu vzorkovania
- Ladiace parametre
  - váha vstupu

  - váha výstupu pre budúce hodnoty v kroku k=0 . . . N-1, N je horizont predikcie.
  - váha výstupu pre budúcu hodnotu v kroku k=N, kde N je horizont predikcie

- Obmedzenia systému
  - minimálny vstup
  - maximálny vstup
  - minimálna zmena vstupu
  - maximálna zmena vstupu
  - minimálny výstup systému
  - maximálny výstup systému

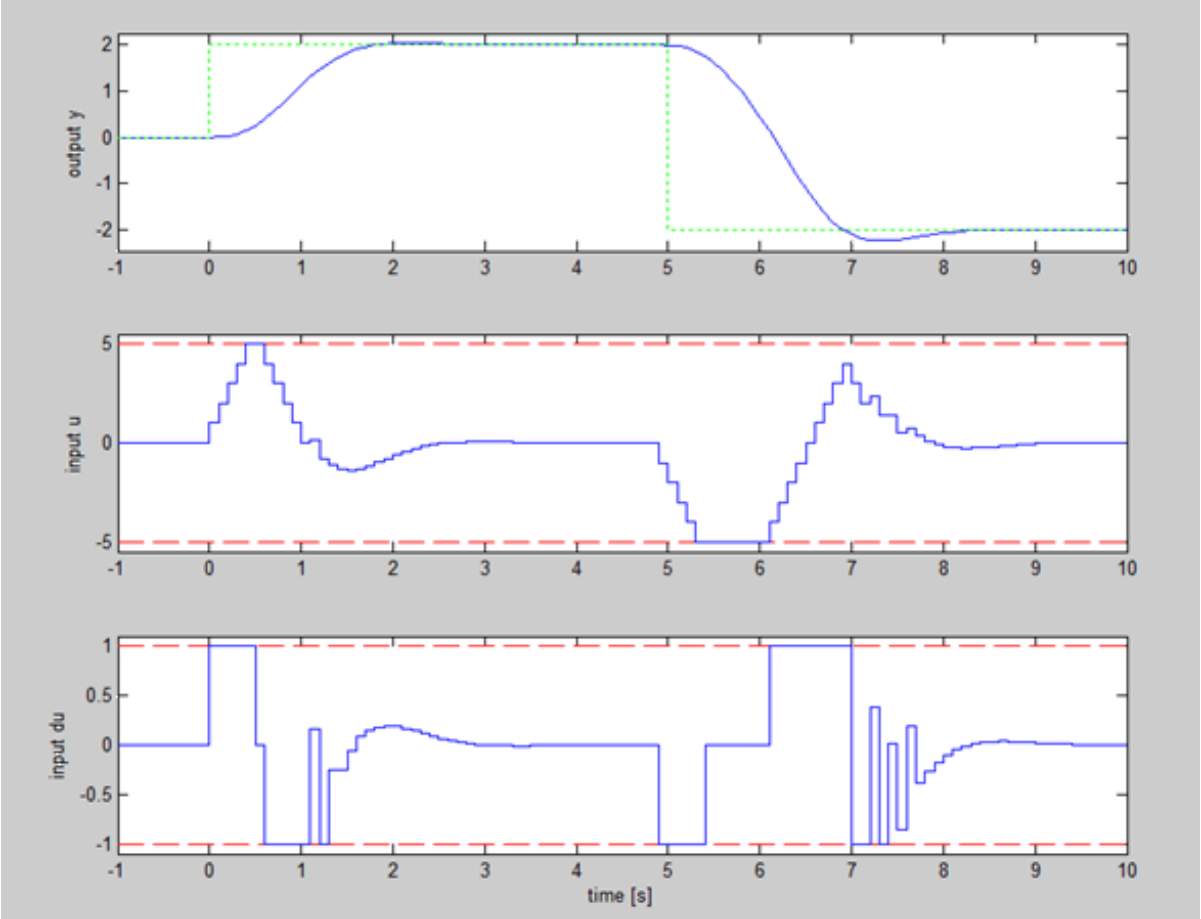
- Parametre simulácie
  - čas simulácie
  - referenčný signál zadávaný vo forme vektora dvojice hodnôt, kde prvá identifikuje čas, kedy ma zmena nastať a druhá hodnota veľkosť skoku.

Výstup z grafického rozhrania po ponechaní prednastavených hodnôt sú grafy zobrazené na obrázku 11.

V prvom grafe obrázku 11 je znázornený časový priebeh sledovania referenčnej hodnoty výstupnou veličinou. Zelenou farbou je referenčná veličina a výstupna veličina modrou. V druhom grafe obrázku 11 je znázornený riadiaci zásah systému modrou farbou a obmedzenia riadiaceho zásahu červenou farbou. V treťom grafe obrázku 11 je znázornená zmena riadiaceho zásahu modrou farbou a obmedzenia červenou. Hodnotenie kvality regulácie priamymi ukazovateľmi [27]

- Doba regulácie:
  - Pre zmenu v čase 0s (zmena 1.) z hodnoty 0 na 2 je doba regulácie 2 sekundy
  - Pre zmenu v čase 5s (zmena 2.) z hodnoty 2 na -2 je doba regulácie 3 sekundy

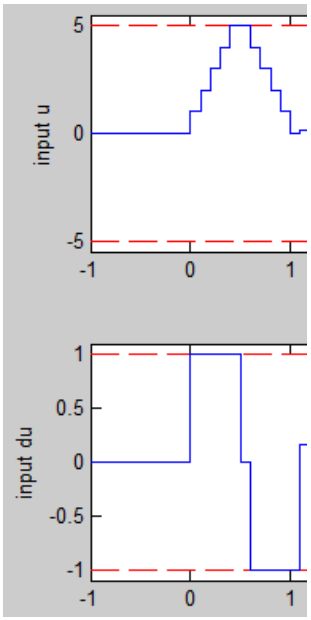
- Preregulovanie:
  - Pre zmenu 1. došlo k minimálnemu preregulovaniu.
  - Pre zmenu 2. je preregulovanie badateľné.



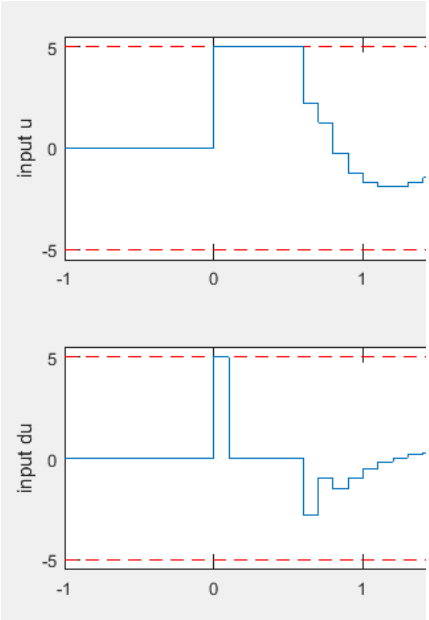
Obrázok 11: Časové priebehy výstupu systému, riadiaceho zásahu a zmeny riadiaceho zázsahu.

- Regulačná odchýlka:
  - Regulačná odchýlka je pre oba prípady 0.

Nie je tu porovnanie voči iným regulátorom, pretože to nie je zámerom tejto kapitoly. Kapitola slúži na demonštráciu funkčnosti predchádzajúcej teórie. Čo je však dôležité na tomto mieste zdôrazniť je znázornenie ako vplývajú obmedzenia na riadenie systému.



(a) Zapnuté obmedzenie



(b) Vypnuté obmedzenie.

Obrázok 12: Ukážka vplyvu obmedzenia zmeny riadiaceho zásahu na časový priebeh riadiaceho zásahu.

- **Obmedzenie veľkosti akčného zásahu** je možné pozorovať v strednom grafe obrázku 11 pri zmene 2. v čase 5.3 sekundy, kedy vidieť ako systém narazil na obmedzenie akčného zásahu a na tejto hodnote ostane až do času 6.1 sekundy. Ak by obmedzenie nebolo samozrejme by bola doba regulácie kratšia avšak akčný zásah bez obmedzení vo väčšine prípadov nezodpovedá realite.

- **Obmedzenie veľkosti zmeny akčného zásahu** je opäť možné pozorovať v strednom grafe obrázku 11 pri obidvoch zmenách referenčného signálu. Zvýraznené to je v obrázku 12a. Riadiaci zásah v tvare „schodov“ je dôsledkom tohto obmedzenia zmeny akčného zásahu. Z obrázku 12a je možné vyčítať periódu vzorkovania 0.1 sekundy, keďže je v grafe za 1 sekundu 10 zmien akčného zásahu. Prípad, že

obmedzenie zmeny akčného zásahu je nastavené na hodnotu 5 (rovnaká ako obmedzenie akčného zásahu) je zobrazené na obrázku 12b.

Na základe týchto experimentov sa potvrdzujú výhody MPC regulátora, ktoré boli spomenuté v úvode ohľadne jednoduchosti zavedenia obmedzení. V tomto má MPC bližšie spojenie s reálnym systémom ako iné regulátory napr. PID, kde kvôli obmedzeniu akčného zásahu treba robiť doplňujúce úpravy tzv. antiwindup zapojenie PID regulátora.

1.3.1 Opis systému riadenia plynovej turbíny

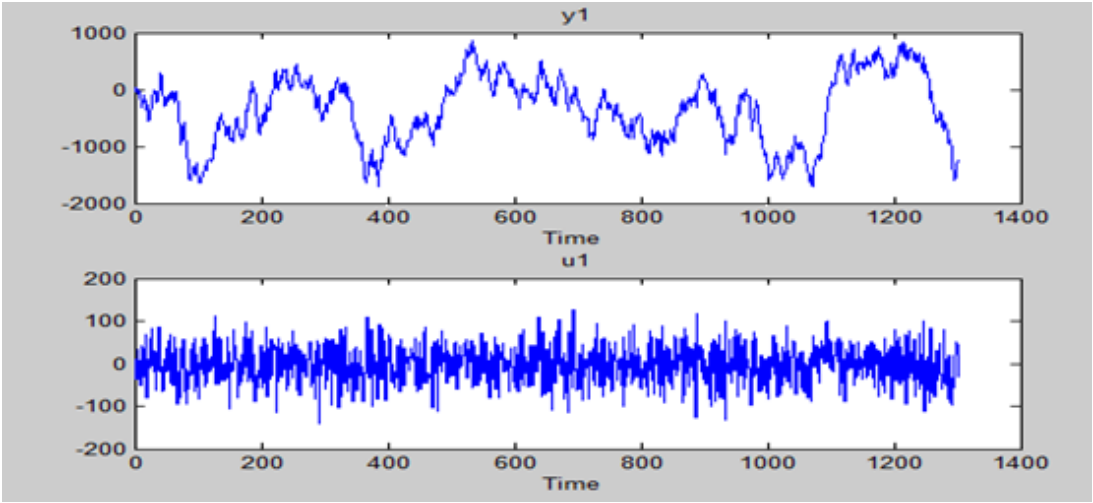
V tejto časti je overenie algoritmu na reálnom systéme z praxe, ktorý je tu popísaný. Následne je identifikovaný matematický model systému a overenie funkčnosti MPC princípov. Treba zdôrazniť, že overenie je stále prostredníctvom simulácie v prostredí Matlab.

Turbína s výkonom 1.5MW obsahuje tri hlavné časti, menovite, kompresor, spaľovaciu komorou a vysokotlakovú turbínu. V rokoch 1950 bolo priemyselné využitie turbín významne rozšírené, kvôli jej nespočetným výhodám. Napríklad neprítomnosť častí, ktoré by sa o seba odierali, nízka spotreba palív a vysoká operačná spoľahlivosť. Prvá časť turbíny zahŕňa stláčanie vzduchu na poskytnutie vysokého pomeru tlaku medzi turbínou a kompresorom, takže sa vzduch rozpína do turbíny. Zvyšovanie teploty vzduchu spaľovaním paliva spôsobuje väčšie rozpínanie horúceho vzduchu v turbíne, poskytujúc tak potrebný výstupný výkon. Pre rôzne prietoky vzduchu je limitovaný pomer vzduchu, ktorý môže byť dodaný, čo je označované ako pomer palivo/vzduch. Tento faktor obmedzuje výstupný výkon, ktorý môže byť dosiahnutý. Maximálny pomer palivo/vzduch je určený pracovnou teplotou lopatiek turbíny, ktoré sú vysoko stláčané. Zaznamenanie možnej poruchy lopatiek je dôležitý problém pri monitoringu a detekcii chýb. Primárna požiadavka na riadenie je výstupný výkon, ale neexistuje žiadny vhodný spôsob merania výkonu. Premenné súvisiace s výkonom sú riadené prostredníctvom riadenia generátora rýchlosti N_g, moduláciou prívodu paliva, kde N_g je funkciou výkonu generátora. Riadiaci zásah určuje množstvo paliva dodávaného do motora, čo je funkciou uhla otvorenia klapky θ_v. Parametre systému a regulátora: rýchlosť motora leží medzi 0 a 30 000 rpm (=500rps) táto premenná môže byť považovaná za známu a reprezentuje od 0 po 1.5MW. Vstup do systému je náklon plynovej klapky 0-60°, čo je ekvivalent toku paliva 0-625kg/h. Riadiaci rozsah výkonu je od 0 po plný výkon. Od 17 001-27 000 rpm (283,35-450rps) je považovaný za stabilný stav rýchlosti generátora.[39]

1.3.2 Identifikácia a riadenie systému plynovej turbíny

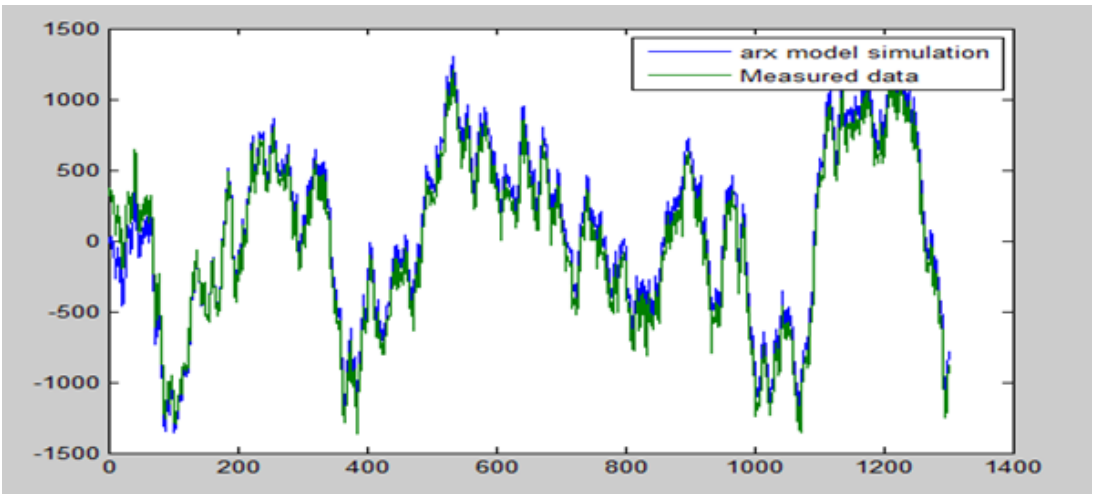
Vstupné dáta na identifikáciu systému po posunutí do 0 sú zobrazené na obrázku 13.

Premenná y_1 reprezentuje výstup - otáčky motora a u_1 vstup systému - natočenie



Obrázok 13: Časový priebeh nameraných údajov výstup a vstup systému.

klapky. Na identifikáciu systému bol použitý Matlab nástroj *ident*. Pri zisťovaní matematického modelu bolo vyskúšaných viacero metód. Najlepšie výsledky - percento zhody nameraných a simulovaných dát dal arx model. Porovnanie nameraných a simulovaných dát sú na obrázku 14.



Obrázok 14: Porovnanie simulovaných a nameraných údajov

Os x predstavuje čas v sekundách a y predstavuje výstup systému. Pomocou modelu arx sa získala nasledovná prenosová funkcia

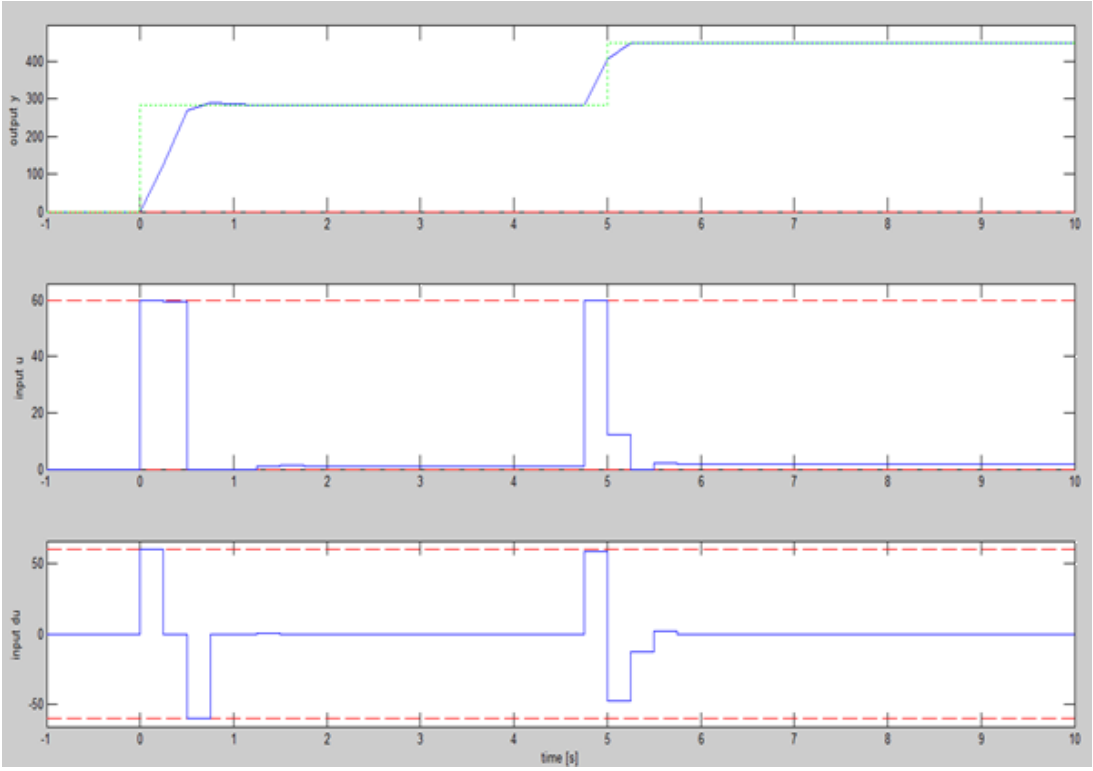
$$Gp(z) = \frac{Y(z)}{U(z)} \tag{38}$$

kde

$$\begin{aligned} \mathrm{Y}(z) &= 2.085z^{-1} + 0.3692z^{-2} \\ U(z) &= 1 - 0.9913z^{-1} + 1.287 \times 10^{-3}z^{-2} \end{aligned} \tag{39}$$

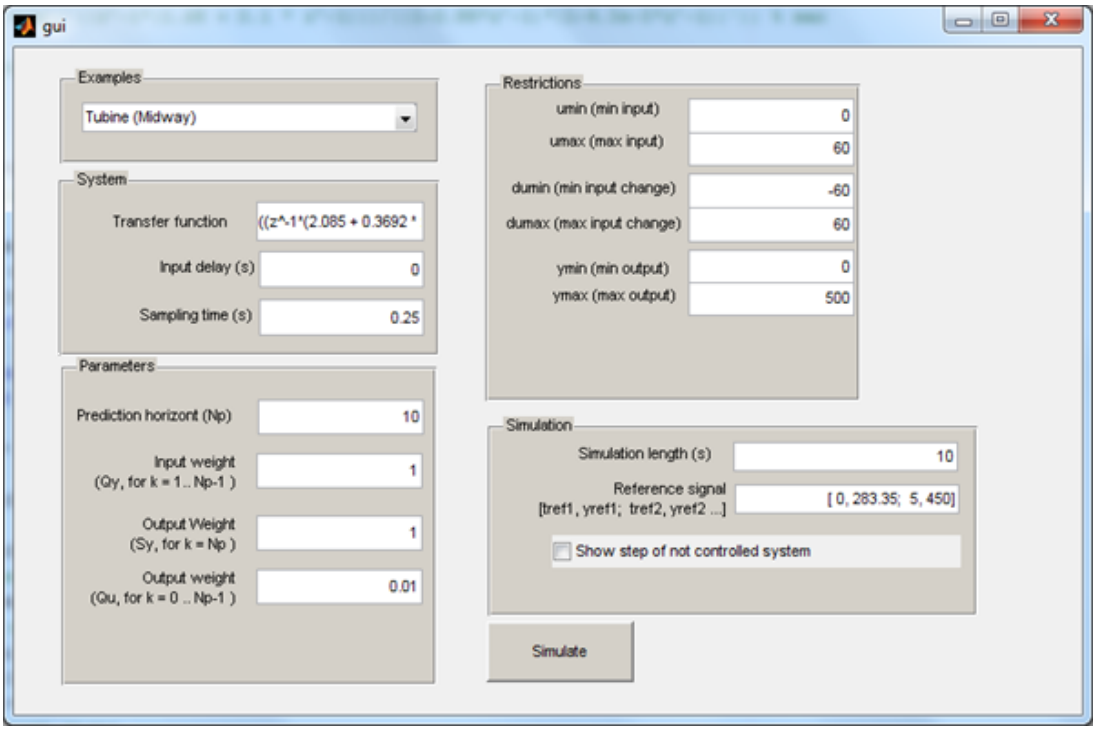
Perióda vzorkovania je $T_s = 0.25$.

Navrhnutý algoritmus bol otestovaný na reálnom systéme s prenosovou funkciou 39. Parametre simulácie sú zobrazene na obrázku 16. Časové odozvy systému s MPC regulátorom sú na obrázku 15. Konkrétne časová odozva výstupu, v tomto prípade výkon motora, meraný v rps (otáčky za sekundu) je zobrazený vo vrchnom grafe. Vstup systému, uhol náklonu plynovej klapky meraný v stupňoch je v strednom grafe a zmena vstupu v spodnom grafe. Všetky spomenuté veličiny majú v grafe modrú farbu. Zelená farba vo vrchnom grafe predstavuje referenčný signál a červené čiarkované čiary sú obmedzenia.



Obrázok 15: Časové odozvy rýchlosti motora (v jednotkách rps) s MPC regulátorom

Prenosová funkcia 39 sa zadala do grafického rozhrania do pola „transfer function“. Horizont predikcie bol nastavený na 10 vzoriek dopredu. Zo simulácii vyplýva, že je to dostatočné a efektívne, berúc do úvahy odozvu systému a kvalitu riadenia. Váhy boli nastavené na prednastavené hodnoty. Obmedzenia algoritmu sú



Obrázok 16: Parametre simulácie

- Na vstup 0-60 stupňov.
- Zmena vstupu -60 – 60 stupňov.
- Výstup systému 0 – 500 rps (30 000 rpm)

Referenčný signál je nastavený na hodnotu 283.35 rps v čase 0. Po 5 sekundách je nastavený na hodnotu 450 rps.

Z výsledkov je možné vidieť výhody prediktívneho riadenia. Keďže prediktívny regulátor je založený na optimalizácii, je možné vidieť minimálne akčné zásahy, čo môže viesť k úspore spotreby.[40]

Týmto končí simulačné overovanie algoritmu MPC regulátora vytvoreného na základe teoretických princípov popísaných v úvodných kapitolách MPC regulátora a nasleduje druhá časť práce, ktorá pripraví podklady pre praktický experiment, ktorý okrem iného zahŕňa definíciu IoT a niekoľko ďalších pojmov z oblasti softvérovej architektúry.

2 Softvérové architektúry pre pokročilé metódy riadenia

Téma softvérovej architektúry (Software architectrue) zahŕňa veľa súčastí. Moderne metódy a algoritmy riadenia vyžadujú využívanie najnovších poznatkov v oblasti IT za účelom zvýšenia kvality, rýchlosti riešení a dosiahnutia vysokej bezpečnosti a spoľahlivosti systému. Jednou z najdôležitejších úloh v oblasti využívania moderných metód riadenia je využitie pokročilých informačných technológii s optimálnou štruktúrou, čím sa zaoberá práve softvérová architektúra. Pri návrhu regulátora do praxe nie je dôležitý len samotný algoritmus regulátora, ale celé jeho zavedenie do procesu. V súčasnej dobe dochádza k užšej spätosti IT a OT, preto je nevyhnutné pri návrhu regulátora mať aj znalosti o prostredí, v ktorom regulátor má spoľahlivo fungovať. Práca sa preto v druhej časti zameriava na oblasť softvérovej architektúry. A hlavne tých oblastí, ktoré vedú do problematiky internetu vecí a zároveň súvisia s aplikáciou pokročilých metód riadenia v praxi. Nasleduje niekoľko definícií softvérovej architektúry na neskoršie pochopenie toho, aké výzvy prináša užšia väzba medzi IT (informačnými technológiami) a OT (operačnými technológiami) a aké znalosti sú nevyhnutné pri návrhu prostredia pre regulátor.

Softvérová architektúra je proces definovania štrukturovaného riešenia, ktoré spĺňa všetky technické a operačné požiadavky, zatiaľ čo optimalizuje bežné kvalitatívne vlastnosti ako je výkonnosť, bezpečnosť a ovládateľnosť. Zahŕňa rad rozhodnutí, ktoré sú založené na viacerých faktoroch a každé z týchto rozhodnutí môže mať značný dopad na kvalitu, výkonnosť, udržiavanie a celkový úspech aplikácie.

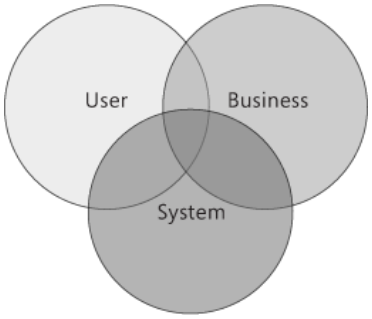
Philippe Kruchten, Grady Booch, Kurt Bittner a Rich Reitman odvodili a vylepšili definíciu architektúry, ktorá vychádza z práce Mary Shaw a David Garlan (Shaw and Garlan 1996). Ich definícia je:

Softvérová architektúra zahŕňa sadu dôležitých rozhodnutí o usporiadaní softvérového systému vrátane voľby stavebných prvkov a ich rozhraní, pomocou ktorých je systém zložený. Rozhodnutie o voľbe správania, ktoré je definované spoluprácou medzi uvedenými prvkami. Rozhodnutie o spájaní týchto štrukturálnych a behaviorálnych elementov do rozsiahlejších subsystémov a voľba architektonického štýlu, ktorý vedie toto usporiadanie. Softvérová architektúra tiež zahŕňa starosť o funkcionalitu, použiteľnosť, pružnosť, výkonnosť, možnosť znovu použitia, zrozumiteľnosť, kompromisy na ekonomické a technologické obmedzenia a tiež estetickosť. [2]

Dôležité si je uvedomiť, že softvérová architektúra je prostriedok na vytvorenie softvérového systému avšak softvérový systém je stále len prostriedok na dosiahnutie určitého

cieľa v našom prípade vytvorenie softvérového prostredia pre aplikovanie moderných metód riadenia.

Preto by systémy mali byť navrhované s prihliadaním na **používateľa**, **systém** (IT infraštruktúra) a **biznis ciele**, ako je zobrazené na obrázku 17. Pre každú z týchto oblastí by mal byť načrtnutý kľúčový scenár a identifikované dôležité kvalitatívne vlastnosti (napríklad spoľahlivosť a škálovateľnosť) a kľúčové oblasti uspokojenia alebo neuspokojenia. Vytvoriť metriky a prihliadať na ne pri meraní úspechu v každej z oblasti, všade kde je to možné.[2] V oblasti riadenia sú prioritné biznis ciele kvalita regulácie, jej dosi-



Obrázok 17: Oblasti, potrebné zvažovať pri návrhu[2]

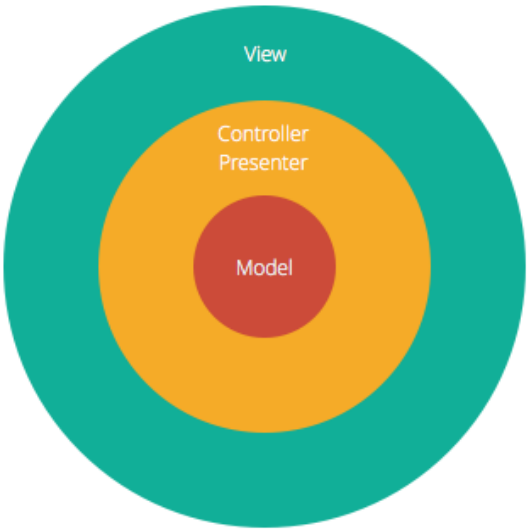
ahnutie a udržiavanie. Tiež platí, že je potrebné si nastaviť miery kvality regulácie a teda škálu od akceptovateľných až po neakceptovateľné ukazovatele. Všeobecné definície zdôraznili kľúčové oblasti, ktoré treba mať na pamäti pri návrhu softvérového systému v našom prípade pre proces riadiacu aplikáciu, avšak definície zatiaľ nepomohli s návrhom samotným, preto nasleduje vymenovanie základných typov architektúr, z ktorých si je pri návrhu možné voliť.

2.1 Základne typy architektúr

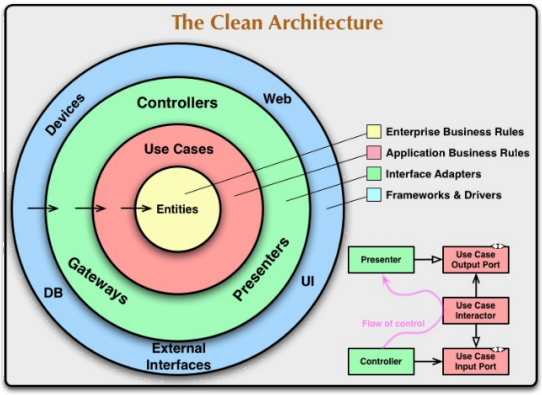
Jedno z kľúčových rozhodnutí je voľba typu architektúry. Tie najpoužívanejšie sú vymenované nižšie podľa článku [11] napísanom na základe knihy[41], ktorú napísal Mark Richards - softvér architekt s 30 ročnou skúsenosťou. V článku sa tiež uvádza, že v jednom systéme môže byť použitých viacero typov, čo je dôležitá informácia pri návrhu.

- **N-vrstvová architektúra.** Je to jeden z najpoužívanejších prístupov, pretože je to postavené okolo databázy a veľa aplikácií v biznise prirodzene potrebujú ukladať informácie v tabuľkách. Veľa z najznámejších frameworkov ako Java EE, Drupal a Express sú postavené tak, aby mali túto štruktúru na mysli, takže aplikácie, nimi vytvorené sú automaticky N-vrstvové. Zdrojový kód je usporiadaný tak, že dáta vstupujú na najvyššej úrovni a prepracujú sa cez každú vrstvu až dosiahnú najnižšiu,

čo je zvyčajne databáza. Po ceste má každá vrstva svoju úlohu, ako kontrolovanie konzistencie dát alebo formátovanie dát, tak aby ostali konzistentné. Je bežné, že rôzny programátori pracujú nezávisle na jednotlivých vrstvách. MVC (Model-View-



(a) 3 vrstvová architektúra [18]



(b) 4 vrstvová architektúra[4]

Controller) štruktúra, ktorú poskytuje väčšina obľúbených frameworkov je zjavne N-vrstvová architektúra. Nad databázou je model, ktorý často obsahuje biznis logiku a informácie o type dát databáze. Na vrchu je zobrazovacia vrstva, ktorá často pozostáva z CSS, JavaScript a HTML. V strede je ovládač, ktorý má viacero pravidiel a funkcií na transformáciu dát zo zobrazovacej vrstvy do modelu.

- **Architektúra riadená udalosťami.** Veľa programov trávi väčšinu času čakaním, kým sa niečo stane. Tento fakt špeciálne platí pre systémy, ktoré priamo spolupracujú s ľuďmi, ale rovnako je to bežné aj v oblasti sietí. Architektúra riadená udalosťami pomáha spravovať uvedené fakty tak, že sa vytvorí centrálna jednotka, ktorá prijíma dáta a potom ich deleguje do samostatných modulov, ktoré dáta spracujú. Toto odovzdanie sa nazýva vygenerovanie udalosti. Udalosť je následne spracovaná kódom tzv. event-handler.

- **„Microkernel“ architektúra.** Veľa aplikácií má základnú sadu operácií, ktoré sú znova a znova použité v iných prípadoch, ktoré závisia od aktuálneho typu dát a typu úlohy. Obľúbený nástroj na vývoj Eclipse, napríklad, najskôr otvorí súbory, pridá im poznámky, upraví ich a potom spustí pomocníka na pozadí. Nástroj je vykonávaním týchto operácií známy a s kódom napísaným v jazyku Java na jedno stlačenie tlačítka sa kód skompiluje a spustí. V tomto prípade, základný program

na zobrazovanie a upravovanie súboru sú súčasťou microkernel-u. Java kompilátor je extra časť, ktorá je pridaná na podporu základných čŕt microkernel-u. Ostatní vývojári vyvinuli ďalšie časti, aby bolo možné vyvíjať aj v iných jazykoch s inými kompilátormi. Často krát sa kompilátor ani nepoužíva, ale využívajú len funkcie na úpravu súborov. Špeciálne pridaná funkcionalita sa nazýva plug-in. Často je tento prístup nazývaný aj Plug-in architektúra.

- **„Microservice“ architektúra.** Softvér môže byť ako malý slon. Keď je malý je milý a zábavný, ale keď dospeje, je ťažké ho viesť a bráni sa zmene. Návrh Microservice architektúry pomáha vývojárom predísť tomu, aby sa z ich malých programov stali ťažkopádne, monolitické a neflexibilné programy. Preto na miesto vytvárania jedného veľkého programu je cieľ vytvoriť množstvo rozličných malých programov a vždy keď chce niekto pridať funkcionalitu, tak vždy pridať malý program. Tento prístup je podobný Microkernel a udalosťami riadenému prístupu, ale je používaný zvyčajne, keď rozličné úlohy sú ľahko oddeliteľné. Vo veľa prípadoch, rozličné úlohy môžu vyžadovať rôzny čas na spracovanie a môžu sa líšiť v spôsobe použitia. Napríklad servery spoločnosti Netflix, ktoré poskytujú obsah zákazníkom (filmy a videá) majú oveľa väčšiu záťaž v piatok a sobotu večer, takže musia byť pripravený zvýšiť výpočtové a sieťové kapacity. Servery, ktoré sledujú vrátenie požičaných DVD, na druhej strane, robia svoju robotu cez týždeň hneď ako pošta doručí príchodzie zásielky. Ak sa to implementuje ako oddelené služby, Netflix cloud ich môže nezávisle škálovať podľa dopytu.

- **„Space-based“ architektúra.** Veľa aplikácií, ktoré sú postavené okolo databázy a fungujú správne pokiaľ databáza stíha spracovávať záťaž. Keď však databáza prestane stíhať zapisovať veľa transakcií, celá aplikácia spadne. „Space-based“ architektúra je navrhnutá tak, aby predišla zlyhaniu pri veľkej záťaži rozložením spracovania a ukladania na viacero serverov. Dáta aj zodpovednosť za možnosť zavolania služby sú rozdistribuované na viacero uzlov. Niektorí architekti používajú širokejší pojem „cloud“ architektúra. Táto architektúra podporuje prípady, kedy je ťažké predikovať nárast požiadaviek, ktorý by databáza nestíhala spracovávať. Uchovávanie informácie v RAM umožňuje vykonávať veľa úloh rýchlejšie a zjednodušuje zdieľanie medzi uzlami. Avšak distribuovaná architektúra robí niektoré analýzy komplikovanejšími. Výpočet, ktorý musí prebehnúť cez všetky dáta, napríklad výpočet priemeru alebo vytváranie štatistickej analýzy musí byť rozdelený na podúlohy cez všetky uzly a keď skončia zoskupenie dát je potrebné.

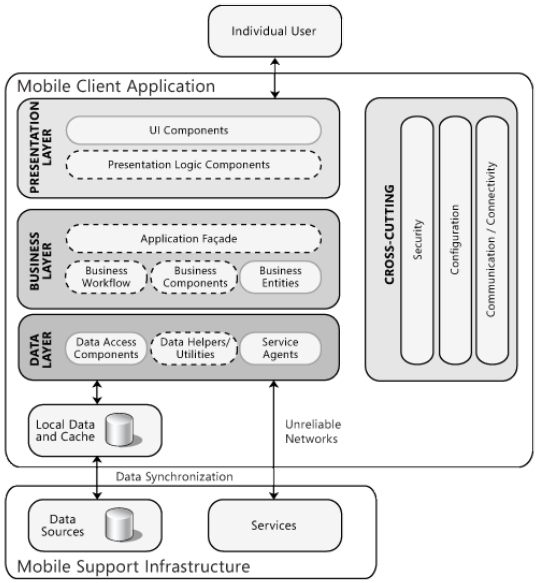
2.2 Základné typy aplikácií

Ďalšie z kľúčových rozhodnutí je voľba typu aplikácie. Na výber podľa článku [3] sú:

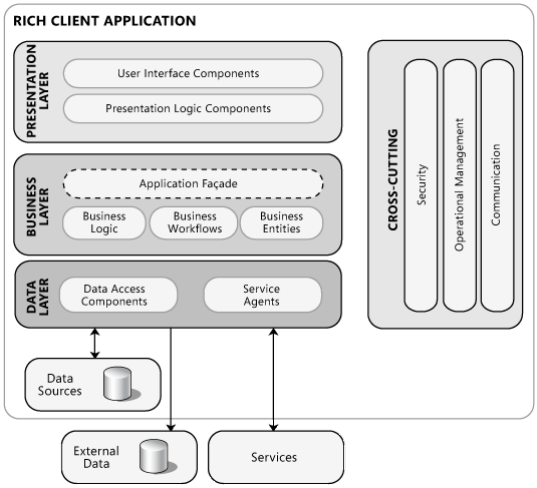
- **Mobilná aplikácia obrázok 19a.** Aplikácie tohto typu môžu byť vyvíjané ako tenký alebo tučný klient. Mobilná aplikácia vo forme tučného klienta môže podporovať scenáre bez pripojenia alebo s občasným pripojením. Webové aplikácie alebo tenkí klienti podporujú iba scenáre s pripojením. Zdroje, ktorými disponujú mobilné zariadenia sa často ukazujú ako obmedzenie pri návrhu mobilných aplikácií.
  Výhody:
  - Podpora mobilných zariadení.
  - Dostupnosť a jednoduchosť použitia pre používateľov mimo kancelárie.
  - Podpora pre scenáre bez pripojenia a s občasným pripojením.

  Na zváženie:
  - Limity ohľadne zadávania údajov a navigácie v aplikácii.
  - Limitovaná šírka zobrazovanej plochy.
- **Tučná klientská aplikácia obrázok 19b.** Aplikácie tohto typu sú zvyčajne vyvíjané ako stand-alone aplikácie s grafickým rozhraním, ktoré zobrazuje dáta prostredníctvom rôznych ovládacích prvkov. Tuční klienti sú väčšinou navrhnutí na scenáre bez pripojenia alebo s občasným pripojením, keď potrebujú prístup k vzdialeným dátam alebo funkcionalite.
  Výhody:
  - Možnosť využívať zdroje klienta.
  - Lepšia odozva, bohatá funkčnosť používateľského rozhrania a lepší používateľský zážitok.
  - Dynamická a responzívna interakcia.
  - Podpora scenárov bez pripojenia alebo s dočasným pripojením.

  Na zváženie:
  - Komplikovanejšie nasadenie. Avšak existuje množstvo dostupných inštalačných možnosti ako ClickOnce, Windows Installer a XCOPY.
  - Problém s nasadzovaním nových verzií v čase.



(a) Mobilná aplikácia [3]



(b) Clientská aplikácia [3]

Obrázok 19

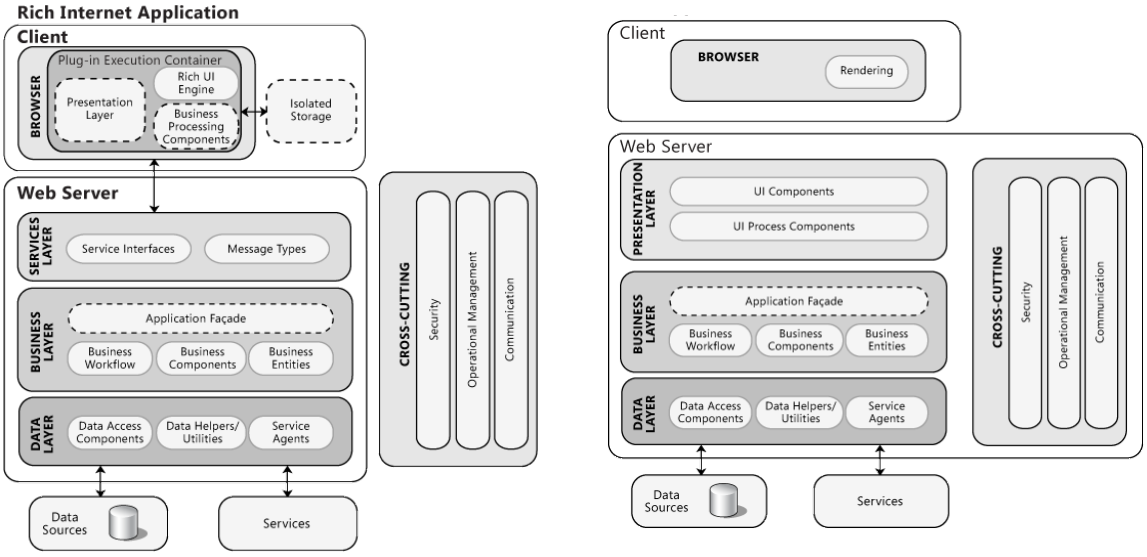
  - Závislé od platformy.

- **Tučná internetová aplikácia obrázok 20a.** Aplikácie tohto typu sú vyvíjané tak, aby podporovali viacero platforiem a prehliadačov tak, že zobrazujú multimédia alebo iný grafický obsah. Tučné internetové aplikácie bežia v prehliadači, čím sú obmedzené pristupovať k niektorým zdrojom klienta.
  Výhody:
  - Rovnaké schopnosti užívateľského rozhrania ako tuční klienti.
  - Podpora multimédií a stream medií
  - Jednoduché nasadzovanie s rovnakými možnosťami distribúcie ako weboví klienti.
  - Jednoduchý upgrade na novú verziu.
  - Podpora cez väčšinu platforiem a prehliadačov.

  Na zváženie:
  - Kladie vyššie nároky na klienta ako webové aplikácie.
  - Obmedzenia na využívanie zdrojov klienta oproti tučnej klientskej aplikácii.
  - Vyžaduje mať na klientovi nasadený vhodný runtime framework.

- **Webová aplikácia obrázok 20b.** Aplikácie tohoto typu zvyčajne podporujú scenáre s pripojením, podporujú viacero prehliadačov, ktoré bežia na rôznych operačných systémoch.
  Výhody:
  - Široká dostupnosť pre väčšinu platforiem a užívateľské rozhranie je vytvárané prostredníctvom štandardov.
  - Jednoduchosť nasadenia a správy zmien.

  Na zváženie:
  - Závislé od nepretržitého sieťového spojenia.
  - Komplikácie s poskytnutím bohatého (rich) užívateľského rozhrania.



(a) Tučná internet aplikácia [3]

(b) Web aplikácia [3]

Obrázok 20

- **Servisná aplikácia obrázok 21.** Služby vystavujú zdieľanú biznis funkcionalitu a umožňujú klientom pristupovať k nim z lokálneho alebo vzdialeného systému. Operácie služieb sú volané prostredníctvom správ založených na XML schémach, posielaných cez transportnú vrstvu. Cieľom týchto aplikácií je dosiahnutie voľných väzieb medzi klientom a serverom.
  Výhody:
  - Interakcie medzi klientom a serverom je prostredníctvom voľných väzieb.

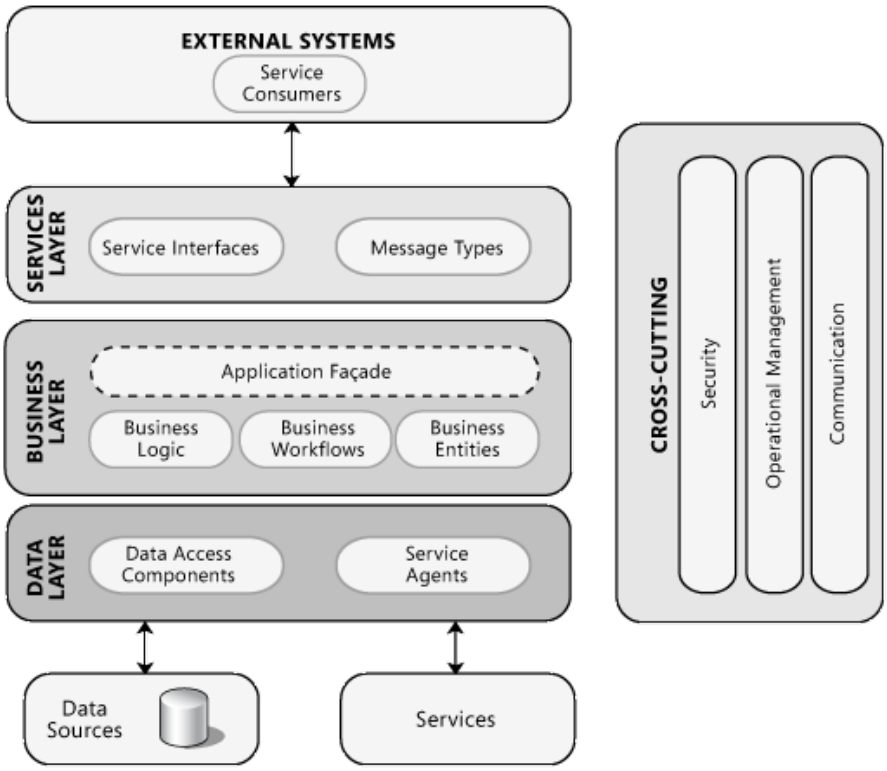
– Môže byť použitá rôznymi nezávislými aplikáciami.

– Podpora pre interoperabilitu.

Na zváženie:

– Nie je podpora pomocou užívateľského rozhrania.

– Závisí od sieťového pripojenia.



Obrázok 21: Aplikácia poskytujúca službu [3]

2.2.1 Servisná aplikácia

Servisným aplikáciam je v práci venovaná samostatná kapitola, pretože je to najvhodnejší spôsob akým moderné metódy riadenia, či už jeden alebo viac regulátorov integrovať do softvérového systému. Servisné aplikácie majú svoje miesto vo väčšine enterprise architektúr. Enterprise architektúra je koncepčný návrh, ktorý definuje štruktúru a prevádzkovanie spoločnosti.

Zámer enterprise architektúry je rozhodnúť ako môže organizácia najefektívnejšie dosiahnuť aktuálne a budúce ciele [30]. Inými slovami ide o architektúru podnikových informačných systémov, ktoré zvyčajne pozostávajú z viacerých súčasti. Príkladom týchto súčasti môže byť:

- **CRM aplikácia.** Podľa APICS slovníka [7] je CRM definované ako úložisko a analýza informácií navrhované pre podporu predajných a marketingových rozhodnutí, na pochopenie a podporu potrieb existujúcich a potenciálnych zákazníkov. Zahŕňa správu užívateľských účtov, môže zahŕňať katalóg produktov, zadávanie objednávky, spracovanie a úpravu platieb a iné funkcie.[6]. Všetky, ktoré sú v definícii uvedené ako „môže zahŕňať“, závisí od konkrétnej implementácie, či dané funkcionality CRM obsahuje alebo nie. Pretože každá zo spomenutých funkcionalít môže byť ako samostatná servisná aplikácia. V tejto práci sú uvedené ako samostatné časti a CRM sa tu obmedzí na správu užívateľských účtov.

- **Katalóg produktov.** Na manažovanie závislosti medzi produktami a ovplyvňovanie typov zliav a teda výpočet ceny za produkty býva v podnikoch samostatná aplikácia

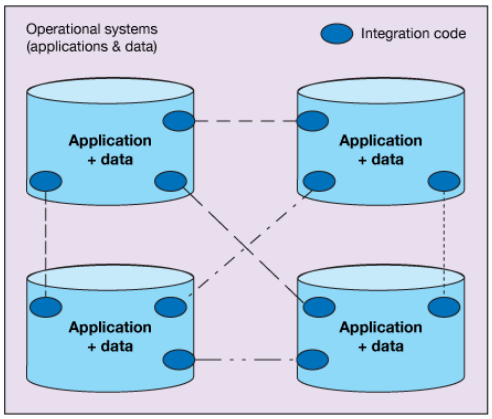
- **Účtovná aplikácia.** Aplikácia na vedenie účtovníctva

- **Správa majetku.** Aplikácia na správu majetku.

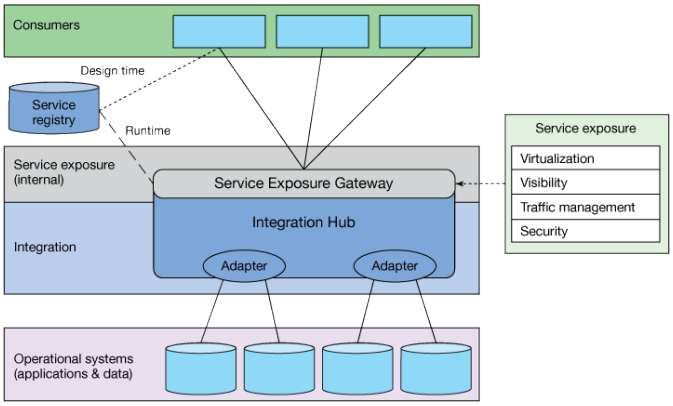
- **ERP aplikácia.** (Plánovanie zdrojov v podniku je pojem používaný v priemysle pre širokú škálu činností, ktoré pomáhajú organizácii spravovať jej biznis. Dôležitý cieľ je pomôcť, aby tok informácií bol nastavený tak, že biznis rozhodnutia môžu byť robené na základe poskytnutých dát. ERP aplikácie sú robené tak, aby zbierali a zatrieďovali dáta z rôznych úrovni organizácie a poskytovali tak manažmentu náhľad na kľúčové ukazovatele výkonnosti tzv. KPIs v reálnom čase.[31]

Súčastí môže byť samozrejme viac a každá organizácia si podľa svojich potrieb vyberá tie súčasti, ktoré potrebuje. Väčšina spomenutých súčasti je dodávaná ako servisná aplikácia. Aplikácie medzi sebou môžu komunikovať prostredníctvom vystavených rozhraní. Pri malom počte vystavených rozhraní sa často používa **Point to Point 22a** architektúra, kedy neexistuje centrálny bod vystavenia služby, ale ak ktorákoľvek aplikácia potrebuje zavolať inú, tak ju priamo zavolá. Takýto prístup je však krátkozraký, lebo prax ukazuje, že počet vystavených služieb časom vždy pribúda, preto je dnes rozšírené používať tzv. SOA - **Service oriented architektúru 22b**, čo je iný názov spomínanej „Microservice“ architektúry s centrálnym bodom vystavovania služieb tzv. ESB - Enterprise service bus. Existuje viacero spôsobov ako SOA implementovať.

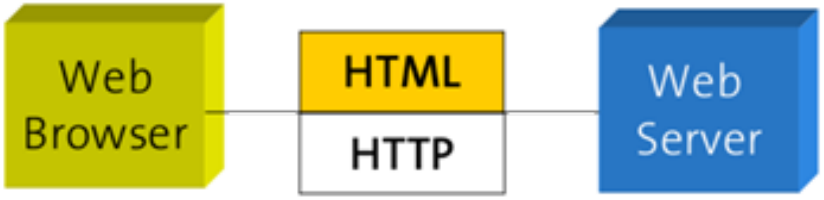
- Najznámejší spôsob implementácie SOA, ktorí dnes používa väčšina užívateľov informačných technológií je implementácia protokolu WWW - **World Wide Web**



(a) Point to Point [13]



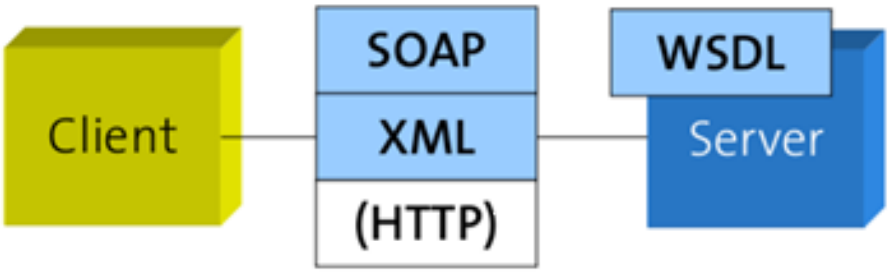
(b) SOA [13]



Obrázok 23: WWW [12]

v skratke často označovaný len WEB. Obrázok 23 znázorňuje princíp fungovania. Prehliadač (web browser) je v pozícii klienta, konzumera služby, ktorú poskytuje Web Server. Web Server je označenie takého počítača, na ktorom beží servisná aplikácia, ktorá vie na základe definovaných pravidiel poskytnúť požadovaný obsah. Najpoužívanejšie aplikácie, ktoré z počítača spravia web server, v čase písania práce, z prieskumu robenom vo februári 2016 [8] sú Apache, Nginx, Microsoft IIS. Ich preferencie striedavo kolíšu podľa úspechu najnovších verzií. Pod službou sa v tomto prípade rozumie poskytnutie HTML dokumentu prostredníctvom aplikačného komunikačného protokolu HTTP.

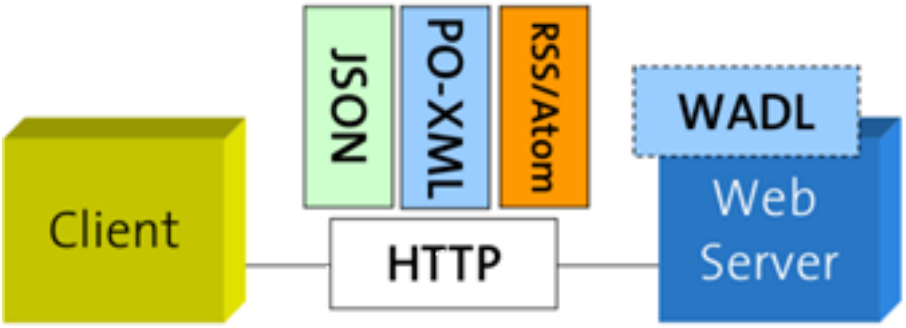
- Ďalší rozšírený protokol, ktorý implementuje SOA architektúru je **protokol SOAP**. Princíp fungovania je na obrázku 24. Klient je v tomto prípade generickejší. Zvyčajne je to aplikácia, ktorá má XML parser (prekladač) a schopnosť komunikovať po sieti HTTP protokolom. Rovnako server je generickejší. SOAP je v tomto prípade pre server „len“ príručka ako službu vystaviť. Pod službou sa rozumie vystavenie RPC (remote procedure call), čo znamená umožnenie volania vzdialenej procedúry. Čo bude procedúra robiť je plne v rukách programátora. Aj v SOAP je pre transportnú vrstvu použitý HTTP protokol, obsahom správ sú XML objekty so špeci-



Obrázok 24: SOAP [12]

fickým tvarom, ktorý SOAP definuje. Postup komunikácie je taký, že klient vytvorí XML objekt v ktorom definuje vstupné parametre do volania vzdialenej procedúry odošle požiadavku, na serveri sa vykoná procedúra a do odpovede sa pošle výsledok spracovania vzdialenej procedúry vo forme XML objektu, ktorý klient spracuje a na základe toho vie ako volanie dopadlo. Medzi najväčšie výhody tohto protokolu patrí možnosť striktnej validácie správnosti formátu správ, ktoré si klient a server vymieňajú. Formát správ a operácie, ktoré server poskytuje sú zadefinované vo WSDL dokumente.

- Iná implementácia SOA architektúry uvádzaná v tejto práci je **architektonický štýl REST**. Možností samozrejme existuje viacero, ale ako bolo na začiatku tejto kapitoly uvedené, cieľom je pripraviť podklad pre vysvetlenie Internetu vecí a pre implementáciu MPC v ňom. REST je skratka od REpresentational State Transfer. Ako je možné vidieť na obrázku 25 aj tu vystupuje generický klient a generický server ako v prípade SOAP protokolu. Základný rozdiel oproti SOAP je, že HTTP



Obrázok 25: REST [12]

je využívaný nie len ako transportný protokol, ale ako aplikačný protokol, takže sa využíva celý jeho potenciál. V praxi to znamená, že operácie, ktoré server poskytuje sú unifikované HTTP protokolom a nie je nutné, aby ich klient vyťahoval z

WSDL dokumentu. Ďalšia výhoda, že HTTP je využívaný ako aplikačný protokol je, možnosť definovania aký obsah server pošle. Nie je teda obmedzenie na XML dokumenty, ale je možnosť posielať JSON objekty, binárne objekty a všetky ostatné typy správ podporované HTTP protokolom. Z uvedeného tiež vyplýva, že klientom môže byť samotný prehliadač respektíve implementácia klienta sa značne zjednodušuje. Možnosť posielať JSON objekty tiež napomáha zjednodušeniu klienta, ktorý, ak sa hovorí o web aplikácii, je zvyčajne napísaný v jazyku Javascript. Takže preklad posielanej správy na dátový typ klienta je priamočiary, bez používania ďalšieho prekladača.

Kvôli uvedeným vlastnostiam je používanie REST rozhrania s JSON objektom ako formátom správ rozšírené pri tvorbe servisnej aplikácie, ktorá poskytuje rozhranie pre webovú aplikáciu.

Fakty ohľadne enterprise architektúry, SOA architektúry a REST princípov, na prvý pohľad nesúvisiace s témou boli volené zámerne. Okrem toho, že trend aktuálnych IT systémov, je stavať ich v duchu uvedených faktov, tak aj zavádzanie moderných metód riadenia sa javí ako najoptimálnejšie týmto spôsobom, a teda pripájanie modulov (služieb) do systému, prostredníctvom REST princípov. Enterprise architektúra je navyše spomínaná z dôvodu toho, že zavedenie moderných metód riadenia nebude len pripojenie do jednoduchého systém, ale často veľkých podnikových IT prostredí. Posledný dôvod uvádzania týchto faktov je práve využívanie servisných aplikácií v IoT prostredí. Definíciou IoT pokračujeme v nasledujúcej kapitole.

2.3 Internet vecí - IoT

Organizácia IEEE vydala celý dokument s názvom „Smerom k definícii Internetu vecí“. Cieľom dokumentu je podať plnohodnotnú definíciu IoT v rozmedzí od malých lokálnych systémov, obmedzených na konkrétnu lokalitu, až po globálny systém, ktorý je distribuovaný a poskladaný z komplexných systémov. V dokumente je možnosť nájsť prehľad základných požiadaviek na architektúru IoT [22]. Podľa spoločnosti Gartner, ktorá je svetový líder v oblasti výskumu informačných technológií je IoT sieť fyzických objektov, ktoré majú vstavanú technológiu na komunikovanie a snímanie alebo interakciu s ich vnútornými stavmi alebo vonkajším prostredím [14]. Ešte definícia podľa stránky Techopedia: IoT je koncept, ktorý popisuje budúcnosť, kde každodenné fyzické objekty budú pripojené do internetu a budú sa vedieť sami identifikovať iným zariadeniam. Pojem je úzko spojený s RFID technológiou ako spôsobom komunikácie, aj keď to môže zahŕňať iné technológie na snímanie, bezdrôtové technológie alebo QR kódy.

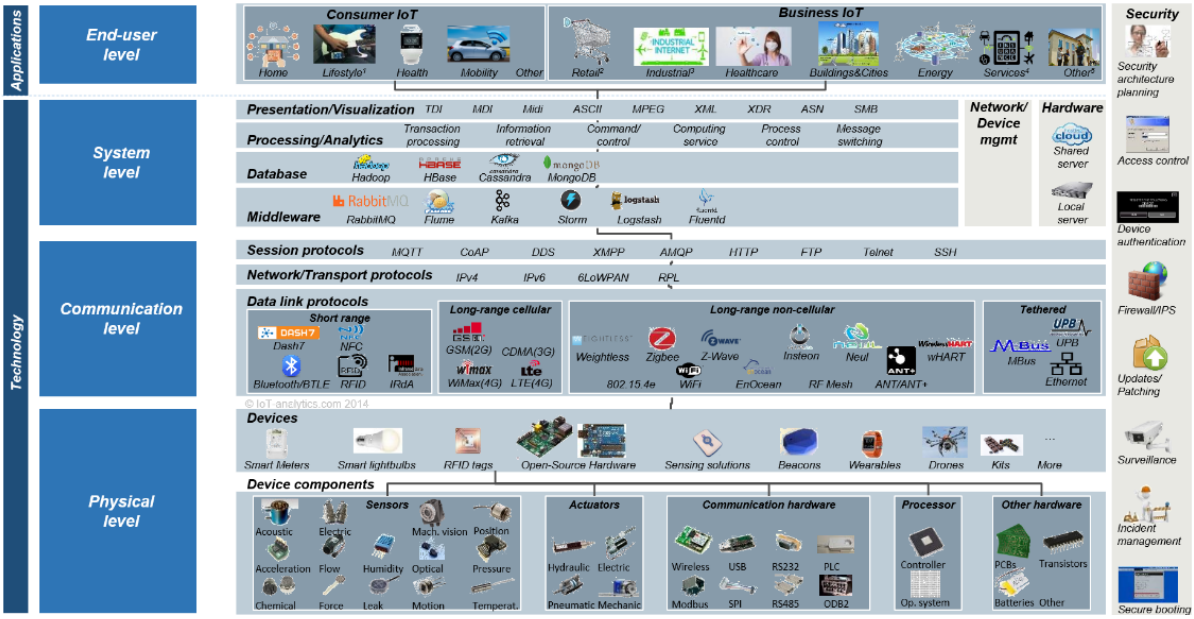
IoT je významné, pretože ak objekt sa vie sám digitálne reprezentovať stáva sa z neho niečo viac ako objekt samotný. Objekt sa už nevzťahuje len na nás, ale je spojený s okolitými objektami a dátami v databázach. Keď veľa objektov spolupracuje, je možné to nazvať ako inteligencia okolitého prostredia „ambient intelligence“ [32]. Viacero ďalších technologických spoločností majú ich definíciu. Cieľom tejto práce nie je vytvoriť ďalšiu definíciu, ale identifikovať spoločné črty väčšiny definícií, ale hlavne načrtnúť možnosti zavedenia moderných metód riadenia do praxe.

- Prvá základná črta definícii je, že v nej vystupujú objekty, ktoré môžu **snímať a ovplyvňovať okolie**. Na dosiahnutie tejto črty je potrebné, aby objekty mali senzory a akčné členy, čo je jeden z hlavných záujmov odboru automatizácia, pre ktorý to nie je nič nové. Dostupnosť a klesajúca cena snímačov a akčných členov umožňuje ich umiestňovanie aj na objekty mimo priemyslu, čo otvára priestor pre Internet vecí.

- Druhá základná črta definícii je **prepojenie**. Opäť v automatizácii a priemysle to nie je nič nové, veď koľko priemyselných štandardov rieši prepojenie senzorov a akčných členov na výrobných linkách po celom svete. Hnacou silou rozvíjajúceho sa Internetu vecí sú nové možnosti drôtového a bezdrôtového pripojenia dostupné pre koncových používateľov, klesajúce náklady na prevádzku sietí a klesajúca cena zariadení nazývaných - edge device, gateway alebo agregátor, ktoré umožňujú zber a posielanie dát do dátových centier.

- Ďalšia spoločná črta je vyústením dvoch predošlých a to **vytváranie inteligentného prostredia**. Keď bežné objekty vedia snímať rôzne fyzikálne veličiny, zosnímané hodnoty prostredníctvom pripojenia môžu poslať prostredníctvom agregátora do dátového centra, v dátovom centre sú hodnoty zoskupené a pripravené na urobenie analýz a štatistík, na základe ktorých, ak je možné robiť rozhodnutia v reálnom čase alebo kvázi reálnom čase, tak je spätne možnosť ovplyvniť akčný člen, čo vo veľkom môže znamenať vytvorenie inteligentného prostredia. Zabezpečenie tejto črty je predmetom oblasti ukladania a spracovania dát. V tomto bode sa IoT často spája s pojmom Big Data. Definícia tohto pojmu má viacero verzií podobne ako definícia IoT. Zvolila sa aspoň jedna od spoločnosti Gartner pre ilustráciu. Big data sú informácie veľkého objemu, veľkej rýchlosti a/alebo veľkej variability, ktoré si vyžadujú rentabilné, inovatívne formy spracovania informácii, ktoré umožňujú väčší náhľad, napomáhať rozhodovaniu a automatizácii procesov [29]. V implementačnej časti je jedna časť venovaná problematike Big data.

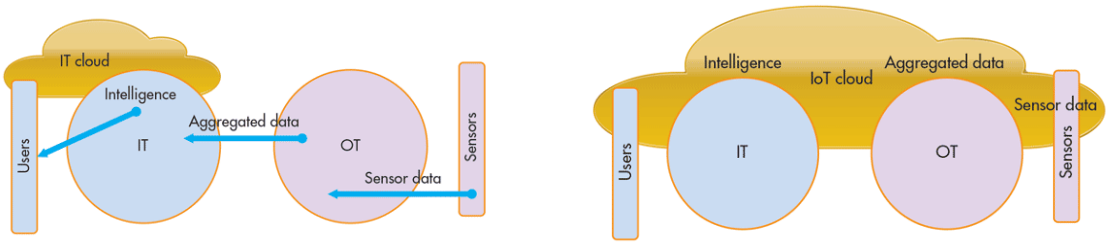
Množina technológii, určite nie všetkých, ktoré môže IoT zhŕňať je ďalej na obrázku 26. Na obrázku je možné vidieť súčasti, ktoré sme uviedli ako spoločné črty IoT definícií.



Obrázok 26: IoT súhrn techológií [34]

Pri pohľade na tento obrázok si je dôležité uvedomiť, koľko rôznych typov softvérov, jazykov a architektúr tu prichádza do interakcie. Začínajúc zo spodu. Senzory a akčné členy podľa zložitosti majú buď len nahratý jednoúčelový firmware, ktorý umožňuje len meranie prípadne vykonávanie akčných zásahov a posielanie dát. Alebo majú určitý druh operačného systému, ktorý umožňuje na tejto úrovni do merania a akčných zásahov robiť úpravy. Ak sa spomenie posielanie dát, tak opäť je tam softvér zodpovedný za odosielanie a prijímanie dát, pričom ako je možné vidieť na obrázku, tak je viacero úrovni abstrakcie odosielania údajov. Na systémovej úrovni sú to rôzne databázové systémy, systémy na spracovanie správ (message queue), analytické nástroje a vizualizačné rozhrania. Nakoniec až na najvyššej úrovni sú aplikácie, ktoré spojením rôznych pojmov z nižšej vrstvy dokopy majú pridanú hodnotu pre koncového používateľa. Na prvý pohľad by sa dalo povedať, že IoT sa nijako nelíši od existujúcich priemyselných inštalácií, preto je ďalej znázornené, v čom sa IoT líši od tradičnej architektúry priemyselných systémov na obrázkoch 27a a 27b.

Tradičná priemyselná architektúra na posunutie dát do platformy informačných technológií (Information Technology - IT), ktorá vie robiť analýzu dát, poskytovať ich používateľovi aj vystavovať do cloud prostredia, používa vstavaný softvér na platforme obslužných technológií (Operational Technology - OT). IoT architektúra práve posúva väčšiu



(a) Tradičná architektúra [17]

(b) IoT architektúra [17]

Obrázok 27

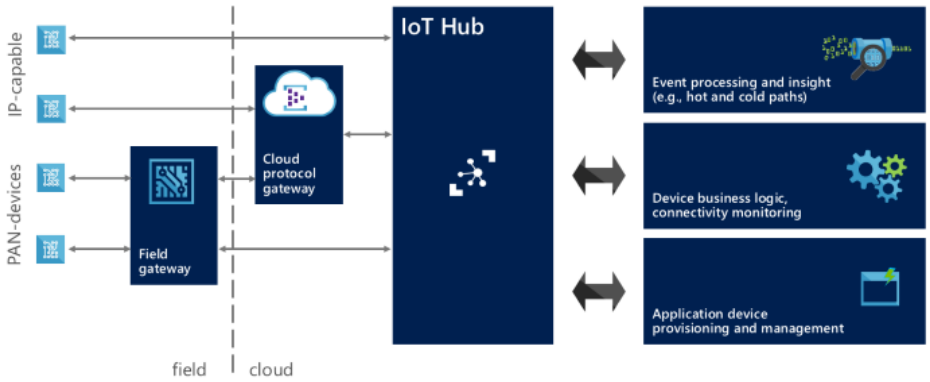
časť inteligencie systému z IT strany do OT strany. Toto umožňujú mikroprocesory a vstavané platformy s jednoduchým prístupom do cloud prostredia a rovnako s jednoduchým prístupom ku autorizovaným zariadeniam a používateľom. [17]. Táto definícia potvrdzuje to, čo sme už vyslovili, že ide o užšie prepojenie IT a OT.

Základné ciele IoT architektúr je teda zosnímané dáta čo najskôr, najbezpečnejšie a najspoľahlivejšie poslať ďalej na miesto, kde môžu prejsť analýzou, či už je to cloud alebo dosť často to býva aj samotný agregátor, ktorý má dostatočný výkon na určitý druh analýz a na základe analýzy ovplyvniť akčné členy, aby sa optimalizoval proces, náklady, zdroje atď. Aktuálny trend vo svete je, že veľké technologické firmy sa snažia spraviť univerzálnu IoT platformu a podchytiť si tak čo najviac zákazníkov. Veľké firmy typu Microsoft, Amazon, HP už pripravili svoje platformy, ktoré sa líšia v použitých technológiách, ale princíp ostáva rovnaký. Na demonštráciu sa uvádza porovnanie dvoch hotových platforiem Azure IOT HUB od Microsoft a AWS IOT od Amazonu v tabuľke 1 a na obrázku 28.

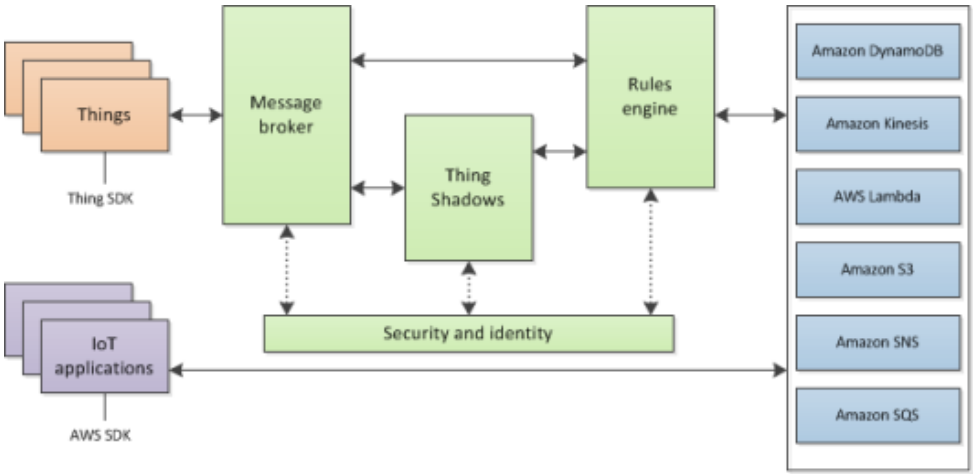
Tabuľka 1: Porovnanie dvoch IOT platforiem [16]

Produkty	MS AZURE IOT HUB	AMAZON AWS IOT
Protokoly	HTTP, AMQP, MQTT, vlastné protokoly	HTTP, MQTT
Komunikačné vzory	Diaľkové meranie, príkazy	Diaľkové meranie, príkazy
Certifikované platformy	Intel, Raspberry Pi 2, Freescale, Texas Instruments, MinnowBoard, BeagleBoard, Seeed, resin.io	Broadcom, Marvell, Renesas, Texas Instruments, Microchip, Intel, Mediatek, Qualcomm, Seeed, BeagleBoard
SDK/jazyky	.Net, Java, C, NodeJS	C, NodeJS

Na portály Devexperience a stránke [16], kde bolo porovnanie vykonané, definujú



(a) Azure IOT HUB [16]



(b) AWS IOT [16]

Obrázok 28

tieto súčasti IoT architektúry:

Kompletné IoT riešenie pozostáva z viacero častí. Ako prvé je potrebné prijať všetky udalosti a dáta poslané zo zariadení a to je veľký problém, lebo v dobe internetu vecí je potrebné myslieť na škalovateľnosť v stovkách, tisíckach, miliónoch a ... miliardách zariadení. Preto je potrebné mať **prijímací** (ingestion) systém, ktorý je schopný spracovať dáta veľmi rýchlo bez spomalenia celého procesu. Takto sa nazýva komunikačný vzor **diaľkové meranie**.

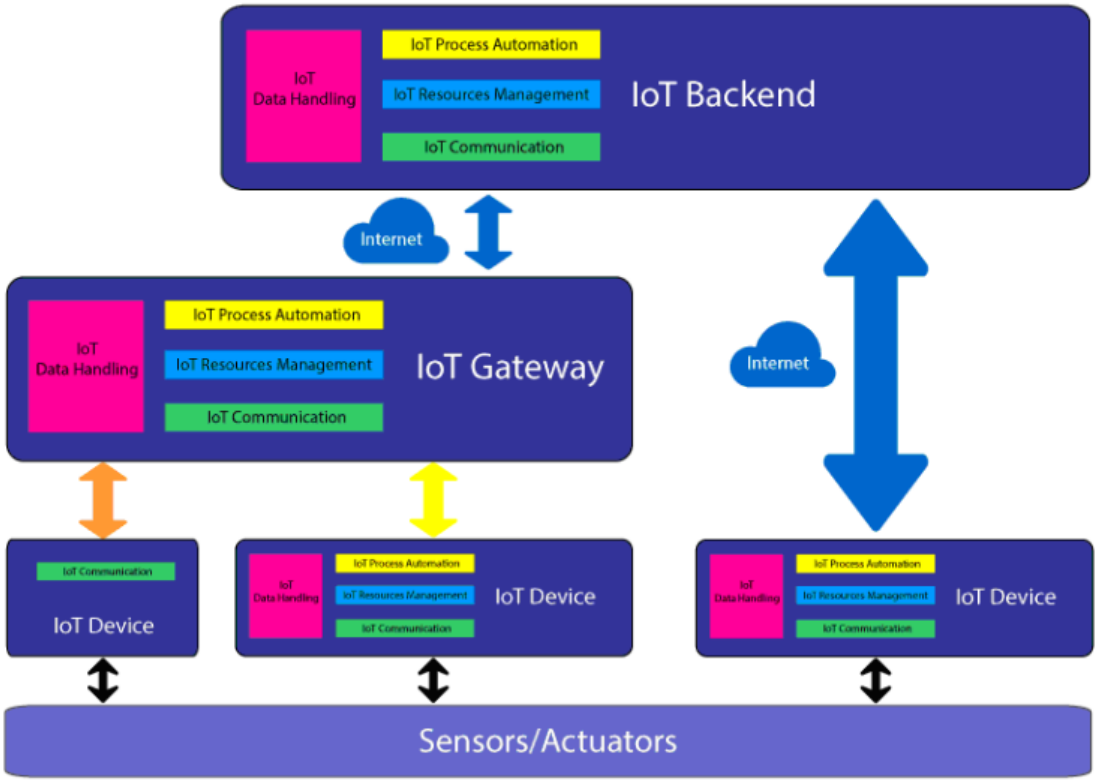
Po získaní dát, prijímací systém ich musí poskytnúť biznis pravidlovému systému. V spomenutých riešeniach môže byť použitá tzv. horúca cesta na analýzu dát ako tok dát v reálnom čase a tzv. studená cesta pre ukladanie dát pre budúce analýzy. Je možné to považovať za Big Data problém. Obidve cesty môžu vystaviť informácie konečnému používateľovi, ktorý môže sledovať, čo sa zo zariadeniami v reálnom svete deje. Rovnaká informácia je užitočná pre systém strojového učenia, ktorý môže pri prediktívnej analýze napomôcť pochopiť ako sa dáta môžu vyvíjať v budúcnosti na základe aktuálnych hodnôt.

Rovnako sa nesmie zabudnúť na opačnú cestu z cloud systému do zariadení. Vo väčšine prípadov na interakciu s nimi je potrebný vzor **príkazy** a **notifikácie**. Pomocou príkazov je možné komunikovať so zariadeniami, takže môžu vykonávať nejaké úkony. Pomocou notifikácií je možné poskytnúť zariadeniam informácie, ktoré potrebujú počas behu. Na zabezpečenie príkazov a notifikácií na komunikáciu s koncovými zariadeniami sa často používajú tzv. brány na komunikáciu zo zariadenia do cloudu a naopak často označovaný ako Cloud gateway.

Všetky zariadenia by boli schopné pristupovať na Cloud gateway, keby boli pripojené pomocou ethernetových sietí s podporou TCP/IP protokolu. Pre zariadenia s obmedzením na prostriedky s využitím PAN (Personal Area Network) protokolov (napríklad Bluetooth, Zigbee, Z-Wave, rovnako je možné uvažovať na AllJoyn frameworkom) je potrebné uvažovať nad tzv. field gateway, ktorý vystupuje ako lokálna brána na prístup do cloud priestoru. Táto brána má úlohu prekladača protokolov a môže zabezpečovať lokálne ukladanie, filtrovanie a spracovanie úkonov, keď prídu dáta, ešte pred tým ako sa pošlú do cloud. Samozrejme to môže byť vstupný bod pre lokálny systém na prijímanie príkazov a notifikácii poslaných z cloud a smerovaných na zariadenia. [16] Cieľom porovnania nie je zaoberať sa detailami implementácie jednotlivých riešení, ale poukázať na principiálnu podobnosť architektúr. Obrázok 29 zovšeobecňuje a znázorňuje všetky súčasti a možnosti zapojenia hlavných súčastí IoT riešení, ktoré sú uvedené v predošlej citácií a rovnako sa uvádzajú vo väčšine literatúr. Finálne zovšeobecnenie jednotlivých komponentov IoT

architektúry je nasledovné.

- Snímač akčný člen, z ktorého komunikačná vrstva robí IoT zariadenie alebo naopak zariadenie s komunikačnou vrstvou a snímačom alebo akčným členom je IoT zariadenie. IoT zariadenie môže mať určitú výpočtovú kapacitu na jednoduchú automatizáciu procesu. Tiež môže mať výpočtovú kapacitu na obsluhu viacerých snímačov akčných členov a dočasné ukladanie dát.
- Ďalší prvok je IoT brána (Gateway, agregátor, field gateway), ktorá spravuje celú sieť IoT zariadení, komunikáciu s nimi, ich správu, pokročilú automatizáciu procesov a dátovú analýzu. Zvyčajne je rozhraním do cloud služieb, ak samotné IoT zariadenie nevie komunikovať priamo so službami v cloude.
- Posledným prvkom je IoT Backend (cloud služby), ktoré sú považované za neobmedzené kapacity priestoru, výkonu, kde môžu prebiehať zložité analýzy dát a zložitá automatizácia procesov.



Obrázok 29: IoT súčasti - všeobecne[26]

V tomto bode máme zadefinované všetky pojmy a princípy, ktoré sú potrebné pri návrhu reálnej IoT aplikácie s modernými metódami riadenia a teda spojenia oblasti automatizácie a softvérovéj architektúry.

2.3.1 Oblasti využitia

Po zadefinovaní hlavných súčasti architektúry IoT riešení, prácav krátkosti popisuje oblasti využitia IoT. Ako podklad pre vymenovanie je dokument [25].

- IT a siete - verejné, podnikové (PC, router, switch, pobočkové ústredne, ...).
- Digitálna a verejná bezpečnosť - verejné osvetlenie, kamerové systémy...
- Obchod - digitálne popisky, registračné pokladnice, ...
- Doprava - lode, lietadla, autá, mýta, ...
- Priemysel - distribučné siete, automatizácia zdrojov, ...
- Zdravotná starostlivosť - telemedicína a sledovanie pacientov, ...
- Energetika - optimalizácia dopytu a ponuky, podpora efektivity alternatívnych zdrojov, ...
- Komerčné budovy - správa budov, úspora na prevádzkovanie, ...
- Domácnosť a spotrebiteľ - pohodlie a zábava, spotrebiče, ...

Pre návrh a implementáciu bol zvolený IoT systém inteligentnej domácnosti, ktorého špecifiká sú v práci priebežne menované. Týmto končí časť definícii a nasleduje popis technických krokov návrhu a vývoja IoT systému a aplikovania MPC do tohto systému.

3 Návrh a implementácia IoT systému s modernými metódami riadenia

Táto časť popisuje postup návrhu a implementácie aplikácie moderných metód riadenia do IoT prostredia. Pod modernými metódami riadenia sa rozumie zavedenie nového pojmu controller as a service (CaaS) - regulátor ako služba a pod IoT prostredím sa tu rozumie inteligentná domácnosť. Myšlienka CaaS je výsledkom sledovania IoT trendu a návrh je inšpirovaný IoT architektúrami v dvoch bodoch. Prvý bod je využitie potenciálu výpočtového výkonu serverov. Vstavané zariadenia sa ukázali ako nedostatočné pre vykonávanie online MPC výpočtu. Druhý bod je užšia spolupráca serveru a akčného člena. Návrh konkrétnej realizácie je založený na SOA architektúre s REST princípmi. Viac je popísane v časti 3.2. Teraz nasleduje popis IoT prostredia s odôvodnením voľby kľúčových elementov s ohľadom na finálnu architektúru.

3.1 Popis a voľba experimentálneho IoT prostredia

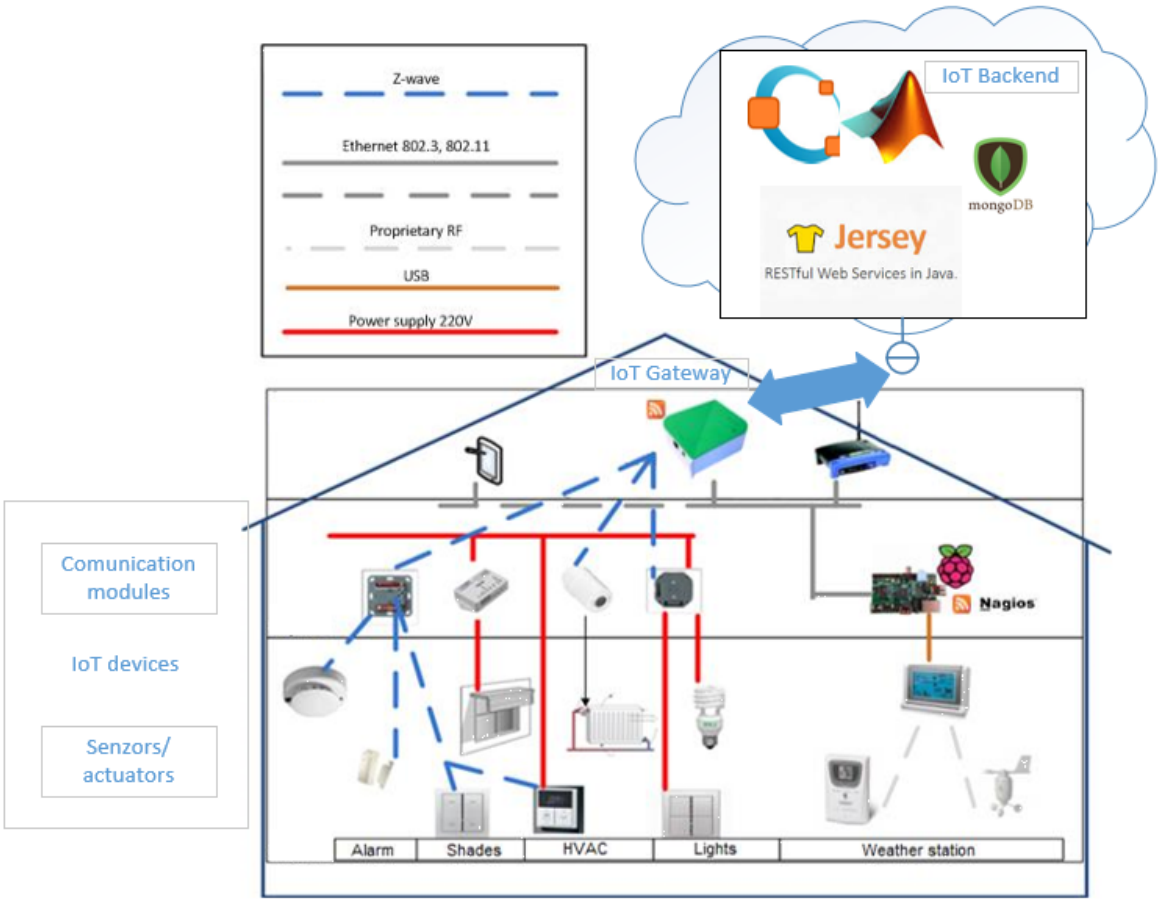
Cieľová architektúra vychádza z obrázkov 26 a 29. IoT prostredie preto pozostáva z týchto prvkov:

- Fyzická vrstva - snímače.
  - Svetelný senzor.
    - Aeotec MultiSensor 4-in-1.
    - Slim Multi-Sensor PSM01.
    - Fotorezistor napojený na AD/DA prevodník s PCF8591 čipom
  - Teplotný senzor.
    - Aeotec MultiSensor 4-in-1.
    - Slim Multi-Sensor PSM01.
    - Oregon Scientific WMR 88.
  - Senzor vlhkosti.
    - Aeotec MultiSensor 4-in-1.
    - Slim Multi-Sensor PSM01.
    - Oregon Scientific WMR 88.
  - Pohybový senzor.
    - Aeotec MultiSensor 4-in-1.

        * Vision ZP3102 EU PIR Motion Sensor.
    - Okenný/dverový senzor na magnetickom princípe.
        * Slim Multi-Sensor PSM01.
        * FIBARO FGK 101-107.
    - Záplavový senzor.
        * FIBARO FGFS 101.
    - Senzor intenzity a smeru vetra.
        * Oregon Scientific WMR 88.
    - Senzor barometrického tlaku.
        * Oregon Scientific WMR 88.
    - Tlačítko.
        * Z-Wave.Me Wall Controller 06443.
        * Z-Wave.Me Wall Controller WallC.
- Fyzická vrstva - akčné členy.
    - Termostatická hlavica.
        * Danfoss living connect Z.
    - Spínacie relé.
        * PAN04 Dual realy.
    - Univerzálny stmievač.
        * FIBARO FGD 211.
        * Aeotec Micro Smart Dimmer.
        * Z-Wave.Me Wall Flush-Mountables.
    - Spínacia zásuvka.
        * FIBARO FGWPE/F 101.
- Fyzická vrstva - komunikačný hardware.
    - USB
    - I^2C
- Fyzická vrstva - Zariadenia.

  - Raspberry Pi model A+
  - Raspberry Pi 2 model B
  - Vera Control VeraLite
- Komunikačná vrstva smer IoT zariadenie gateway.
  - Z-Wave
  - RSS (HTTP)
  - REST (HTTP)
- Komunikačná vrstva smer gateway IoT Backend.
  - REST (HTTP)
- Systémová vrstva.
  - MongoDB databáza
  - Jersey REST interface
  - Matlab/Octave nástroje na výpočty
  - JSON komunikačné rozhranie
  - InfluxDB
  - Grafana
- Používateľská vrstva.
  - Správa inteligentnej domácnosti
  - Vizualizácia nameraných údajov
  - Aplikácie s pridanou hodnotu

Fyzická vrstva sa skladá z vymenovaných senzorov a akčných členov, z ktorých väčšina je pripojená do systému pomocou protokolu Z-Wave. Protokol Z-Wave je bezdrôtový protokol stredného dosahu v Európe fungujúci na frekvenciách 868.42MHz, ktorý potrebuje na svoje fungovanie jeden hlavný uzol, ktorý spravuje sieť a identifikáciu zariadení. Architektonické rozhodnutie, pri návrhu IoT prostredia, pre voľbu Z-Wave protokolu bol fakt, že Z-Wave je bezdrôtový a zariadenia je možné implementovať do existujúcej elektrickej inštalácie. Požiadavka implementácie do existujúcej elektrickej inštalácie vychádza z ekonomického aspektu a z potreby overenia možnosti vytvárania IoT prostredia v byte,



Obrázok 30: Experimentálne prostredie

a teda s obmedzenejšími možnosťami na zásahy do ovládania kúrenia, merania spotrieb a podobne oproti klasickým domom. Spomínaný hlavný uzol v navrhovanom prostredí je zariadenie VeraLite. Existuje viacero alternatív ku tomuto zariadeniu, ktoré majú svoje výhody a nevýhody. Architektonické argumenty pre voľbu tohto zariadenia boli stabilita, možnosť custom vývoja a nižšia cena oproti iným. Zariadenie od spoločnosti Fibaro [9] je vysoko stabilné, ale neumožňuje veci programovať a aj cena je rádovo vyššia. Zariadenie Vera3 [28] je vyššia verzia zariadenia VeraLite, ktoré je ešte stabilnejšie, ale aj pomerne drahšie. Ďalšie uvažované zariadenie bolo RaZberry [10], ktoré sa predáva ako pripojiteľný modul so Z-wave komunikačným rozhraním ku zariadeniu Raspberry Pi, ktoré by zohrávalo úlohu hlavného uzlu. Cena aj možnosť programovania tu boli výborne avšak stabilita nebola dostatočná. Preto hlavný uzol siete Z-Wave zohráva zariadenie VeraLite. Pomocou tohto zariadenia sú jednotlivé fyzické zariadenia pridávané do siete. Dôležitou vlastnosťou je, že okrem fyzických zariadení vie toto zariadenie pridávať aj **zariadenia softvérové**, čo súvisí s možnosťou programovateľnosti. Softvérovým zariadeniam sa bude venovať neskôr. Spôsob pridávania fyzických zariadení do siete Z-Wave funguje na základe špeciálneho módu (inclusion mode), ktorý je potrebné nastaviť na hlavnom uzle. Hlavný uzol vtedy počúva a očakáva sekvenciu príkazov, pomocou ktorých identifikuje nové zariadenie a zaregistruje ho do siete - proces nazývaný inclusion. Pridávané zariadenie rovnako musí byť nastavené do inclusion módu. Spôsob zapnutia inclusion módu je plne na výrobcovi zariadenia väčšinou to je určitá sekvencia stlačenia tlačítka. Pri senzoroch sú to zvyčajne špeciálne tlačítka pre účely nastavovania sieťových parametrov. Pri ostatných zariadeniach sa využívajú používateľské tlačítka. Po zaregistrovaní zariadenia do siete hlavný uzol vie zariadenie ovládať a pýtať sa na aktuálny stav. Zariadenie VeraLite v našom IoT systéme, ako je možné vidieť na obrázku 30 zohráva úlohu brány.

Ďalšie zariadenie pripojené v IoT systéme je meteostanica a to prostredníctvom USB káblu na zariadenie Raspberry Pi model A. Meteostanica pozostáva z vnútornej zobrazovacej jednotky, na ktorej je meraná vnútorná teplota a vlhkosť, vonkajšieho snímača teploty, vlhkosti a atmosferického tlaku, merača úhrnu zrážok a anemometra na meranie intenzity a smeru vetra. Komunikácia medzi snímačmi a vnútornou jednotkou je prostredníctvom rádiového signálu na frekvencii 433 MHz. Na Raspberry Pi beží operačný systém Raspbian, špeciálna verzia distribúcie Debian. V operačnom systéme beží aplikácia *wview*, ktorá komunikuje s USB portom a ukladá údaje do *sqlite* databázy, robí tzv. dočasné úložisko. Síce dlhodobejšie ako je pamäť na zobrazovacej jednotky, ktorá ukladá posledných 24 hodín, ale nie je to úložisko, na ktoré sa možno spoľahnúť.

Ďalší spôsob pripojenia do IoT systému tvorí fotorezistor s PCF8591 čipom cez GPIO

rozhranie pripojené do zariadenia Raspberry pi 2 Model B+. Dôvod voľby tejto kombinácie do IoT systému je možnosť prístupu na najnižšiu úroveň ovládania zariadenia. Pri ostatných Z-Wave zariadeniach je firmware daný a ak je obmedzená minimálna hodnota periódy vzorkovania na hodnotu, ktorá nie je dostatočná pre identifikáciu systému, je potrebné mať možnosť zariadenia, v ktorom dané parametre možno zmeniť. Spôsob vystavania hodnoty fotorezistora do IoT prostredia je vďaka takémuto súboru aplikácií:

- základ tvorí operačný systém typu Linux, špeciálna edícia Debian distribúcie pre Raspberry Pi.
- Nadstavbu tvorí Node.js framework, ktorý umožňuje vytvárať udalosťami riadené systémy.
- V Node.js frameworku je v jazyku Javascript vytvorená aplikácia, ktorá
  - umožňuje komunikovať s GPIO rozhraním a tak načítať hodnotu zo senzora.
  - Prekladá údaj o odpore zo svetlocitlivého fotorezistora na údaj zodpovedajúcemu intenzite osvetlenia a
  - poskytuje nameranú hodnotu prostredníctvom REST rozhrania.

Fyzikálna veličina odporu je takto skonvertovaná do digitálnej podoby a takto prístupná takému zariadeniu, ktoré vie vytvoriť HTTP požiadavku a prečítať jej odpoveď.

3.1.1 Detaily IoT brány

Táto kapitola sa detailne venuje

- popisu VereLite zariadenia.
- Spôsobu komunikácie zariadení Raspberry Pi s VeraLite (IoT bránou).
- Spôsobu programovania zariadenia VeraLite a vytvárania softvérových zariadení v ňom, ktoré sú demonštráciou úzkeho prepojenia IT s OT. Ďalej sa prostredníctvom nich demonštrujú výhody SOA architektúry a REST princípov.

Výrobcovia zariadení rodiny Vera si ako základný prvok postavili Linux systém. Na VeraLite beži distribúcia OpenWrt. Framework na ovládanie IoT zariadení je postavený na protokole UPnP - priemyselný štandard na ovládanie zariadení, a skriptovacom jazyku Lua. Framework sa volá *Luup* [21].

Vďaka UPnP je možné definovať zariadenia, jeho služby, stavové premenné služby,

operácie služby a parametre na vstupe a výstupe operácie. Ako príklad definície zariadenia uvádzame spínané svetlo a stmievané svetlo.

ID definície spínaného zariadenia je *urn:schemas-upnp-org:device:BinaryLight:1*. Spínané svetlo má len jednu službu a jej ID je *urn:upnp-org:serviceId:SwitchPower1*. Jediná premenná tejto služy je *Status*, ktorá môže nadobúdať hodnoty 0, ak je svetlo vypnuté a 1, ak je svetlo zapnuté. Jedina operácia tejto služby *SetTarget*, ktorá má na vstupe parameter s názvom *newTargetValue*. Ukážka zapnutia svetla je v algoritme 1.

Algoritmus 1 Zapnutie svetla

```
1 luup.call_action("urn:upnp-org:serviceId:SwitchPower1",
2                  "SetTarget", {newTargetValue = "1"}, 37)
```

Funkcia call_action frameworku *Luup* robí zmenu nastavenia UPnP premennej prostredníctvom funkcie, ktorá bola pri špecifikácii služby zariadenia na to určená. Parametre potrebné na túto zmenu sú meno služby, meno funkcie, meno vstupnej premennej, jej hodnota a nakoniec ID inštancie zariadenia v systéme. ID inštancie je vytvárané tak pre fyzické zariadenia, ako aj pre softvérové zariadenia.

Stmievané svetlo má ID definície *urn:schemas-upnp-org:device:DimmableLight:1*. Má dve služby. Má aj službu *urn:upnp-org:serviceId:SwitchPower1*, ktorú má spínané svetlo, keďže aj stmievané svetlo je možne vypnúť ako spínané. Navyše má službu *urn:upnp-org:serviceId:Dimming1*, ktorá pomocou premennej operácie *SetLoadLevelTarget* a vstupnej premennej *newLoadlevelTarget* nastavuje stavovú premennú *LoadLevelStatus*. Čím sa nastavuje percento intenzity zotmenia. Kód na zistenie stavu stmievaného svetla vo frameworku *Luup* je zobrazený v algoritme 2. Na základe tejto generickosti UPnP pro-

Algoritmus 2 Načítanie stavu stmievaného svetla

```
1 lightLevel = luup.variable_get("urn:upnp-org:serviceId:Dimming1",
2                                "LoadLevelStatus", 37)
```

tokolu sa v práci definovalo šesť nových typov zariadení, štyri pomocné a dve zariadenia nevyhnutné pri implementácii CaaS. Začneme s popisom jednoduchších, pomocných a následne prejdeme k tým dôležitým.

- Zariadenie typu barometer s ID *urn:schemas-micasaverde-com:device:Barometer:1* a službou *urn:upnp-org:serviceId:Barometer1*, ktorá bola vytvorená na základe zari-

adenia pre snímanie teploty so zmenou mena stavovej premennej na *CurrentAir-Pressure*

- Zariadenie typu veterný snímač s ID *urn:schemas-micasaverde-com:device:WindSensor:1* a službou *urn:upnp-org:serviceId:WindSensor1*. Toto zariadenie pozná stavové premenné *WindSpeed* a *WindDirection*. Obe vytvorené zariadenia, nemajú operáciu na zadávanie hodnoty, keďže sú to senzory. Používateľ pri nich nemá a ani nemá mať možnosť zadávať hodnotu, má iba možnosť hodnoty zo senzora čítať.

- Na komunikáciu VeraLite s Raspberry PI model A a čítanie hodnôt z meteostanice sa vo VeraLite vytvorilo všeobecné softvérové zariadenie typu RSS Reader s ID zariadenia *urn:demo-micasaverde-com:device:RssReader:1*.

- Na komunikáciu VeraLite s Raspberry PI model B a čítanie hodnôt z fotorezistora sme vytvorili všeobecné zariadenie Rest Handler, všeobecne na čítanie akejkoľvek rest služby s ID *urn:demo-micasaverde-com:device:RestHandler:1*.

To boli pomocné softvérové zariadenia, ktoré rozširujú možnosti komunikácie a dátové body zariadenia VeraLite a pripravujú IoT prostredie na aplikovanie prediktívnych metód riadenia do systému. Nasleduje krátky opis RSS Reader a REST Handler zariadení a následne bude kapitola venovaná aplikácii MPC v IoT systéme.

RSS Reader predstavuje zariadenie, ktoré pravidelne odoberá novinky z webových stránok vo formáte XML. Vďaka tomu, okrem toho, že je možné sa napojiť na odber akýchkoľvek noviniek z bežných webových stránok, je možné sa napojiť aj na aplikáciu *wview*, ktorá v popisovanom systéme beží na Raspberry Pi model A. Do tela noviniek generovaných aplikáciou *wview* sa nakonfigurovalo, aby sa tam vkladalo XML v špecifickom tvare, v ktorom sú uložené dáta zo senzorov meteostanice. Ukážka časti konfiguračného súboru je v algoritme 3.

Takto sa zabezpečí, že v odbere noviniek zo „serveru“ s meteostanicou je pravidelne podávaná informácia o senzorických dátach, v prípade práce to je konkrétne vnútorná a vonkajšia teplota, vnútorná a vonkajšia vlhkosť, intenzita a smer vetra a barometrický tlak. V grafickom rozhraní na nastavenie aplikácie *wview* sa volí ako často sa majú dáta publikovať. Týmto je vybavená strana poskytovateľa údajov. Pokračuje sa v opise RSS Reader softvérového zariadenia naprogramovaného v *Luup* frameworku. Základné súčasti programu naprogramované v jazyku Lua, ktoré možno jednoducho pridať do *Luup* frameworku, sú modul *socket.http*, ktorý slúži na poslanie HTTP GET požiadavky na „server“ s meteostanicou alebo url na odoberanie noviniek. Ďalej to je XML parser, ktorý

Algoritmus 3 Časť konfiguračného súboru na vytvorenie XML k odberu noviniek

```
...
<item>
  <title><!--stationCity-->,<!--stationState--> </title>
      <link>Insert_Your_WX_HomePage_URL_Here</link>
      <description>Current Weather Conditions</description>
      <pubDate><!--stationTime-->, <!--stationDate--></pubDate>
      <dc:date><!--stationDate--> - <!--stationTime--></dc:date>
          <!-- vera specific content -->
          <temperature>
            <sensor id="0" attr="inside" descr="" >
                  <value><![CDATA[<!--insideTemp-->]]></value>
                  <unit><![CDATA[<!--tempUnit-->]]></unit>
                </sensor>
            <sensor id="1" attr="outside" descr="" >
                  <value><![CDATA[<!--outsideTemp-->]]></value>
                  <unit><![CDATA[<!--tempUnit-->]]></unit>
                </sensor>
          </temperature>
  </title>
</item>
...
```

bol stiahnutý ako open-source zdrojový kód a použitý na čítanie jednotlivých položiek v novinkách a na čítanie definovaného tvaru XML, ktoré chodí z meteostanice. V programe je definovaný prepínač, ktorý umožní, aby na základe prečítaných dát z meteostanice sa vytvorili samostatné „podzariadenia“ vytvárane RSS Reader zariadením, ktoré sú typu teplotného senzoru, senzoru vlhkosti a všetky ostatné, ktoré boli vyššie vymenované. V grafickom rozhraní pre používateľa to vyzerá tak, ako je znázornené na obrázku 31. Na

Obrázok 31: Vera GUI - vytvorené softvérové zariadenie

obrázku je zariadenie s popisom *Rss reading device* hlavné zariadenie, ktoré vytvorilo všetky „podzariadenia“ s prefixom Oregon. Výhoda tohto prístupu je, že v automatizačných skriptoch sa možno rovno dopytovať na údaj z meteostanice a vyzerá to akoby teplota bola snímaná Z-Wave zariadením, keďže VeraLite je primárne hlavný uzol Z-Wave zbernice. Súčasti, ktoré je potrebné pri tvorbe softvérového zariadenia vytvoriť:

- Definíciu zariadenia pre RSS Reader, čo je *D_RssRead.xml*. XML súbor kde je definované ID definície a služby, z ktorých pozostáva. V tomto prípade služba *urn:demo-micasaverde-com:serviceId:RssRead1*.
- Ďalej je potrebné vytvoriť definíciu služby v tomto prípade ID služby z predošlého bodu je definované v súbore *S_RssRead.xml*. Tu sa nachádzajú stavové premenné a operácie. Najdôležitejšie stavové premenné sú URL a prihlasovacie údaje v prípade zabezpečenej URL. Riešený je len prípad zabezpečenia Basic authentication technológiou. Potom je tam niekoľko pomocných premenných a následne tieto operácie:
  - SetURL - nastavenie URL adresy servera.
  - SetUser - nastavenie prihlasovacieho mena.

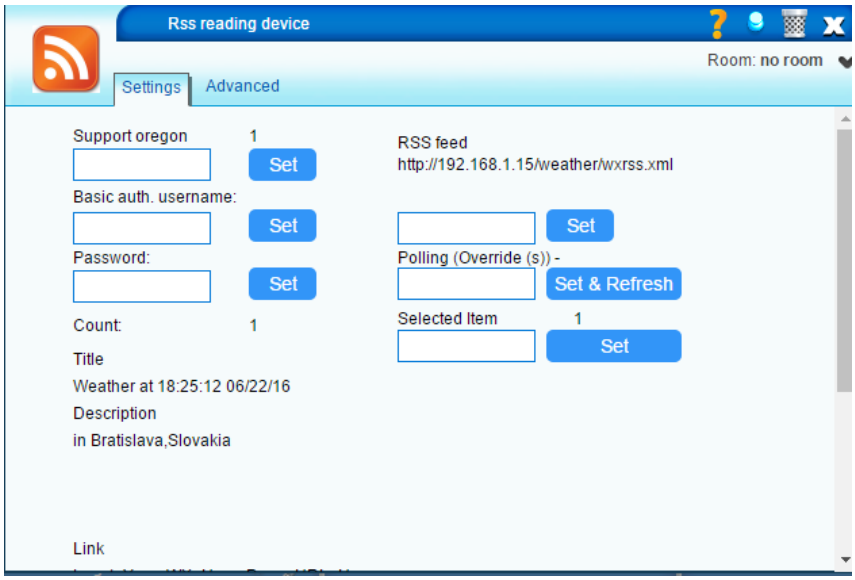
  - SetPass - nastavovanie hesla.
  - SetSelected - nastavovanie zvolenej položky v prípade, ak server vráti viacero položiek v odbere noviniek. Nemôže sa stať v prípade odberu noviniek z meteostanice, lebo XML je definované tak, že tam je len jedna položka.
  - SetRefPeriod - nastavenie času ako často sa má RSS Reader spýtať servera na aktuálne hodnoty. Tu si treba všimnúť, že je uplatnený princíp pravidelného pýtania, namiesto asynchronného prístupu, ktorý by bol lepší, ale náročnejší na zdroje aj na strane klienta ja na strane servera a navyše to *Luup* framework nepodporuje.
  - GetFeed - akcia na vyžiadanie si najnovších noviniek kliknutím. Bez zásahu používateľa je táto akcia pravidelne volaná podľa periódy nastavovanej v predošlej funkcii.

- Ak je zadefinované zariadenie, služba, operácie a stavové premenné, tak je potrebné ich implementovať v špecifickom súbore. Pre RSS Reader to je *I_RssRead.xml*. V tomto súbore je implementácia jednotlivých operácii. Keďže to je XML súbor, môže sa v tom komplikovane vyvíjať komplexná funkcionalita, preto je možné komplexnú logiku v Lua skripte programovať do špeciálneho súboru pre RSS Reader to je *L_RssRead.lua*.

- Nakoniec Vera rozhranie s *Luup* frameworkom umožňuje definovať vzhľad ako sa bude na GUI softvérové zariadenie zobrazovať. Toto je konfigurované v JSON súbore (*D_RssRead.json*). Je to pomerne obmedzený a ťažkopádny vývoj rozhrania, ale je to daň za konfiguračnú unifikovanosť. Vzhľad RSS Reader zariadenia je zobrazný na obrázku 31 a nastavenia RSS Reader zariadenia na obrázku 32.

Potom ako je všetko zadefinované, Vera poskytuje vývojárske rozhranie, kde sa spomínané súbory vložia a tým Vera pozná nové zariadenia, ktoré ma definované v XML súboroch, ktorých štruktúra je daná UPnP protokolom. Inštancia zariadenia však stále nie je vytvorená. Na to existuje ďalšie rozhranie, kde sa definuje ID definície zariadenia a jeho meno. Následne Vera vytvorí inštanciu zariadenia a priradí mu jednoznačné ID inštancie, podľa ktorého je možné sa dopytovať na stavové premenné alebo vykonávať operácie podľa algoritmov 1 a 2.

Rest Handler softvérové zariadenie funguje na veľmi podobnom princípe ako RSS Reader. Má len jednu službu s menom HandleRest, ktorej ID je *(urn:demo-micasaverde-com:serviceId:HandleRest1)* - spracuj REST službu, čo znamená, že program pravidelne



Obrázok 32: Nastavenia RSS Reader zariadenia

oslovuje vystavený REST interface, ktorý poskytne dáta, v našom prípade informáciu z fotorezistora. Ohľadne implementácie, opäť je potrebné použiť modul na vytvorenie HTTP klienta a v tomto prípade je formát správ v JSON notácii, takže miesto XML parsera je tu použitá knižnica [19] na konverziu JSON objektov. Základne operácie definovane v službe sú nasledovné.

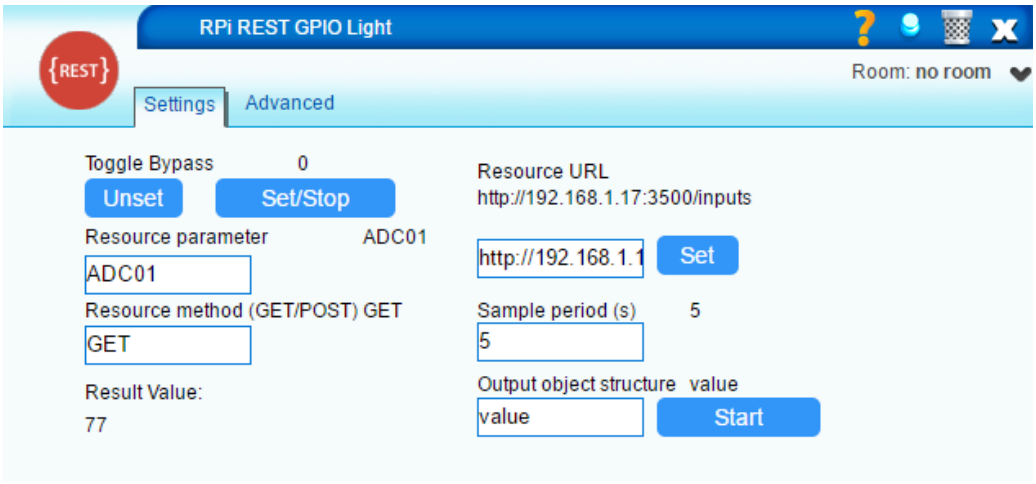
- setRestUrl - operácia na zadefinovanie URL, na ktorú sa ma HTTP klient dopytovať.
- handleRest - operácia na samotné spustenie programu na pravidelné načítavanie hodnôt. Je potrebné definovať REST zdroj, HTTP metódu, ktorá sa ma použiť, perióda s akou ma byť REST zdroj volaný a objekt, z ktorého sa ma čítať výstupná hodnota.
- (un)setBypass - operácie na zastavenie načítavania hodnôt.

Rozhranie na zadanie hodnôt a spustenie je zobrazené na obrázku 33. Výstup, ktorý REST rozhranie poskytne má tvar naznačený v algoritme 4. Z odpovede je možné vidieť,

Algoritmus 4 Príklad odpovede zo snímača vystaveného REST princípom.

```
1 {
2     "pin":0,"gpio":"ADC01","AIN":1,"value":75
3 }
```

že takto by bolo možné osloviť ktorýkoľvek pin GPIO rozhrania Raspberry PI a tak s



Obrázok 33: Rozhranie na zadávanie parametrov REST klienta.

pomocou jedného zariadenia poskytovať rozhranie ku celej množine senzorov a akčných členov. V našom prípade je pripojený len fotorezistor.

Toto bola ukážka ako programovať do špecifickej IoT brány (VeraLite), ktorá okrem obsluhy Z-Wave zariadení umožňuje ovládať aj zariadenia komunikujúce s TCP protokolom a jeho nadstavbami (HTTP, RSS, REST...). Tvorcovia Luup framewroku si zvolili mikrokernel (plugin) architektúru, kde hlavné jadro je implementované a kto čo potrebuje si môže v podobe pluginu naprogramovať a pridať do mikrokernel prostredia. Pre nás táto voľba je prínosná, pretože takýmto spôsobom je možné pripojiť akúkoľvek servisnú aplikáciu do systému a teda môžeme vidieť výhody správne zvolenej architektúry softvéru.

Ďalej si tu môžeme uvedomiť otvorenosť UPnP protokolu a možnosti definovania vlastných zariadení pomocou neho. Čo v sumáre predstavuje štandardom podporenú mikrokernel architektúru, keďže UPnP je definované v ISO/IEC 29341 štandarde. Všetky doteraz menované aj neskôr spomenuté softvérové zariadenia, vytvorené v práci, sú dostupné pre využívanie používateľmi Vera zariadenia prostredníctvom github platformy. Nasleduje najdôležitejšia časť práce.

3.2 CaaS - Regulátor ako služba

Táto kapitola sa zameriava na vysvetlenie, motiváciu a popis implementácie myšlienky regulátor ako služba (Controller as a Service). Ako už bolo naznačené ide o vystavovanie služby regulovania procesu do cloud prostredia. Kedy na servery je uložený matematický model regulátora a na požiadavku vie server vrátiť akčný zásah pre daný regulátor v danom stave. Od klienta sa očakáva, že vie obsluhovať HTTP komunikáciu a základnú

prácu s reťazcami, žiadne výpočty nevykonáva, len sa pravidelne spýta na akčný zásah. Pre regulátory s nízkymi nárokmi na výpočtovú kapacitu tento koncept nepridáva hodnotu výpočtového výkonu serveru, iba možnosť uloženia veľa inštancií, takže sa môže využiť iba pamäťová kapacita servera. Tento princíp dáva zmysel hlavne pre regulátory s veľkými požiadavkami na výpočtový výkon. Medzi takéto patrí práve online metóda MPC algoritmu, ktorý si vyžaduje, ako je v prvých kapitolách naznačené, prácu s veľkými maticami ich násobenie a inverzia. Až niekoľko neúspešných pokusov v postupe práce priviedli k tomuto konceptu. Prvotné úvahy boli implementovať algoritmus popísaný v kapitole 1.2 do jazyka, ktorý je blízky strojovému, aby bolo možné dať regulátor čo najbližšie procesu a teda ku zariadeniu, ktoré nemá veľký výpočtový výkon. Na toto boli použité konverzné metódy Matlab skriptov na spustiteľné súbory v jazyku C. Tieto pokusy neboli úspešné. Paralelne prebiehali aktivity implementácie MPC algoritmu na FPGA zariadenia, kde sa rovnako narážal na problem inverzie matíc a ďalších problémov, ktoré kvadratické programovanie prináša. Preto bol pre FPGA implementovaný explicitný - offline MPC regulátor. Tejto problematike sa venuje iná dizertačná práca. Predchádzajúce fakty z oblasti softvérovej architektúry a štúdium IoT architektúr priviedli na myšlienku CaaS a teda proces riadený pomocou regulátora, uloženého a výpočty vykonávajúceho na vzdialenom servery. Dôvody, prečo bol volený implicitný - online MPC regulátor sú popísané na konci kapitoly 1.1.4. Stručné opakovanie toho je, že v práci sa kládol dôraz, aby sa matematický model mohol pravidelne „synchronizovať“ s výstupmi systému a teda sa mohol model meniť v prípade potreby aj v každom kroku. Implementovaná aplikácia služby regulátora je pripravená na to, aby sa matematický model menil aj v každom kroku. Služba na online identifikáciu pre experiment nebola pripravená, kvôli svojej komplexnosti. Najskôr je potrebné overiť základnú myšlienku CaaS a ak sa tá ukáže ako zmysluplná, tak sa v budúcnosti implementuje aj služba online identifikácie, ktorá má rovnaké miesto v prostredí s vysokými nárokmi na výpočtovým výkon a pamäť. Touto kombináciou by prediktívne riadenie bolo doplnené o adaptívnu zložku. Zhrnutie aktuálneho princípu CaaS. CaaS poskytuje rozhranie na zadefinovanie regulátora a rozhranie na spýtanie sa na hodnotu akčného zásahu. Na vstup pri požiadavke na akčný zásah sa zadáva nameraná hodnota výstupu, aby bola zabezpečená spätná väzba. Ďalej na vstup ide žiadaná hodnota, teda referenčný signál, ktorý ma výstupná veličina sledovať.

3.2.1 Implementácia služby regulátora

Po vysvetlení myšlienky a jej motivácie nasleduje popis implementácie. Štúdiom architektúr a architektonických princípov bol zvolený typ servisnej aplikácie (service application) bez grafického rozhrania na základe mikroservice architektúry. Je to ľahko

oddeliteľná funkcionalitu s potrebou škálovateľnosti, preto je microservice architektúra alebo aj SOA najvhodnejší kandidát. Na obrázku 30 v časti nad oblakom sú znázornené základne technológie, na ktorých je aplikácia postavená.

- V prvom rade sú to nástroje Matlab a Octave. V prvej časti štúdia bol čas venovaný vyladeniu MPC algoritmu v prostredí Matlab pre SISO aj MIMO systémy, s obmedzeniami na vstupné a výstupné veličiny a ich zmeny atď. funkcionalita popísaná v kapitole 1.2. Toto sa zobralo ako základ pre optimálnu prácu s maticami a pripravil sa interface na vystavenie tejto funkcionality REST architektonickým štýlom. Pri prvých pokusoch o vytvorenie C programu, blízko akčného člena, ktorý sa testoval na Raspberry Pi zariadení, Matlab nebolo možné nainštalovať, preto sa zobrala jeho open-source verzia Octave. Pri pokusoch došlo ku optimalizácii algoritmu, aby fungoval aj pre Octave, na čo bolo potrebné niekoľko minoritných zmien. Nakoniec sa Octave dostal aj ako možnosť na serverovej strane. Takže pri nasadzovaní aplikácie CaaS je možné si zvoliť medzi Matlab a Octave, ako výpočtové jadro a nevyhnutnú súčasť CaaS aplikácie.

- Ďalšia súčasť aplikácie je dokumentová NoSQL databáza MongoDB, ktorá zo sebou prinášala výhodu, že nie je potrebné definovať tabuľky a ich stĺpce ako je to pri relačných databázach, ale jej úlohou je ukladať matematický model a sprievodné údaje podľa toho, ktoré sú pre aký regulátor potrebné. Ukladanie je vo forme JSON dokumentov.

- Posledná súčasť je framework pre jazyk Java na vystavenie funkcionality prostredníctvom REST rozhrania s názvom Jersey. Táto časť tvorí prístupový bod pre klientov, ktorí na vstupe najskôr definujú matematický model a potom si v každom kroku vypýtajú akčný zásah, ktorý im služba vypočíta. Tento nástroj robí typovú striktnosť medzi vstupom od klienta a objektom, na ktorý sa transformuje. Inými slovami robí preklad z JSON notácie na jednotlivé Java triedy. V tejto časti je integrované volanie dvoch predošlých súčastí. Rozhrania, ktoré aplikácia poskytuje pre klientov sú:

  - POST c-a-a-s/rest/mpccontrollers
    Toto rozhranie slúži na zadanie regulátora do aplikácie. Príklad tela správy, ktorá je potrebné poslať na server je v algoritme 5. Možnosti zadania systému sú viaceré. Či už prostredníctvom stavového modelu alebo prenosovej funkcie zadanej reťazcom alebo vektormi čitateľa a menovateľa. Ostatné parametre ako

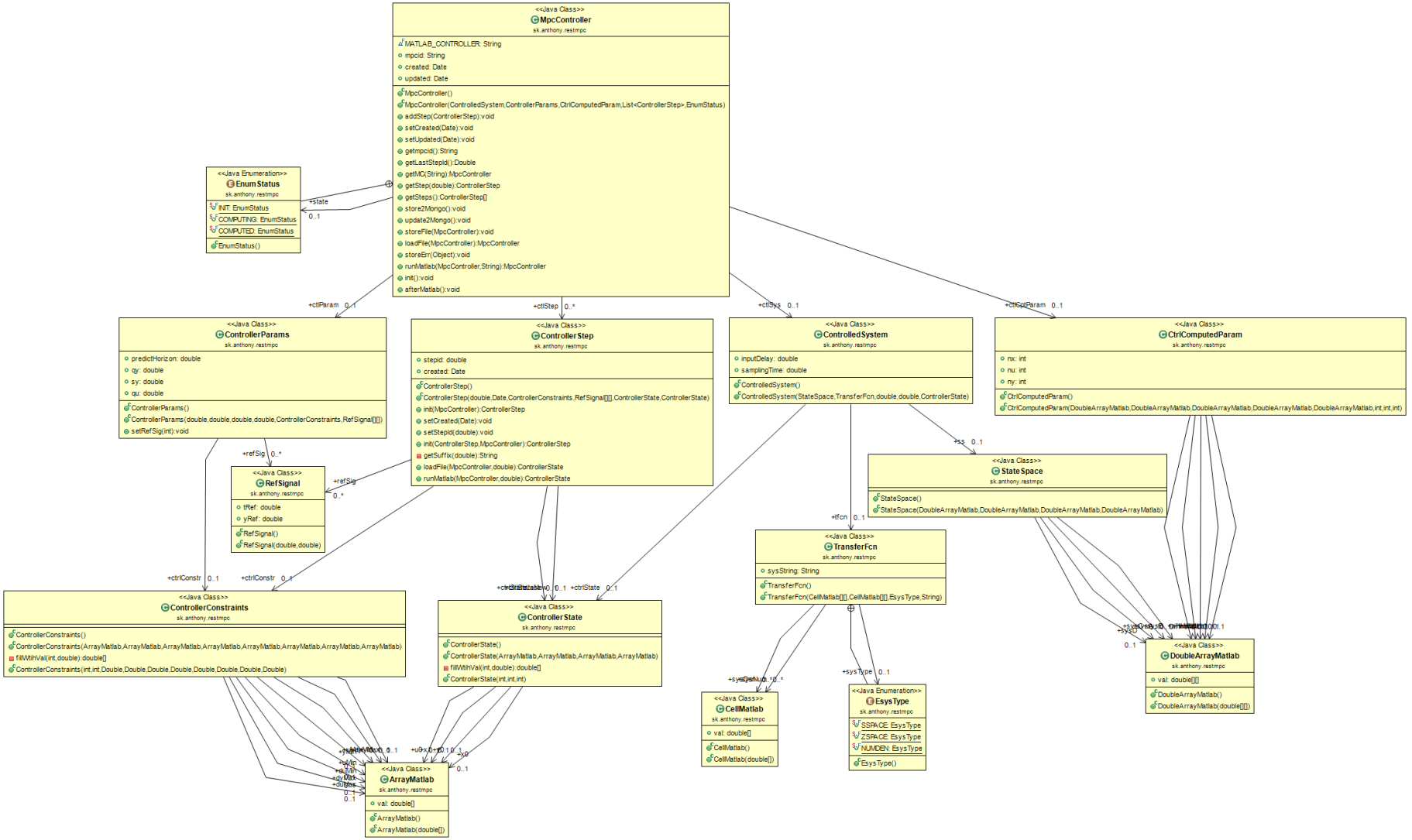
Algoritmus 5 Príklad tela HTTP požiadavky na definovanie MPC

```
{
    "CtlSys": {
        "Ss" : {
            "SysA" : { "Val" : [ [-1, 0], [1, 0] ] },
            "SysB" : { "Val" : [ [1], [0] ] },
            "SysC" : { "Val" : [ [0, 1] ] },
            "SysD" : { "Val" : [ [0]] }
        }
    },
    "CtlParam": {
        "PredictHorizon": 10
    }
}
```

obmedzenia systému, horizont predikcie atď. sú nepovinné. Ak sa nezadajú vygenerujú sa pre nich default hodnoty. Výstupom tohto volania je vygenerované ID regulátora.

- GET c-a-a-s/rest/mpccontrollers/{id}
  Ak už klient pozná ID regulátora, tak pomocou tohto rozhrania je možné si pozrieť detail uloženého objektu. Na výstupe je JSON dokument uložený v databáze, ktorý reprezentuje všetky údaje o regulátore.
- POST c-a-a-s/rest/mpccontrollers/{id}/steps Pomocou tohto rozhrania vie klient požiadať o ďalší riadiaci zásah. Zadanie periódy vzorkovania a jej dodržiavanie v zmysle pravidelného volania je plne na zodpovednosti klienta. Výstupom je ID kroku, ktoré je inkrementálne na rozdiel od ID regulátora a vo výstupnom JSON objekte je aj riadiaci zásah, ktorý je potrebné nastaviť akčnému členu.
- GET c-a-a-s/rest/mpccontrollers/{id}/steps/{id} Ak je potrebné pozrieť kroky v minulosti, tak je to možné pomocou tohto rozhrania.

Diagram tried teda parametrov a ich typov, s ktorými aplikáciu uvažuje a je možné ich zadať do aplikácie, či pre regulátor alebo krok riadenia, je na obrázku 34.



Obrázok 34: Diagram tried vstupného rozhrania.

Funkcionalita, ktorú musí rozhranie na servery vykonať pri vytváraní inštancie regulátora a rovnako aj pri výpočte akčného zásahu pozostáva z týchto krokov:

1. Inicializácia nenastavených default premenných.
2. Uloženie do Mongo databázy, ktorá vygeneruje ID záznamu, ktoré figuruje ako ID regulátora.
3. Pred výpočtom sa uloží JSON do súboru.
4. Nasleduje volanie výpočtového systému Matlab alebo Octave, ktorý načíta uložený súbor, pomocou pluginu Jsonlab [20], ktorý vie z JSON objektu spraviť Matlab/Octave dátové typy a naopak.
5. Nasleduje krok v Matlab/Octave nástroji, kde prebehne výpočet podľa kapitoly 1.2, či už inicializácia systému alebo výpočet akčného zásahu. Na konci výpočtu sa dátové typy uložia do súboru pomocou Jsonlab vo forme JSON objektu.
6. Späť v Java aplikácii sa súbor načíta a spraví z neho inštancia príslušnej triedy.
7. Upravený objekt sa uloží do databázy.
8. Ten istý objekt sa pošle používateľovi na výstup.

Pri implementácii boli komplikácie s načítavaním objektu zo súboru, ktorý ukladal Jsonlab. Chýbala typová striktnosť, takže sa stalo, že z premennej typu vektor v Java aplikácii sa v Matlab/Octave stalo obyčajné číslo, ak vektor pozostával z jedného prvku. Následne pri ukladaní výsledkov výpočtov do súboru sa vynechala vektorová notácia a keď typovo striktný Java interface vytváral objekt, ktorý poskytne klientovi, tak vektorová notácia bola očakávaná. Na vyriešenie týchto komplikácii, boli potrebné úpravy do jadra Java objektov poskytujúcich transformáciu textu na JSON objekty.

Ďalší problém bol s volaním Matlab z inej aplikácie, kedy nebolo možné spustiť Matlab pod používateľom, ktorý je prihlásený ako vlastník procesu aplikačného serveru, pomocou ktorého je CaaS aplikácia spustená. Preto bol potrebný Python Interface, ktorým sa Matlab podarilo spustiť. Veľkú časť výpočtového času zaberá spustenie aplikácie Matlab. Preto bola vyskúšaná verzia spustenia Octave na Linux prostredí a čas výpočtu sa skrátil z v priemere 14 sekúnd na približne 4 sekundy. Pritom zariadenie s Linuxom bolo oveľa menej výkonné. Porovnanie parametrov v tabuľke 2. Tento fakt naznačuje preferenciu používania Octave verzie. 4 sekundy sa stále môžu zdať veľa, preto treba upozorniť, že ak by požiadavky na kvalitu regulácie boli striktnejšie je potrebné aplikáciu CaaS dať na

Tabuľka 2: Porovnanie testovaných prostredí.

Procesor typ:	Intel Core i5-2520M	Intel Celeron M
Procesor frekvencia	2.5GHz	1.46GHz
Operačná pamäť:	8GB	2GB
Disk	SSD	HDD
Operačný systém	Windows 8.1	Debian 8
Aplikácia	Matlab 2015	Octave 3.6.2

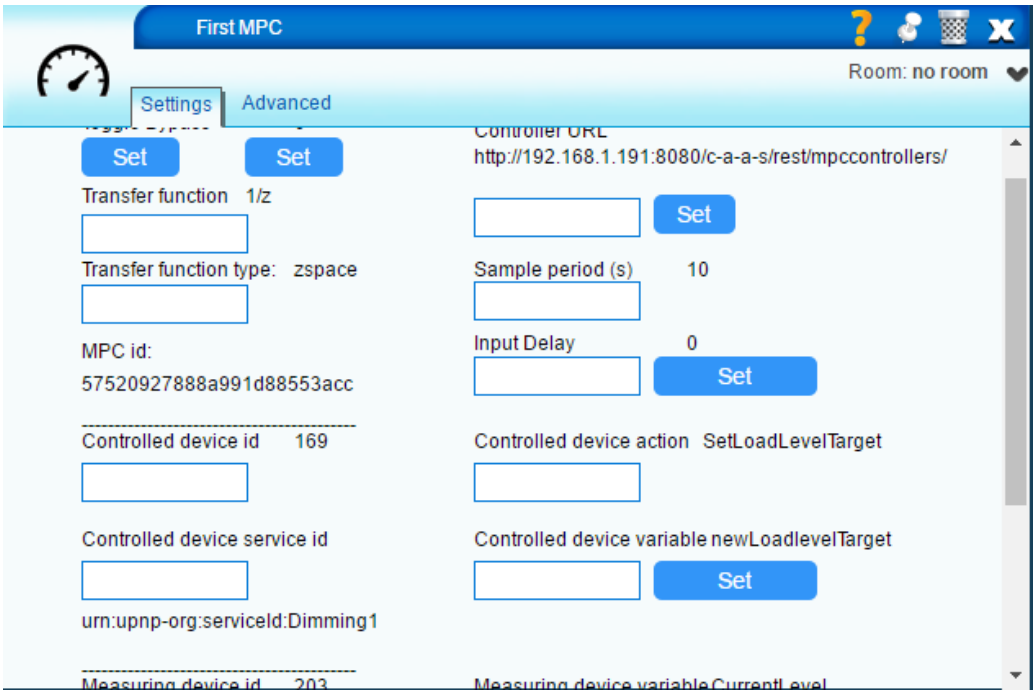
výkonnejší stroj. Pri načítaní regulátora z databázy je výsledok okolo 400 ms na slabšom zo spomínaných prostrediach. Preto pri riadení je možné očakávať podobné časy odpovede a teda na výkonných cloud zariadeniach je možné riadiť procesy s periódou vzorkovania rádovo desiatky milisekúnd. Spôsob garancie odpovede do určitého času a núdzový režim v prípade sieťového výpadku nie je v práci riešený. Dnes však poskytovatelia cloud služieb a poskytovatelia internetu vedia garantovať 24/7 dostupnosť a pripojenie.

3.2.2 Integrácia MPC pomocou softvérového zariadenia

Pre službu regulátora popísanú v predošlej časti a vystavenú v časti IoT Backend - obrázok 30 je potrebné vytvoriť klienta. V rámci práce je vytvorený klient v prostredí popísanom v kapitole 3.1. Postup vytvorenia klienta aj súčasti potrebné na implementáciu sú podobné ako pri RSS Reader alebo REST Handler softvérových zariadení. Princíp ostáva rovnaký. Poskladať a zavolať HTTP požiadavku a spracovať odpoveď vo forme JSON správy, najskôr pre regulátor a potom cyklicky pre jednotlivé kroky, všetko v Lua jazyku vytvorené pre *Luup* framework.

Softvérové zariadenie vytvorené v práci je definované ako MpcClient s ID *urn:demo-micasaverde-com:device:MpcClient:1* so službou, ktorá ma ID definované ako *urn:demo-micasaverde-com:serviceId:MpcControl1*. Stavových premenných je tu viac ako v prípade RSS Reader zariadenia. Venovať sa budeme iba najdôležitejším. Spôsob vytvorenia inštancie zariadenia je rovnaký ako pre RSS Reader. Výhodou tohto prístupu je, že je možné vytvoriť viacero inštancií a riadiť nimi ľubovoľný počet akčných členov. Rozhranie na zadávanie parametrov a spustenie riadenia je na obrázku 35. Nachádza sa tam

- vstup pre prenosovú funkciu.
- Vstup pre zadanie URL služby, kde je aplikácia CaaS vystavená.
- Zobrazenie ID regulátora.



Obrázok 35: Rozhranie na zadávanie parametrov MPC regulátora.

- Tlačítko na vypnutie/povolenie riadenia.
- Vstup pre zadanie periódy vzorkovania, ktorá zároveň identifikuje ako často sa bude volať služba na načítanie akčného zásahu.
- Vstupy na definovanie zariadenia, ktoré bude regulátorom ovládané.
  - ID inštancie zariadenia.
  - ID služby.
  - ID operácie.
  - Meno vstupnej premennej do operácie.
- Vstupy na definovanie zariadenia, ktoré bude slúžiť na meranie výstupnej veličiny
  - ID inštancie zariadenia.
  - ID služby.
  - Meno meranej stavovej premennej.

Princíp fungovania je taký, že na začiatku sa inicializuje regulátor zadaním prenosovej funkcie, snímacieho zariadenia, vykonávacieho zariadenia a periódy vzorkovania. Regulátor sa uloží do IoT Backend a na klientovi sa vždy na konci výpočtu načasuje spustenie

nového výpočtu, ktorý sa spustí v čase periódy vzorkovania. Takto sa pravidelne zavolá IoT Backend, aby vrátil vypočítaný akčný zásah. Na vstupe je vždy nameraná hodnota zo snímacieho zariadenia, aby sa zabezpečila spätná väzba. Akčný zásah na výstupe je aplikovaný do akčného členu.

Pri identifikácii systému, ktorá je popísaná neskôr, je identifikovaný nelineárny systém. Tento fakt spôsobuje, že na jeden proces sú potrebné viaceré regulátory pre každý pracovný bod lineárnej časti jeden. Na to aby bolo možné prepínať, ktorý regulátor ma riadiť danú časť priebehu je v práci vytvorené posledné šieste softvérové zariadenie s názvom MpcMaster. UPnP ID zariadenia je *urn:demo-micasaverde-com:device:MpcMaster:1* a má jednu službu s UPnP ID *urn:demo-micasaverde-com:serviceId:**MpcMultipleControl1***. Operácie, definované v tejto službe sú:

- setSetpoint - slúži na stavenie žiadanej hodnoty
- setMpcs - slúži na zadefinovanie ID softvérových zariadení MPC regulátorov (MpcClient), ktoré sú zodpovedné za riadenie jednotlivých priebehov, ďalej sa tu zadefinuje hranica, podľa ktorej sa rozhodne, ktorý regulátor bude vypočítavať akčný zásah a nakoniec slúži na samotné spustenie riadenia.
- (un)setBypass - slúži na zastavenie riadenia.

Riadenie pozostáva z jednoduchých krokov.

1. Načítanie žiadanej hodnoty.
2. Rozhodnutie, ktorý MpcClient má byť spustený,
3. Spustenie správneho MpcClienta, ktorý následne zavolá výpočet akčného zásahu a aj zabezpečí vykonanie akčného zásahu.
4. Vypnutie MpcClienta, aby nezačal samostatne riadiť proces.
5. Nastavenie časovača, podľa periódy vzorkovania, kedy sa má riadiaca slučka znovu spustiť.

V prípade ak nad MpcClient zariadením preberá zodpovednosť MpcMaster, tak perióda vzorkovania, nastavená v zariadení MpcClient je ignorovaná, lebo zariadenie je plne pod správou MpcMaster. V tomto bode sú vytvorené všetky potrebné súčasti na riadenie pomocou prediktívneho regulátora v IoT prostredí.

3.2.3 Experiment riadenia prostredníctvom MPC

V tejto kapitole je vykonané overenie riadenia pomocou prediktívnej metódy v IoT prostredí. Samozrejme IoT prostredie je systém popísaný v kapitole 3.1. Na experiment overenia riadenia sú využité tieto súčasti.

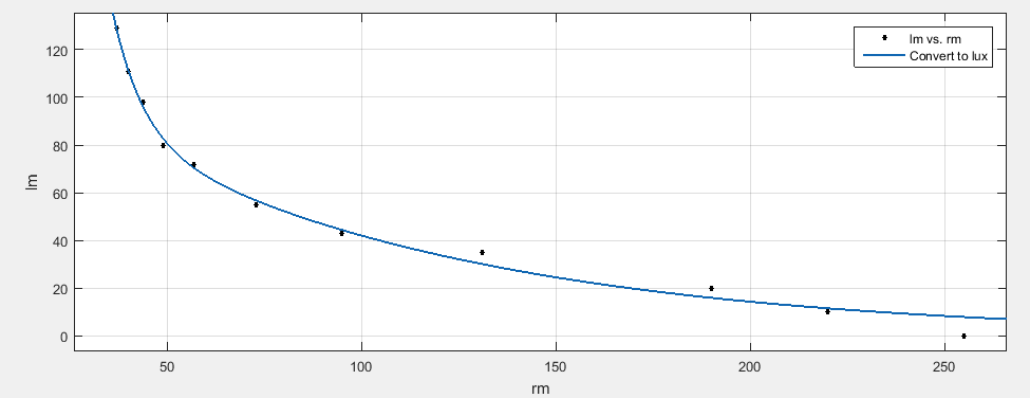
- Z fyzickej vrstvy sú využité zariadenia - stmievač FIBARO FGD 211 a fotorezistor pripojený na Raspberry Pi model B.
- Na IoT bráne sú použité súčasti - softvérové zariadenia - 1x RestHandler na načítanie intenzity žiarenia, 2x MpcClient na získavanie hodnôt z IoT Backend časti, potrebných na riadenie a 1x MpcMaster na riadenie MpcClientov.
- Nakoniec je využitý IoT Backend popísaný v kapitole 3.2.1 na vykonávanie výpočtov prediktívneho regulátora.

Na overenie riadenia je zvolený proces udržiavania žiadanej intenzity svetla pomocou svetelného zdroja, ktorý je ovplyvňovaný univerzálnym stmievačom FIBARO FGD 211 a pomocou fotorezistora, ktorý sníma intenzitu svetla.

Ešte pred identifikáciou systému popíšeme spôsob prepočtu informácie z fotorezistora, ktorá je v jednotkách odporu (Ohm) na jednotku intenzity svetla (Lux). Na získanie referenčnej hodnoty intenzity svetla bol jednorázovo použitý lux meter typu PSH Ambient Light sensor od Intel inc., ktorý nie je pripojený do IoT systému. Údaje namerané v celom rozsahu stmievača v hodnotách po 10% od 0% po 100% v testovacej miestnosti, sú zobrazene v grafe 36, čiernými bodmi. Na osi x je hodnota rm - meraný odpor z fotorezistora a na osi y lm - meraná intenzita osvetlenia. Bodmi zobrazenými v grafe 36 je pomocou *cftool* (Curve fitting tool) funkcionality nástroja Matlab preložená funkcia identifikujúca potrebný prepočet znázornená modrou farbou. Rozsah hodnôt fotorezistora je od 0 po 255, kde 0 je najsvetlejší a 255 najtmavší bod.

Matematický opis funkcie je:

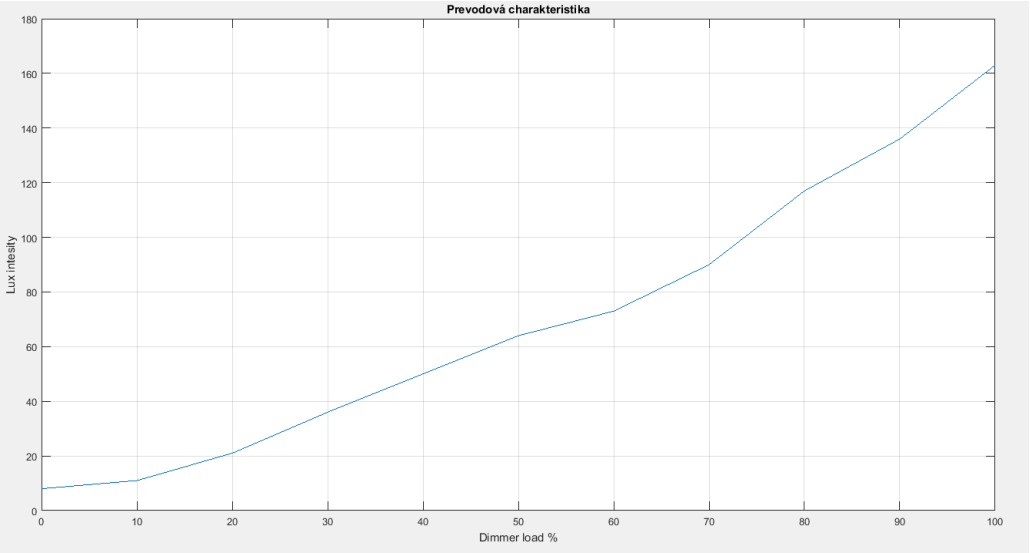
$$\begin{aligned} f(x) &= a * e^(b * x) + c * e^(d * x) \\ Koeficienty: & \\ a &= 5244 \\ b &= -0.1279 \\ c &= 123.2 \\ d &= -0.01074 \end{aligned} \tag{40}$$



Obrázok 36: Graf prepočtu odporu na intenzitu svetla.

Kde x je vstupný odpor a f(x) výstupná intenzita osvetlenia. Na základe toho, že fotorezistor, resp. AD prevodník dáva len hodnoty od 0 po 255 a že interpolačná funkcia je exponenciálna, treba zdôrazniť, že čím je intenzita osvetlenia väčšia tým je údaj nepresnejší a zároveň o to viac kolíše. V testovacej miestnosti sa intenzita pohybovala do 200 luxov (v závislosti od polohy merača), kde od hodnoty 140 luxov meranie začína kolísať v rozmedzí 40 luxov, čo uvidíme neskôr pri identifikácii. Tento fakt neskôr zohľadníme pri vyhodnocovaní kvality riadenia.

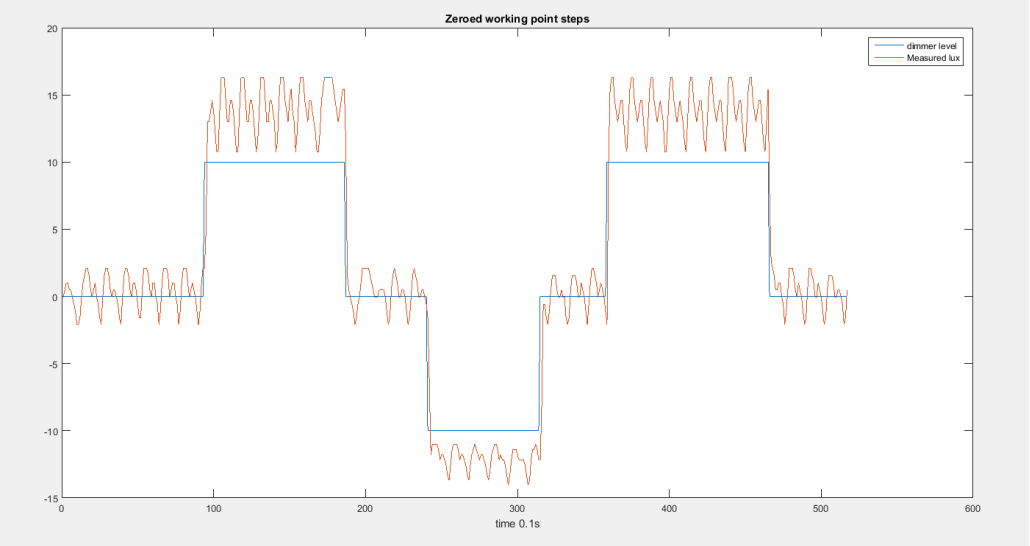
Prejdeme k samotnej identifikácii systému. Na začiatku je vykonaná prevodová charakteristika zobrazená v grafe 37. Na osi y je intenzita osvetlenia v luxoch a na osi x je



Obrázok 37: Prevodová charakteristika závislosť intenzity osvetlenia od percenta zotmenia

miera zotmenia stmievača v %. Z grafu je možné vidieť, že systém nie je lineárny. V prvej časti grafu od 20% do 60% na osi x je možné vidieť lineárnu oblasť, kde na zmenu o 40%

intenzita narástla o necelých 25 luxov na osi y. Potom od 60% do 100% na osi x je druhá lineárna oblasť na zmneu 40% intenzita narástla o viac ako 40 luxov na osi y. Na základe týchto dvoch oblastí sú zvolené dva pracovné body: miera zotmenia 30% a 80% na osi x. Hranica, ktorá tieto dve oblasti oddeľuje je približne 75 luxov na osi y. Po zvolení pracovných bodov je potrebné spraviť identifikáciu systému pre tieto dva body. Pri identifikácii bola perióda vzorkovania čítania hodnoty intenzity osvetlenia nastavená na 0.1 sekundy pri riadení je to nastavené už len na každých 5 sekúnd. Dáta namerané pre identifikáciu v pracovnom bode 30 sú zobrazené v grafe 38. Dáta v tomto grafe sú už posunuté do bodu 0 vstup v percentách o 30% a výstup systému o strednú hodnotu ustáleného skoku, čo je 48 luxov. Následne namerané a posunuté dáta sa použijú na identifikáciu systému. Tá

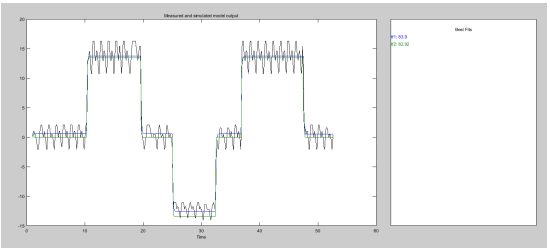


Obrázok 38: Odozva systému v pracovnom bode 30, po posunutí hodnôt do bodu 0

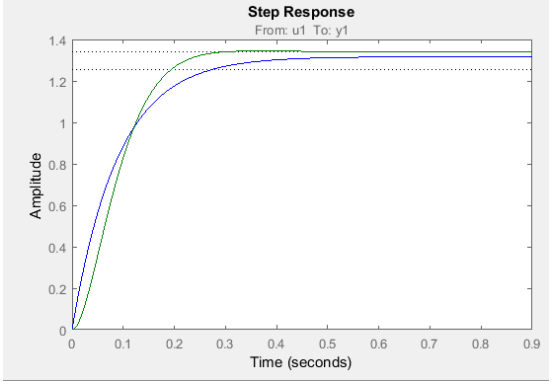
je vykonávaná prostredníctvom funkcionality *ident* nástroja Matlab. Nástroj poskytuje viacero metód identifikácie, v experimente je použitá identifikácia prostredníctvom modelov prenosových funkcií, kde je zvolená identifikácia na systém 2. rádu. a pre porovnanie aj systém 3. rádu. Výstupné grafy identifikácie sú znázornené na obrázku 39a a odozvy identifikovaných funkcií sú na obrázku 39b.

Na obrázkoch sú znázornené čiernou čiarou nameraný výstup, modrou farbou prechodová funkcia 2. rádu a zelenou farbou prechodová funkcia 3. rádu. Keďže funkcia 2. rádu presnejšie opisuje existujúci systém táto prechodová funkcia je vybratá na použitie do riadenia pomocou MPC, ktoré potrebuje matematický model. Táto prenosová funkcia má tvar:

$$G_1 = \frac{14.71 * s + 0.0291}{s^2 + 11.17 * s + 0.0232} \tag{41}$$



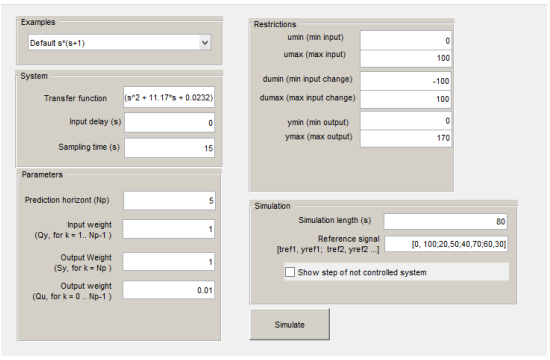
(a) Odozva na vstup pre bod 30.



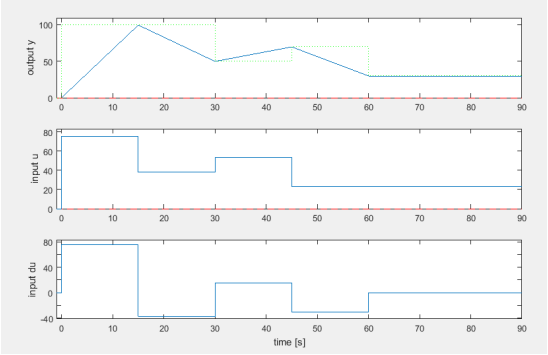
(b) Prenosové funkcie pre bod 30.

Obrázok 39

Pred aplikovaním funkcie do procesu je vyskúšaná v simulačnom nástroji opisovanom na konci prvej časti práce - kapitola 1.3. Parametre a výstup simulácie sú na obrázkoch 40a a 40b. Dôležité informácie na týchto obrázkoch sú voľba periódy vzorkovania až na



(a) Parametre simulácie pre prvú prenosovú funkciu



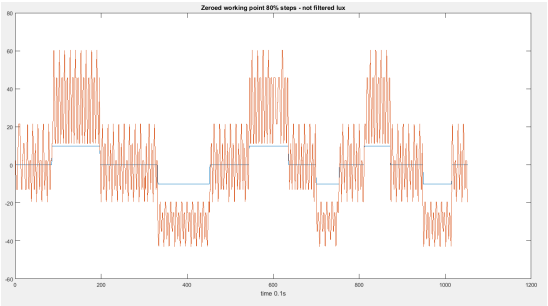
(b) Výstup simulácie pre prvú prenosovú funkciu

Obrázok 40

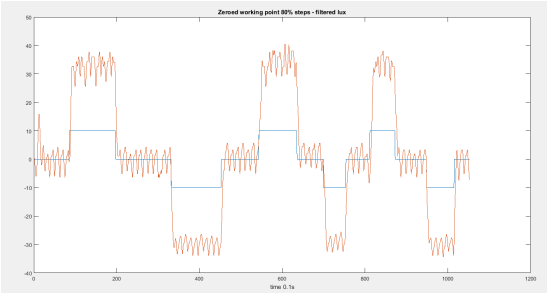
15 sekúnd. Táto hodnota je zvolená kvôli tomu, aby sme týmto jedným softvérovýhm zariadením a riadiacim procesom nevyťažili zdroje VeraLite brány natoľko, že by ostatné funkcie fungovali bez oneskorenia. Ďalej sú tu nastavené obmedzenia procesu tak, aby korešpondovali s reálnymi možnosťami zariadení. Obmedzenie na vstup je od 0 po 100 v našom prípade % a zmena môže byť či už z 0 na 100 alebo zo 100 na 0, takže obmedzenia na zmenu vstupu sú na hodnotách -100 a 100. Nakoniec obmedzenia na výstup sú od 0 po 170 luxov. Váhy prediktívneho regulátora nie je potrebné meniť, pretože výstupná veličina sleduje žiadanú hodnotu s požadovanou kvalitou riadenia.

Keď sme predchádzajúcou prenosovou funkciou dali riadiť systém v celom rozsahu,

tak od hodnoty 100 luxov už dochádzalo k trvalým regulačným odchýlkam. Preto je nevyhnutné robiť identifikáciu pre druhý pracovný bod 80%. Rovnaký postup ako pre bod 30 sa namerali údaje so skokmi okolo pracovného bodu. Namerané údaje je možné vidieť na obrázkoch 41a a 41b, opäť sú namerané dáta posunuté do bodu 0. V tomto prípade bol signál natoľko zašumený, že sme aplikovali filter pomocou pohyblivého priemeru (moving average) po 7 vzorkách takže za čas 0.7 sekundy. Na obrázkoch je vidieť rozdiel. Pre účely identifikácie bol zvolený odfiltrovaný signál, pretože identifikačný nástroj pre nefiltrovaný signál nedával vhodné modely.



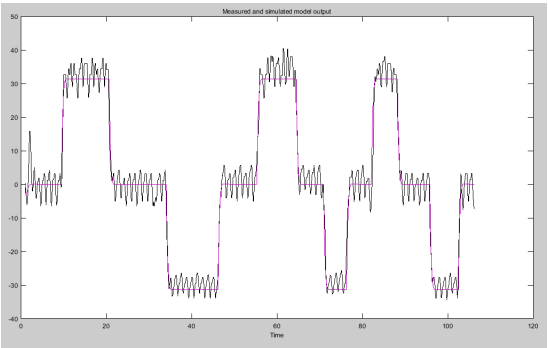
(a) nefiltrovaná



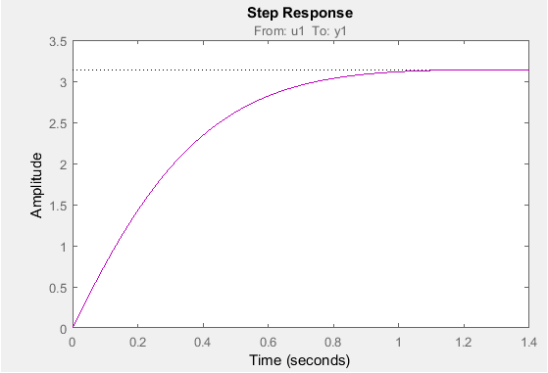
(b) filtrovaná

Obrázok 41: Závislosť intenzity osvetlenia od percenta zotmenia v pracovnom bode 80%.

Po tom ako je signál odfiltrovaný ident funkcionalita vráti kvalitnejšie modely. Percento zhody je síce nižšie ako pri menej zašumenom signály v predchádzajúcom pracovnom bode, ale nemá to vplyv na kvalitu regulácie. Odozvy identifikovaného systému sú znázornené na obrázku 42a, prechodová funkcia je na obrázku 42b. Matematický tvar prenosovej



(a) Odozvy systému v pracovnom bode 80.



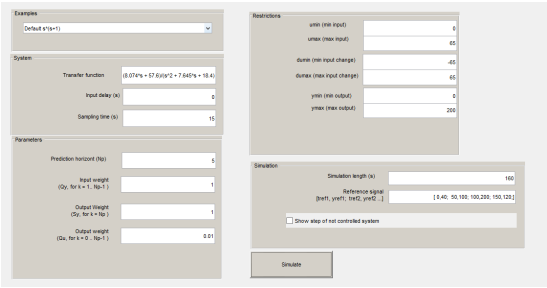
(b) Prenosová funkcia systému v pracovnom bode 80.

Obrázok 42

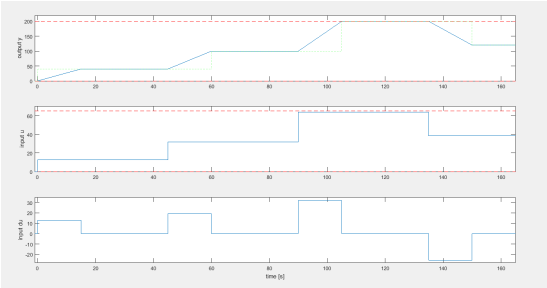
funkcie v pracovnom bode 80 je:

$$G_2 = \frac{8.074 * s + 57.6}{s^2 + 7.645 * s + 18.4} \tag{42}$$

Pre vypočítanú prenosovú funkciu sú vykonané simulácie, kde parametre a výstup sú znázornené na obrázkoch 43a a 43b. V tomto prípade treba upozorniť na obmedzene na



(a) Parametre simulácie pre druhú prenosovú funkciu



(b) Výstup simulácie pre druhú prenosovú funkciu

Obrázok 43

vstup, ktoré je nastavené od 0 po 65%. Ako je možné vidieť v strednom grafe obrázku 43b táto hodnota nie je dosiahnutá ani pri maximálnej intenzite osvetlenia, ktorá bola v miestnosti dosiahnutá - 200 luxov. V tejto simulácii je pracovný rozsah stmievača od 0 po 65%. Tým, že merané hodnoty boli pri identifikácii posúvané k nule, ale hlavne tým, že systém nie je lineárny a dolnú časť má na starosti iný regulátor k akčnému zásahu regulátora pre hornú oblasť je potrebné vždy pridať 100-65 = 35%. Takto identifikované systémy a parametre sú aplikované do pripraveného IoT prostredia.

1. Prenosová funkcia G_1 (41) je aplikovaná do jednej inštancie MpcClient zariadenia.

2. Prenosová funkcia G_2 (42) je aplikovaná do druhej inštancie MpcClient zariadenia a offset akčného zásahu je nastavený na 35%.

3. Na správu dvoch MpcClient zariadení je vytvorená jedna inštancia MpcMaster zariadenia, ktorá má na vstupe identifikované dve predošlé inštancie a hranicu 75 luxov žiadanej hodnoty ako rozhodujúcu. Ak je žiadaná hodnota pod touto hodnotou, tak MpcMaster volá prvú inštanciu regulátora, aby poskytla údaje o akčnom zásahu zo servera, ak je žiadaná hodnota vyššia, tak volá druhú inštanciu regulátora, ktorá je zodpovedná za získanie akčného zásahu zo servera a jeho uskutočnenie.

4. Na zaznamenávanie údajov vo VeraLite zariadení je použité softvérove zariadenie dataMine [5] - plugin do VeraLite zariadenia, ktoré okrem ukladania umožňuje aj

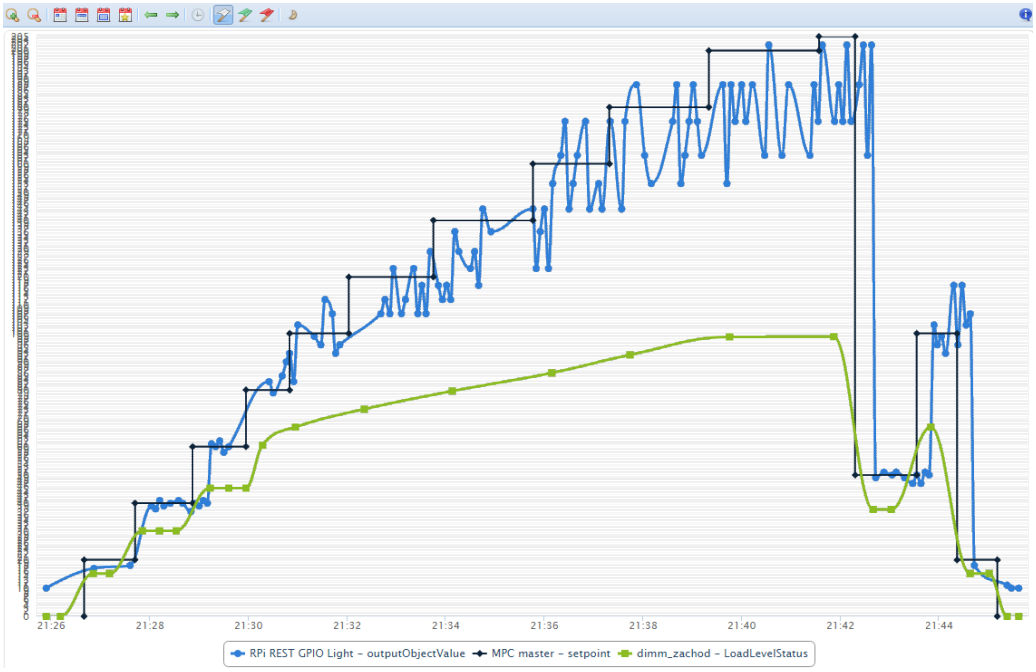
robiť grafy z nameraných údajov. Forma ukladania dát je v podobe časová značka, hodnota. Namerané dáta je potrebné ešte upraviť, pretože každé zariadenie má inú periódu vzorkovania, je potrebné ju zjednotiť. Na zjednotenie periódy vzorkovania je vytvorený jednoduchý skript v nástroji Matlab naznačený v algoritme 6, ktorý prejde vektory vstupných časov a vstupných hodnôt a do každej „medzery“ medzi časmi vloží poslednú nameranú hodnotu. To isté sa spraví pre všetky veličiny, ktoré je potrebné zjednotiť, čím sa získajú dáta o perióde vzorkovania 1s pre všetky veličiny.

Algoritmus 6 Príprava dát na identifikáciu

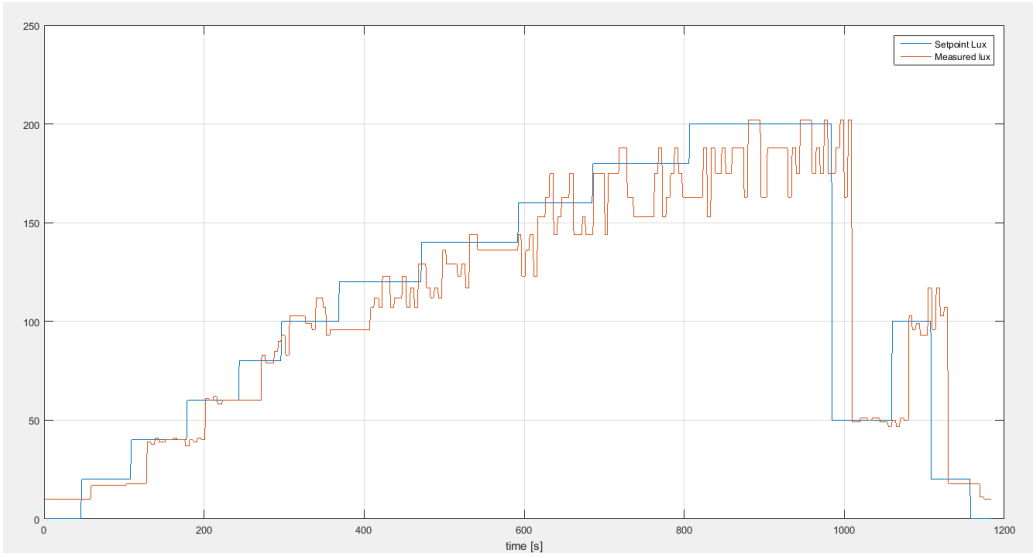
```
function output=fillval(inputtime,inputvalue)
output = zeros(1,floor(inputtime(length(inputtime))));
for j=1:floor(inputtime(1))
    output(j)=inputvalue(1);
end

for i=2:length(inputtime)
    for j=floor(inputtime(i-1)):floor(inputtime(i))
        output(j)=inputvalue(i-1);
    end
end
```

S takto pripravenými zariadeniami sa spúšťa proces riadenia pomocou tlačítka v Mpc-Master zariadení, v ktorom sa rovnako postupne mení žiadaná hodnota. Zaznamenané zmeny nameranej veličiny a akčné zásahy sú v spomenutom DataMine zariadení. Ukážka grafov, ktoré pripravuje DataMine je na obrázku 44. Exportované a spracované dáta experimentu v Matlabe sú na obrázkoch 45 a 46.

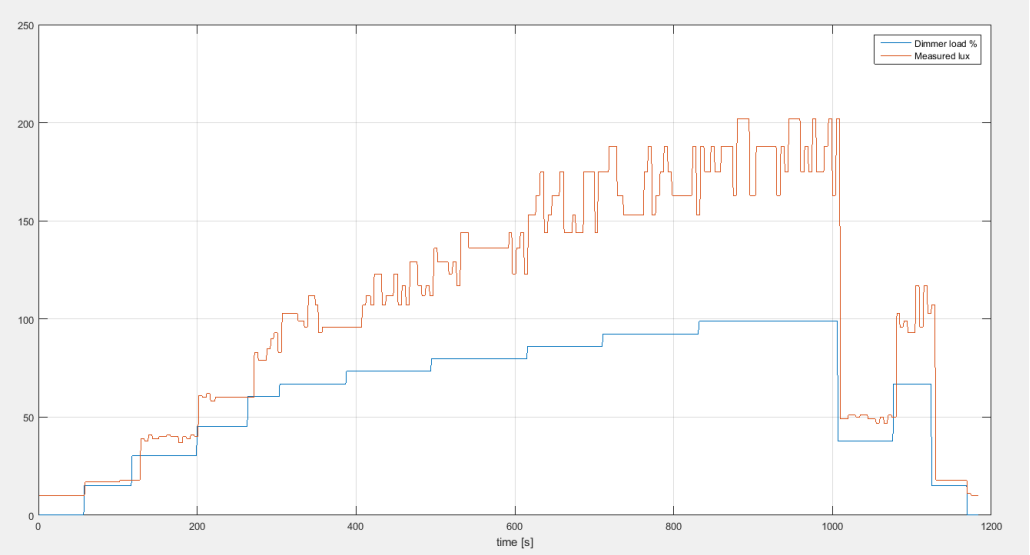
Na obrázku 45 vidieť modrou farbou žiadanú hodnotu a červenou farbou meranú výstupnú veličinu. Kroky žiadanej hodnoty sú na začiatku po 20 luxov až po 200. Následne je skok na 50 luxov, na 100 luxov a nakoniec na 20. Z tohto grafu nie je vidieť, v ktorom čase riadi akčný zásah iná inštancia MPC regulátor. Keďže pravidlo je pomerne jednoduché. prvé akčné zásahy vykonávala prvá inštancia, až kým žiadaná hodnota neprekročila úroveň 75 luxov. Potom až po 200 luxov vykonávala akčné zásahy druhá inštancia. Následne sa vystriedali a nakoniec posledné akčné zásahy vykonávala prvá inštancia. Ohľadne kvality regulácie môžeme vidieť, že po každej zmene žiadanej hodnoty je dopravné oneskorenie regulátora. To je spôsobene dvoma faktormi. Prvý je zvolená perióda vzorkovania 15s a



Obrázok 44: Namerané dáta pomocou DataMine.



Obrázok 45: Sledovanie žiadanej hodnoty výstupnou hodnotu intenzity osvetlenia.



Obrázok 46: Závislosť meranej hodnoty od percenta zotmenia.

druhý je čas výpočtu kroku, ktorý momentálne beží na zariadení, ktoré nemá požadovaný výkon. Keďže nejde o kritický proces naše požiadavky takéto dopravné oneskorenie akceptujú. Ďalej je možné pozorovať už spomenutý fakt nepresnosti a zašumenia merania intenzity osvetlenia pri vyššej intenzite. Do žiadanej hodnoty 100 luxov je stredná hodnota takmer zhodná so žiadanou. Následné hodnoty sú už viac zašumené, preto sa kvalita regulácie komplikovanejšie určuje. Avšak pri skokoch na 50 a 100 kedy sa testuje prepínanie regulátorov, je možné vidieť takmer nulovú regulačnú odchýlku. Pri žiadanej hodnote 20 luxov vidieť trvalá regulačnú odchýlku, ktorá je pravdepodobne spôsobená tým, že systém ma od 0 po 10 % ďalšiu nelinearitu, ktorá nie je zohľadnená pri návrhu riadenia. Rovnakým princípom ako sú vyriešené 2 pracovné body, je možné vyriešiť N pracovných bodov. V rozsahu, kde sú stabilné merania, prediktívny regulátor riadi proces s výbornými ukazovateľmi kvality regulácie a princíp CaaS - vypočítania hodnôt akčných zásahov na servery je teda plne funkčný.

Tento princíp je možné využiť napríklad na udržiavanie intenzity svetla so zapadajúcim slnkom. Ďalšie využitie výpočtového výkonu servera a neobmedzený počet regulátorov je na regulovanie teploty pomocou termostatu, ktorý v pripravenom IoT systéme je možné realizovať. Avšak experiment sa dial mimo vykurovaciu sezónu, preto nebolo možné uskutočnenie experimentu aj na tepelnom systéme. Možností využitia princípu CaaS sú týmto spôsobom neobmedzené, keďže pridanie ďalšieho regulátora je len vytvorenie novej inštancie. Stačí si zvoliť proces, ktorý treba riadiť, jeho akčný člen, senzor a identifikovať systém. Nakoniec spustiť riadiacu slučku. Pre lineárne procesy sa spúšťa na zariadení

MpcClient pre nelineárne procesy pomocou viacerých MpcClientov a jedného MpcMaster zariadenia.

3.3 Činnosti súvisiace s uvádzaním IoT systému do praxe

Táto kapitola sumarizuje praktické skúsenosti z návrhu, realizácie a udržiavania IoT systému. Zdôrazňuje výhody, ale aj výzvy, ktoré IoT a teda vytváranie užšej väzby medzi IT (informačnými technológiami) a OT (operačnými technológiami) prináša. Kapitola porovnáva činnosti potrebné pre čisto softvérové systémy a činnosti potrebné pre IoT systémy.

3.3.1 Návrh

Základný nástroj pri návrhu softvérových systémov je modelovací nástroj s podporou napr. UML. V ňom sa modelujú základné prípady použitia systému, komponenty systému, sekvenčné diagramy a diagramy aktivít. Táto časť je spoločná pre oba typy systémov. Pre oba typy systémov sa volia programovacie jazyky pre jednotlivé komponenty a spôsob komunikácie medzi nimi. Pri IoT systémoch navyše zohráva veľkú úlohu výber hardvérových komponentov a protokolu, pomocou ktorého hardvérové komponenty budú podávať informácie. V prípade, že neexistuje zariadenie, ktoré by spĺňalo požiadavky finálneho systému je potrebné riešiť návrh samotných hardvérových komponentov, čo je ďalšia rozsiahla tematika. Rovnako výber vhodného protokolu je neľahká úloha. Ako možno vidieť na obrázku 26 v časti komunikačná úroveň, na ktorom zďaleka nie sú vymenované všetky existujúce protokoly, sú rôzne typy protokolov na rozličné spôsoby použitia. Ich využitie výhody a nevýhody sa líšia a všetky sa v čase vyvíjajú. To v praxi znamená, že pri návrhu je potrebné mať prehľad o protokoloch a zmenách v čase. Ďalšia oblasť pri návrhu systému je jeho bezpečnosť. V softvérových systémoch ide len o kybernetickú bezpečnosť, zatiaľ čo pri IoT systémoch treba dbať aj na fyzickú bezpečnosť. Ohľadne kybernetickej bezpečnosti v IoT systéme treba povedať, že IoT systém vystavuje viacero miest potenciálnych útokov a systém je tak bezpečný ako jeho najzraniteľnejšie miesto. Ak sa pozrieme na IoT systém popisovaný v kapitole 3.1, tak ide o systém, ktorý je v rámci lokálnej domácej siete a preto zabezpečenie nie je na úrovni systému podnikovej siete a určite nie na úrovni siete internet. O tomto systéme môžeme povedať, že je tak bezpečný ako je zabezpečený prístup do domácej siete, ale zároveň aj zabezpečenie siete Z-Wave, ktorá je bezdrôtová a teda je možné ju jednoducho odpočúvať. Nevyhnutnosť v obidvoch prípadoch, Z-Wave aj Wifi, je šifrovanie komunikácie. Ďalší bod zabezpečenia je prípojka do internetu vo väčšine domácich sieti zohráva túto úlohu router. Tu je nevyhnutné zamedziť nežiadúcim prístupom z internetu do lokálnej siete. Na druhej strane

treba povoliť prístup žiadúcim prístupom napríklad pre prípad ovládania na diaľku. V popisovanom systéme je tento bod vyriešený VPN server aplikáciou bežiacou na routeri, ktorá umožňuje klientom, prostredníctvom vydaných certifikačných kľúčov, sa naň pripojiť a po pripojení ovládať zariadenia lokálnej siete. Ďalší aspekt týkajúci sa bezpečnosti domácej siete je úroveň oprávnenia na rôzne typy úkonov pri správe. Tento nedostatok sa dá vytknúť zvolenému zariadeniu VeraLite, ktoré má iba dve úrovne Admin a Guest, kde admin môže všetko ovládať aj konfigurovať a Guest môže všetko ovládať. Chýba obmedzenie na ovládanie len určitej množiny zariadení, ktoré môže hosť ovládať. Časť o kybernetickej bezpečnosti uzatvárame tým, že je nevyhnutné mať znalosti o spôsoboch zabezpečenia IT, ktoré podľa možnosti aplikovať na OT. Téma fyzickej bezpečnosti je na druhej strane viac rozpracovaná v OT pri návrhu výrobných liniek a podobne. V IoT systémoch sa kybernetická bezpečnosť spája s tou fyzickou, keď napríklad hovoríme o zásuvkách, dverách, chladničkách ovládaných na diaľku, ak sa pozeráme len v merítku oblasti inteligentných budov. O to viac prepojenie narastá v oblasti systémov pre spojené autá, inteligentnú mestskú infraštruktúru a podobne. V tomto bode IoT systémy otvárajú viaceré legislatívne otázky týkajúce sa zodpovednosti za spôsobené škody akýmkoľvek IoT systémom. Právna oblasť bude ešte ovplyvňovať smer a návrhy IoT systémov. Preto ak by sme definovali pozíciu architekta IoT systému, jeho znalosti a kompetencie sú v týchto doménach, či technológiách.

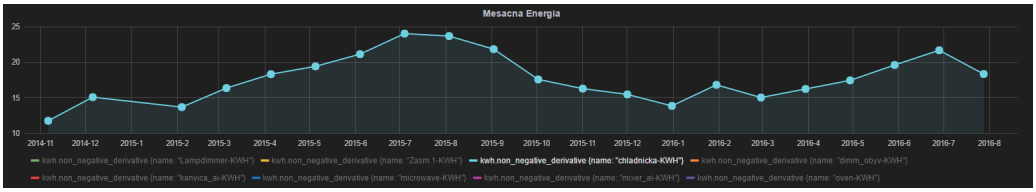
- UML a znalosť možností programovacích jazykov potrebných na vytvorenie IoT Backend častí
- znalosť senzorov, akčných členov (pohonov, relé, ventilov, ...)
- znalosť protokolov a možností prepojenia jednotlivých protokolov na vytvorenie vrstvy IoT zariadení a IoT brán
- v určitých prípadoch znalosť návrhu plošných spojov
- bezpečnosť informačných systémov - autentifikácia a autorizácie na základe rolí, ...
- bezpečnosť informačný sietí - šifrovanie komunikácie, VPN, ...
- bezpečnosť zariadení
- právne aspekty domén nasadzovania IoT systémov

Tento zoznam nie je kompletný, ale snaží sa zaznamenať najdôležitejšie postrehy z praxe a poukázať na komplexitu, ktorú IoT systémy k pôvodnej softvérovej architektúre pridávajú.

3.3.2 Vývoj

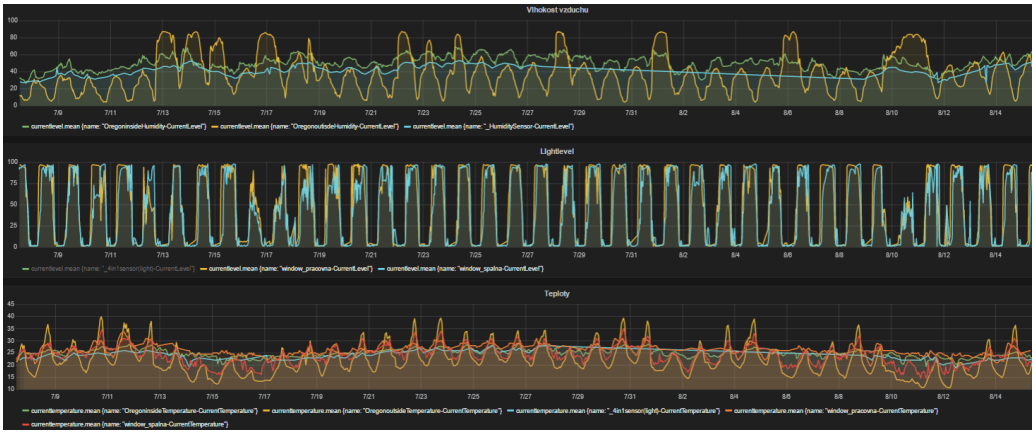
Vývoj softvéru a ako už bolo spomenuté aj hardvéru nepochybne patrí do činnosti uvádzania IoT systémov do praxe. Kapitola sa viac zameriava na softvér, pretože vývoj hardvéru v našom prípade nebol potrebný. Aktuálne pri vývoji IoT systému len zriedka postačí jeden programovací jazyk. Je to z dôvodu rôznych typov zariadení v jednom ekosystéme, s rôznymi nárokmi na výkon a na správanie. Ak začneme od najjednoduchších zariadení, tak firmvér zvyčajne býva písaný v jazyku C, samozrejme sú aj ďalšie alternatívy. V našom systéme firmvér nebolo potrebné písať, pretože väčšina zariadení bola zaobstaraná, aby fungovala so Z-Wave protokolom. Najbližšie k hardvéru je v práci poskytovanie údajov z fotorezistora, cez PCF8591 AD prevodník. Framework Node.js zohráva svojimi nízkymi nárokmi dôležitú úlohu pri vývoji aplikácii na zariadení typu Raspberry Pi. Zariadenie Raspberry Pi a jeho konkurenti zohrávaju tiež dôležitú úlohu pri zavádzaní IoT systémov. Sú to ideálne zariadenia na vytváranie prepojenia s IoT backendom. V našom prípade je ako rozhranie zvolené VeraLite zariadenie, ktoré však svojím výkonom a parametrami je s nimi porovnateľné. Vývoj na úrovni brán je stále s jazykmi s nízkymi nárokmi na výpočtový výkon, ale stále poskytujúce rýchlu odozvu. Ako už bolo spomenuté, tak vo VeraLite je použitý interpretovaný jazyk Lua, ktorého interpreter vykonáva funkcie knižníc napísaných v jazyku C. V oblasti inteligentných domov je často v bráne (agregátore) vizualizácia pre koncového používateľa. Rozhranie prístupné na VeraLite zariadení je optimalizované pre Z-Wave zariadenia s možnosťou integrácie ďalších typov. Vo svete existujú aj projekty typu openHAB [24], ktoré sa tvária nezávisle od protokolu, ktorým sú zariadenia pripojené. Tieto nástroje slúžia ako softvér na vytvorenie centrálnej brány v lokálnej sieti a zastávaju funkciu správy a vizualizácie zariadení. Základne prvky automatizačných funkcií sú tiež podporované na tejto úrovni zariadení. Čo sa týka nezávislosti na protokole v konečnom dôsledku je potrebný plugin, aby určitý typ zariadení podporovaný bol. Keď si zoberieme napríklad Z-Wave protokol, tak na to, aby boli vo vizualizácii Z-Wave zariadenia zobrazené, musí byť naprogramovaný plugin, ktorý to umožní. Takže veľmi podobný prípad, ako integrovanie iných zariadení do VeraLite brány. Zhrnúť to môžeme tak, že mať vizualizáciu ešte v lokálnej sieti je takmer nevyhnutné, aby bolo možné riadiť dom aj v prípade výpadku internetu, ktorá vizualizácia sa zvolí je takmer jedno, na jednej strane vždy treba niečo integrovať. Ďalšia súčasť IoT systémov je IoT backend, ktorého vývoj sa najviac podobá existujúcim čisto softvérovým systémom. Na tejto úrovni sú dátové body natoľko abstraktné, že IoT backend nemusí rozoznávať, akým spôsobom boli získané. Faktor, ktorý sem vstupuje je množstvo dát. Systém nemusí byť IoT, aby spracovával veľa údajov, avšak už bolo spomenuté, že s IoT sa často spája

pojem Big data. Toto je práve oblasť, ktorú treba zvládnuť na IoT backende a ktorá dáva IoT systému ďalšiu pridanú hodnotu. Okrem štandardných jazykov, platforiem a databáz, ktorými sa aplikácie programujú, je potrebné sledovať vývoj nových jazykov, platforiem, databáz. V práci je použitý aj klasický prístup na aplikovanie moderných metód riadenia, kde je zvolený jazyk Java, ako jeden z najstabilnejších a najpoužívanejších. Ako úložisko je použitá MongoDB databáza, ktorá je menej konvenčná, ale trendy ukazujú, že čo chvíľa sa z nej stane konvenčná databáza. Oproti tradičným relačným databázam typu MySQL sa líši tým, že nemá tabuľky, ale kolekcie dokumentov a nemá riadky a stĺpce, ale dokumenty a atribúty dokumentu a štruktúra atribútov nie je definovaná. Ďalej je v práci vyskúšaný jeden z moderných prístupov spracovania dát. Základné súčasti, ktorými sa Big Data zaoberá sú spracovanie, uloženie, vizualizovanie, analyzovanie a vyhodnocovanie veľkého objemu dát. Oproti v práci spomínaným komerčným platformám MS AZURE IOT HUB, AMAZON AWS IOT existujú aj open source platformy na prácu s dátami. V čase písania práce perspektívne vyzerajúce sú dve sady nástrojov. Jedna z nich je ELK [15] sada a druhá TICK [33] sada. Prvá je prezentovaná ako výborný nástroj na sledovanie logov, samozrejme využitie nie je limitované. Druhá je prezentovaná ako výborný nástroj na prácu s časovými radmi opäť využitie nie je limitované. Druhá menovaná je v práci vyskúšaná avšak len 3 nástroje zo štyroch a ich výstup je možné vidieť na obrázkoch 47, 48, 49 a 50 . Štvrtý nástroj slúžiaci na analýzu dát zasahuje do oblasti „data science“, ktorá sa zaoberá machine learning, deep learning a štatistickými modelmi, čo je ďalšia rozsiahla téma. Základné vyhodnotenia anomálnych, rizikových stavov v našom systéme bežia už vo VeraLite bráne, ktorá základnú logiku podporuje. Avšak logika vyplývajúca z dlhodobo nazbieraných dát v práci vyhodnocovaná nie je. Z vizualizácii je možné potvrdiť fakty, ktoré sa pomerne ľahko dajú predpokladať.



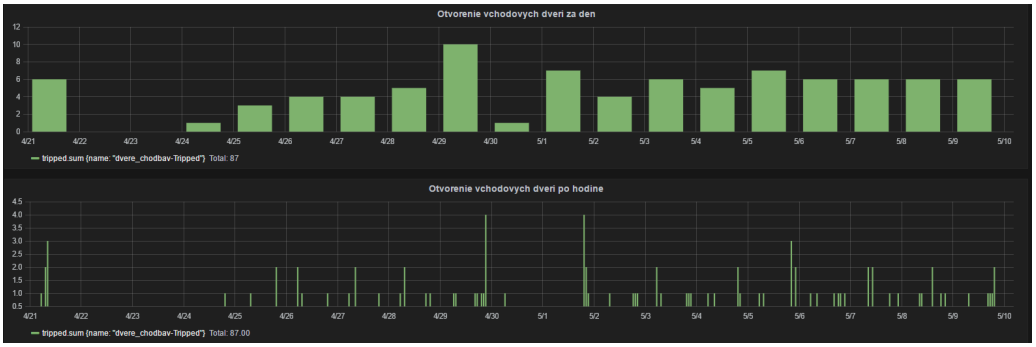
Obrázok 47: Spotreba chladničky po mesiacoch.

Na obrázku 47 je možné pozorovať spotrebu chladničky po mesiacoch, kde je vidieť, že v letných mesiacoch jej spotreba stúpa. Grafy na obrázku 48 ukazujú základné ukazovatele životného prostredia. Grafy na obrázku 49 zobrazujú počty zapnutí jednotlivých svetiel za deň a po hodinách časy a frekventované miestnosti nie sú prekvapujúce. Z grafov 50 je možné odčítať otvorenia vchodových dverí, z čoho by sa dal určiť čas zvyčajného príchodu



Obrázok 48: Grafy, vlhkosti vzduchu, intenzity osvetlenia a teploty.

Obrázok 49: Počty zapnutí svetiel.



Obrázok 50: Počty otvorenia vchodových dverí.

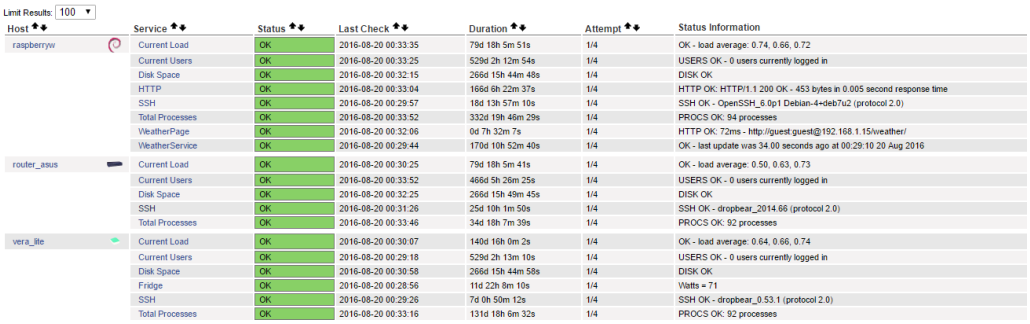
a odchodu domov.

Pole pôsobenia vývojára IoT systému je tak ako v prípade architekta IoT systému rozšírené. Rozdeliť by sa znalosti dali na skupiny.

- Programovanie IoT zariadení - jazyky blízke hardvéru.
- Programovanie IoT brán - jazyky s nízkymi nárokmi na výkon, ale vysokými na rýchlu odozvu.
- Programovanie IoT backend - zahŕňajúce aktuálne konvenčné technológie.
- Programovanie IoT backend - zahŕňajúce nové rýchlo rozvíjajúce sa technológie na spracovanie a analýzu dát veľkých objemov, rýchlosti a variability.

3.3.3 Prevádzka

Posledná oblasť činností je údržba IoT systému. Čisto softvérové systémy spravujú administrátori, pre ktorých existujú nástroje, ktoré sledujú stav serverov a služieb bežiacich na serveroch. Rovnaká požiadavka je potrebná aj pre IoT systém, ktorý ale má viacero úrovni a pod kontrolou je potrebné mať všetky úrovne. Starosť o IoT backend a o IoT bránu sa nelíšia. Pre účely monitorovania systému je v práci použitý nástroj Nagios [23]. Tento nástroj umožňuje čiastočne konfiguračne a čiastočne implementačne doplniť sledovanie stavu rôznych zariadení. Pre klasické služby má pripravenú sadu použiteľnej konfigurácie. Na obrázku 51 je znázornené grafické rozhranie, ktoré ukazuje stav hlavných zariadení a služieb bežiacich na nich. Doplňujúci prvok k tomuto nástroju je v práci


Limit Results: 100

Host	Service	Status	Last Check	Duration	Attempt	Status Information
raspberryw	Current Load	OK	2016-08-20 00:33:35	79d 18h 5m 51s	1/4	OK - load average: 0.74, 0.66, 0.72
	Current Users	OK	2016-08-20 00:33:25	529d 2h 12m 54s	1/4	USERS OK - 0 users currently logged in
	Disk Space	OK	2016-08-20 00:32:15	266d 15h 44m 48s	1/4	DISK OK
	HTTP	OK	2016-08-20 00:33:04	166d 6h 22m 37s	1/4	HTTP OK: HTTP/1.1 200 OK - 453 bytes in 0.005 second response time
	SSH	OK	2016-08-20 00:29:57	18d 13h 57m 10s	1/4	SSH OK - OpenSSH_6.0p1 Debian-4+deb7u2 (protocol 2.0)
	Total Processes	OK	2016-08-20 00:33:52	332d 19h 46m 29s	1/4	PROCS OK: 94 processes
	WeatherPage	OK	2016-08-20 00:32:06	0d 7h 32m 7s	1/4	HTTP OK: 72ms - http://guest:guest@192.168.1.15/weather/
	WeatherService	OK	2016-08-20 00:29:44	170d 10h 52m 40s	1/4	OK - last update was 34.00 seconds ago at 00:29:10 20 Aug 2016
router_asus	Current Load	OK	2016-08-20 00:30:25	79d 18h 5m 41s	1/4	OK - load average: 0.50, 0.63, 0.73
	Current Users	OK	2016-08-20 00:33:52	466d 5h 26m 25s	1/4	USERS OK - 0 users currently logged in
	Disk Space	OK	2016-08-20 00:32:25	266d 15h 49m 45s	1/4	DISK OK
	SSH	OK	2016-08-20 00:31:26	25d 10h 1m 50s	1/4	SSH OK - dropbear_2014.66 (protocol 2.0)
	Total Processes	OK	2016-08-20 00:33:46	34d 18h 7m 39s	1/4	PROCS OK: 92 processes
vera_lite	Current Load	OK	2016-08-20 00:30:07	140d 16h 0m 2s	1/4	OK - load average: 0.64, 0.66, 0.74
	Current Users	OK	2016-08-20 00:29:18	529d 2h 13m 10s	1/4	USERS OK - 0 users currently logged in
	Disk Space	OK	2016-08-20 00:30:58	266d 15h 44m 58s	1/4	DISK OK
	Fridge	OK	2016-08-20 00:28:56	11d 22h 8m 10s	1/4	Watts = 71
	SSH	OK	2016-08-20 00:29:26	7d 0h 50m 12s	1/4	SSH OK - dropbear_0.53.1 (protocol 2.0)
	Total Processes	OK	2016-08-20 00:33:16	131d 18h 6m 32s	1/4	PROCS OK: 92 processes

Obrázok 51: Rozhranie na sledovanie stavu spustených zariadení.

nainštalovaný vlastný mail server, ktorý zabezpečí poslanie notifikácie okamžite pri detekcii nefunkčnosti zariadenia alebo služby. Pre promptnejšie notifikácie sa to často rieši SMS bránou, ktorá odošle SMS pri probléme, toto však v práci vyskúšané nie je. Rozmer, o ktorý je IoT systém v tomto smere rozšírený je starosť opravu zariadení. Veľká téma

rozpracovaná v oblasti OT, na ktorú je potrebné upozorniť, keď sa hovorí o tom, čo IoT systém prináša. Nevyhnutná požiadavka pri výbere zariadenia je, aby vedelo informovať o stave bateriek, aby sa dala plánovať ich výmena. Počas prevádzky od apríla 2014 na opísaných zariadeniach došlo viac krát k výmene bateriek na niektorých snímačoch, čo navyše pri niektorých zariadeniach nebol jednoduchý proces. V tomto intervale došlo aj k znefunkčneniu dvoch spínačov, ktoré bolo potrebné vymeniť. Na základe týchto skúsenosti zdôrazňujeme, že pri odovzdávaní IoT systému je dôležite, aby bolo dohodnuté, kto systém spravuje a aký je reklamačný model resp. proces nastavenia správneho fungovania systému.

Záver

Za hlavný cieľ práce bolo aplikovať prediktívny regulátor do IoT systému, z čoho ako je už v úvode napísane vznikla myšlienka CaaS - regulátor ako služba, keďže MPC je metóda riadenia s pomerne náročnými požiadavkami na matematické výpočty, čím sa cieľ rozšíril o overenie tejto myšlienky. S aplikáciou prediktívneho riadenia do IoT systému a overovania myšlienky CaaS bol nevyhnutý návrh, implementácia, prevádzka a samozrejme aj definícia IoT systému, čo boli ďalšie činnosti rozširujúce cieľ práce. Vysvetlenie a odôvodnenie použitých architektonických princípov pri implementácii sme pokladali za povinnosť uviesť do práce. Rovnako postrehy z prevádzky IoT systému sme vyhodnotili ako dôležité v práci spomenúť, to už však len ako vedľajší cieľ práce.

Cieľ aplikovania prediktívneho regulátora do IoT systému považujeme za splnený. Rovnako overenie, či je možné MPC do IoT systému zaviesť konceptom CaaS, považujeme za splnené, keďže bol v práci navrhnutý, implementovaný a prevádzkovaný IoT systém v inteligentnej domácnosti, v ktorom sa pomocou MPC regulátora udržiavala hladina žiadanej intenzity osvetlenia a výpočty regulátora náročné na výpočtový výkon boli poskytované prostredníctvom IoT Backend súčasti, ktorá je charakteristická disponovaním veľkej kapacity výpočtového výkonu. Vyhodnotenie kvality riadenia naznačilo nedostatky tohto prístupu práve v potenciálnej obmedzenosti periódy vzorkovania regulátora a pri kritických procesoch závislosťou na pripojení 24/7. Avšak výhody, že je možné aj zložité výpočty realizovať v kvázi reálnom čase a že je možné vytvoriť ľubovoľný počet inštancií regulátorov a riadiť tak všetky procesy v danom IoT systéme, bez dodatočných nákladov, sú nesporné. Cieľ definície, návrhu, implementácie a prevádzky IoT systému považujeme za splený. V práci je pojem IoT zadefinovaný podloženými zdrojmi. IoT systém bol v práci podrobne popísaný a architektonické voľby zdôvodnené rovnako ako implementácia IoT systému, opierajúce sa o architektonické princípy definované v práci na základe zdrojov. Vedľajší cieľ nahliadnutia do prevádzky IoT systému tiež považujeme za splnený, pretože práca prináša pohľad na to, aké činnosti sú pre jednotlivé fázy životného cyklu IoT systému potrebné, aké kompetencie respektíve oblasti zájmu je potrebné sledovať, aby bola udržiavaná kvalita IoT systémov, či už novo vznikajúcich alebo existujúcich a pridaná hodnota IoT systémov nebola znižovaná ich nespoľahlivosťou, nedostatočným zabezpečením, či dokonca nebezpečnosťou prevádzky.

Zoznam použitej literatúry

[1] 10 iot predictions for 2016 - page: 1 | crn. http://www.crn.com/slide-shows/networking/300079629/10-iot-predictions-for-2016.htm. Accessed on 14-06-2016).

[2] Chapter 1: What is software architecture? https://msdn.microsoft.com/en-us/library/ee658098.aspx. (Accessed on 16-06-2016).

[3] Chapter 20: Choosing an application type. https://msdn.microsoft.com/en-us/library/ee658104.aspx. (Accessed on 16-06-2016).

[4] The clean architecture | 8th light. https://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html. (Accessed on 17-06-2016).

[5] Datamine graphing and logging plugin. http://code.mios.com/trac/mios_datamine. (Accessed on 23-06-2016).

[6] Definition of a crm system - crm system defined - aptean. http://www.aptean.com/additional-crm-and-erp-related-links-pages/crm-resources-folder/definition-of-a-crm-system. (Accessed on 18-06-2016).

[7] Dictionary information. http://www.apics.org/dictionary/dictionary-information?ID=993. (Accessed on 18-06-2016).

[8] February 2016 web server survey | netcraft. http://news.netcraft.com/archives/2016/02/22/february-2016-web-server-survey.html. (Accessed on 18-06-2016).

[9] Fibaro home center 2 z-wave controller - control devices in your smart home. http://www.fibaro.com/en/the-fibaro-system/home-center-2. (Accessed on 21-06-2016).

[10] Home. http://razberry.z-wave.me/. (Accessed on 21-06-2016).

[11] How to choose the right software architecture: The top 5 patterns. http://techbeacon.com/top-5-software-architecture-patterns-how-make-right-choice. (Accessed on 17-06-2016).

[12] Integracia-soa-rest.pdf. https://prest-tech.appspot.com/docs/Integracia-SOA-REST.pdf. (Accessed on 18-06-2016).

[13] Integration architecture: Comparing web apis with service-oriented architecture and enterprise application integration. http://www.ibm.com/developerworks/websphere/library/techarticles/1503_clark/1305_clark.html. (Accessed on 18-06-2016).

[14] The internet of things - internet of things companies. http://www.gartner.com/it-glossary/internet-of-things/. (Accessed on 19-06-2016).

[15] An introduction to the elk stack (now the elastic stack) | elastic. https://www.elastic.co/webinars/introduction-elk-stack. (Accessed on 19-08-2016).

[16] An iot platforms match : Microsoft azure iot vs amazon aws iot | devexperience. https://paolopatierno.wordpress.com/2015/10/13/an-iot-platforms-match-microsoft-azure-iot-vs-amazon-aws-iot/. (Accessed on 20-06-2016).

[17] Iot—the industrial way | iot content from electronic design. http://electronicdesign.com/iot/iot-industrial-way. (Accessed on 20-06-2016).

[18] javascript - angularjs: Understanding design pattern - stack overflow. http://stackoverflow.com/questions/20286917/angularjs-understanding-design-pattern/30745329#30745329. (Accessed on 17-06-2016).

[19] Jeffrey friedl's blog » simple json encode/decode in pure lua. http://regex.info/blog/lua/json. (Accessed on 22-06-2016).

[20] Jsonlab: a toolbox to encode/decode json files - file exchange - matlab central. https://www.mathworks.com/matlabcentral/fileexchange/33381-jsonlab--a-toolbox-to-encode-decode-json-files. (Accessed on 22-06-2016).

[21] Luup intro - micasaverde. http://wiki.micasaverde.com/index.php/Luup_Intro. (Accessed on 21-06-2016).

[22] Microsoft word - ieee_iot_towards_definition_internet_of_things_revision1_27may15.docx. http://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf. (Accessed on 18-06-2016).

[23] Nagios - the industry standard in it infrastructure monitoring. https://www.nagios.org/. (Accessed on 20-08-2016).

[24] openhab. http://www.openhab.org/. (Accessed on 19-08-2016).

[25] robindukewoolley010713.pdf. http://www.cambridgewireless.co.uk/presentation/robindukewoolley010713.pdf. (Accessed on 20-06-2016).

[26] Simple steps to learn iot architecture. https://www.linkedin.com/pulse/simple-steps-learn-iot-architecture-brucelin-kithion. (Accessed on 20-06-2016).

[27] Stabilita, kvalita a presnosť regulácie | www.senzorika.leteckafakulta.sk. http://www.senzorika.leteckafakulta.sk/?q=node/298. (Accessed on 15-06-2016).

[28] Vera3 advanced smart home controller vera. http://getvera.com/controllers/vera3/. (Accessed on 21-06-2016).

[29] What is big data? - gartner it glossary - big data. http://www.gartner.com/it-glossary/big-data/. (Accessed on 20-06-2016).

[30] What is enterprise architecture (ea)? - definition from whatis.com. http://searchcio.techtarget.com/definition/enterprise-architecture. (Accessed on 18-06-2016).

[31] What is erp (enterprise resource planning)? - definition from whatis.com. http://searchsap.techtarget.com/definition/ERP. (Accessed on 18-06-2016).

[32] What is the internet of things (iot)? - definition from techopedia. https://www.techopedia.com/definition/28247/internet-of-things-iot. (Accessed on 19-06-2016).

[33] What is the tick stack? | influxdata. https://influxdata.com/get-started/what-is-the-tick-stack/. (Accessed on 19-08-2016).

[34] Where can i find the best description of the iot architecture and development flow? - quora. https://www.quora.com/Where-can-I-find-the-best-description-of-the-IoT-architecture-and-development- (Accessed on 20-06-2016).

[35] Balandat, M. Constrained Robust Optimal Trajectory Tracking: Model Predictive Control Approaches. http://hybrid.eecs.berkeley.edu/~max/pubs/pdf/Diploma_Thesis.pdf. Accessed on 12-02-2014).

[36] Bemporad, A., and Morari, M. Robust Model Predictive Control: A Survey, 1999.

[37] Bradley, A., Bennick, A., and Salciccioli, M. Model Predictive Control. https://controls.engin.umich.edu/wiki/index.php/MPC/. Accessed on 12-02-2014).

[38] Řehoř, J. Návrh explicitního prediktivního regulátoru s využitím Multi-Parametric Toolboxu pro Matlab. http://www2.humusoft.cz/www/papers/tcp07/rehor.pdf. Accessed on 12-02-2014).

[39] Grimble, M. J. Robust industrial control systems: Optimal design approach for polynomial systems, 2006.

[40] Kozák, ., and Dúbravský, J. On-line predictive controller for gas turbine, 2013.

[41] Richards, M. Software architecture patterns, 2015.

[42] Takács, G., and Rohaľ-Ilkiv, B. Model Predictive Vibration Control, 2012.

Prílohy

A Vyvinuté zdrojové kódy

Zoznam vyvinutých komponentov s odkazom na ich zdrojové kódy:

- IoT zariadenia
  - Vystavenie hodnoty fotorezistora: `https://github.com/anton-pytel/rest-gpio`
- IoT brána
  - Čítanie údajov z meteostanice: `https://github.com/anton-pytel/vera-rss`
  - Čítanie údajov z vystaveného fotorezistora: `https://github.com/anton-pytel/vera-rest`
  - Akčný člen prediktívneho regulátora: `https://github.com/anton-pytel/vera-mpc`
  - Správca viacerých prediktívnych regulátorov: `https://github.com/anton-pytel/vera-mpcmaster`
- IoT backend
  - Súčasti serverovej časť prediktvíneho regulátora: `https://github.com/anton-pytel/caas`
  - Ladiaci nástroj na úpravu jadra GSON funkcionality: `https://github.com/anton-pytel/javadebug`

B Nainštalované súčasti

Zoznam nainštalovaných súčastí:

- IoT zariadenia
  - Operačný systém Raspbian pre Raspberry Pi zariadenia: `https://www.raspberrypi.org/downloads/raspbian/`
  - Systém na komunikáciu s meteostanicou: `http://www.wviewweather.com/`
  - Systém na monitorovanie zariadení Nagios: `https://www.nagios.com/downloads/nagios-xi/`
  - Mail server súčasti na odosielanie notifikačných e-mailov.
    Postfix: `http://www.postfix.org/download.html`
    Dovecot: `http://www.dovecot.org/download.html`
  - Node.js framework na vystavenie hodnôt REST princípom: `https://nodejs.org/en/download/`
  - npm balíky pre komunikáciu s GPIO.
    `https://www.npmjs.com/package/rpio`
    `https://www.npmjs.com/package/pcf8591`
  -
- IoT brána
  - Testovanie brány Razberry: `http://razberry.z-wave.me/index.php?id=24`
  - Nástroj DataMine na zaznamenávanie hodnôt vo VeraLite: `http://code.mios.com/trac/mios_datamine`
  - Nástroj PLEG na pokročilú automatizáciu vo VeraLite: `http://rts-services.com/Vera/Plugin/PLEG/`
- IoT backend
  - Nástroje na výpočty.
    Matlab: `http://www.mathworks.com/downloads/`
    Octave: `https://www.gnu.org/software/octave/download.html`
  - Ukladanie regulátorov - MongoDB: `https://www.mongodb.com/download-center#community`

– Aplikačný server Tomcat8 na spustenie Java aplikácie: `https://tomcat.apache.org/download-80.cgi`
– InfluxDB na zbieranie údajov o časových radoch: `https://influxdata.com/downloads/`
– Vizualizácia údajov časových radov Grafana: `http://grafana.org/download/`