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

Písomná práca k dizertačnej skúške je zameraná v prvom rade na odbor automatizácia, konkrétne na opis vybraných metód a algoritmov prediktívneho riadenia lineárnych dynamických systémov. Na základe teoretických princípov metód prediktívneho riadenia z domácej a zahraničnej literatúry bol v priebehu prvej časti doktorandského štúdia vyvinutý všeobecný podporný programový systém, pre účely výučby a výskumu v predmetnej oblasti. Podporný programový systém je vytvorený v prostredí Matlab-Guide a umožňuje testovať, ladiť a konfigurovať rôzne prediktívne algoritmy riadenia. Všeobecný programový systém bol testovaný pre rôzne typy priemyselných procesov (tepelné-optické procesy, DC motory, riadenie turbíny apod.).

V druhom rade sa zameriava na odbor informatika. Práca rozoberá možnosti architektúr softvéru a ich prienik s odborom automatizácia. S využitím znalostí z oboch odborov je popísaný typ architektúry nazývaný Internet vecí (Internet of Things - IoT). Sú taktiež definované jeho vlastnosti, prípady použitia a ich variácie. V tejto časti sú analyzované aj rozdiely a zhody v návrhu, vývoji a spravovaní existujúcich aplikácií voči IoT systémom.

V poslednom rade je v práci popísaný konkrétny IoT systém inteligentnej budovy a všetky jej súčasti. Myšlienka regulátor ako služba (Controller as a service) a jej implementácia v uvedenom IoT systéme. Konkrétne ide o implementáciu online prediktívneho regulátora.

Kľúčové slová: TODO: kľúčové slová, MPC, REST, IoT

Obsah

Zoznam skratiek a značiek

CRM - Customer Relationship Management - aplikácia na správu vzťahov so zákazníkom
ERP - Enterprise Resource Planning - plánovanie zdrojov v podniku
ESB - Enterprise Service Bus - servisná zbernica podniku
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
UPnP - Universal Plug aNd Play
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

Úvod

V súčasnosti existuje mnoho metód automatického riadenia systémov. Od klasických metód ako je PID regulátor, cez rôzne pokročilé metódy ako je robustné, adaptívne alebo aj prediktívne riadenie. Predmetom tejto práce je

- spomenúť hlavné výhody prediktívnej metódy a jej súčasný stav.
- Predmetom práce je tiež definovať pojem Internet vecí (Internet of Things - IoT) ako nový typ architektúry informačných systémov a jeho špecifiká.
- V praktickej časti je spojenie predošlých dvoch bodov v myšlienke regulátor ako služba (Controller as a Service - CaaS) a jej implementácia.

Motivácia k MPC

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.[38]

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 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 pre-

menných, 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.[33]

Motivácia k IoT

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: „Napriek tomu, že IoT bol vždy trochu vágny pojem pri špecifikácii obchodných partnerov a produktov, ktoré už sú na trhu, analytici predikovali, že pripojené zariadenia (connected devices) majú veľký potenciál príležitosti, ktoré budú výnosné. 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] Na základe tohto a ďalších podobných článkov je teda motivácia hľadanie vhodných prípadov využitia IoT.

Okrem toho treba zopakovať, že IoT je spojenie minimálne dvoch rozsiahlych technických odborov, neberúc do úvahy spoločenské, prírodne ani lekárske vedné odbory, ktoré tiež môžu skúmať dopad IoT na ne. Rozsiahlosť tejto problematiky je určite nepopierateľná. Druhá motivácia teda je získavanie nadhľadu nad spleťou vznikajúcich a zanikajúcich technológií a štandardov.

Ostatná motivácia k skúmaniu IoT je porovnanie rozdielov pri návrhu, vývoji a správe oproti klasickým čisto softvérovým informačným systémom.

Motivácia k CaaS

Napriek tomu, že aktuálny trend vo vývoji hardvéru je znižovanie rozmerov a zvyšovanie výkonu, online prediktívny regulátor je stále náročný na výpočtový výkon. Preto popri výskumoch aplikovania offline metódy prediktívneho algoritmu na zariadenia blízke hardvérovej úrovni napríklad FPGA hradlá, vznikla myšlienka implementácie online MPC regulátora na server - „do cloudu“, ako službu, kde je možné zabezpečiť takmer neobmedzený výkon a jedinou prekážkou môže byť rýchlosť sieťového pripojenia. Realizácia tejto myšlienky je implementovaná v prostredí inteligentnej budovy.

1 Prediktívne metódy riadenia

Ako už bolo v úvode spomenuté, práca spája viacero odborov do jednej implementácie. Táto časť preto je venovaná návrhu a overenia MPC regulátora.

1.1 Teoretický základ MPC

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

1.1.1 On-line MPC

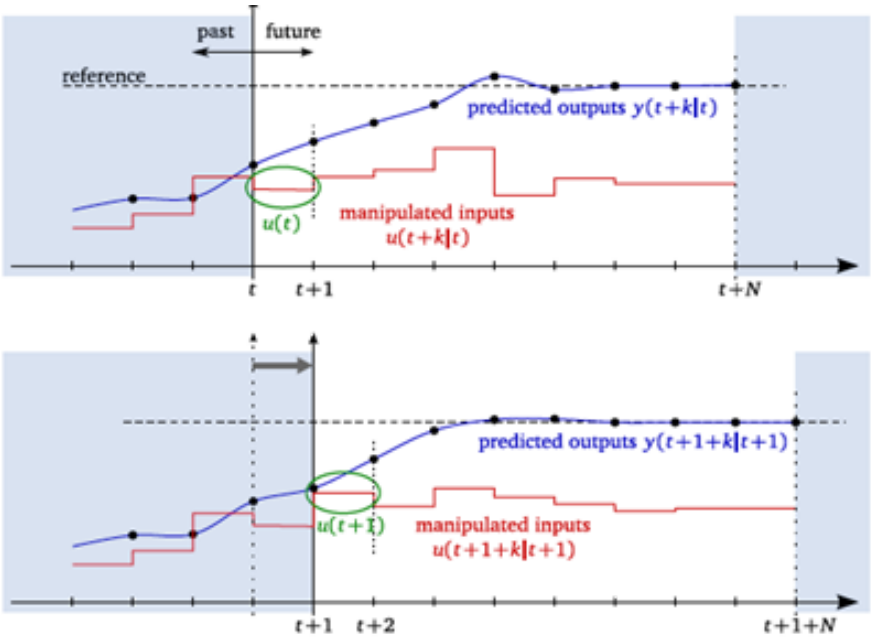
Prediktívny regulátor je, na najnižšej úrovni, metóda riadenia dynamických systémov, ktorá využíva nástroje matematickej optimalizácie. Spoločné črty všetkých prístupov riešenia problému prediktívnych regulátorov je vypočítať on-line, v každej časovej vzorke, optimálny riadiaci zásah v konečnom horizonte 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“. [31] Popisovaný princíp je znázornený na obrázku 1. Nech existuje lineárny dynamický model s časovo invariantnými parametrami, vyjadrený diskrétnou stavovou rovnicou:

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

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



Obrázok 1: Postupujúci horizont riadenia

m – predstavuje počet vstupov

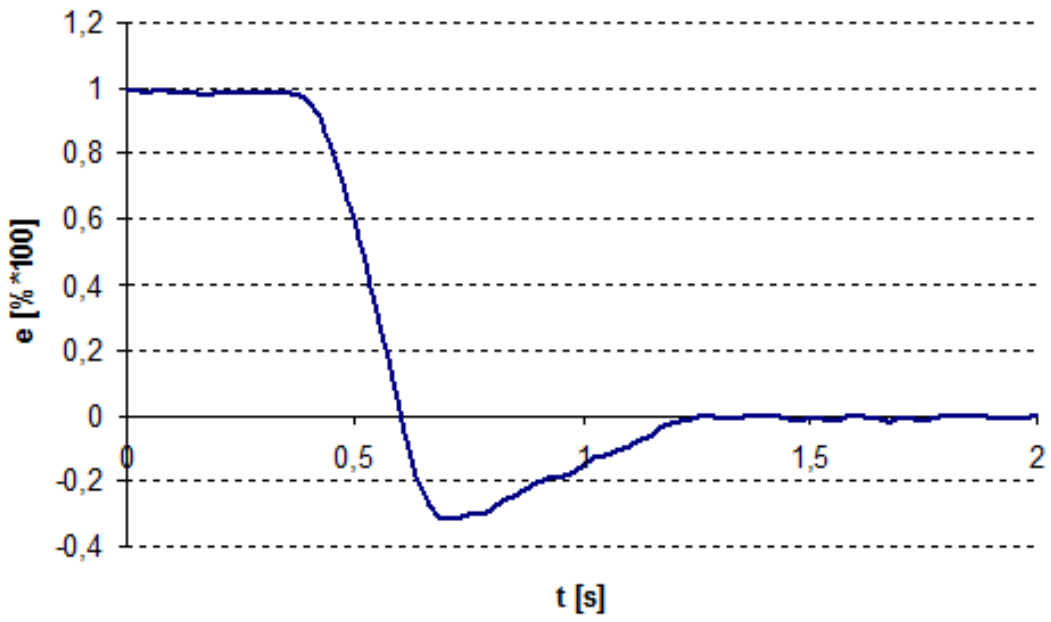
Matice A, B, C sú systémová matica, vstupná matica a výstupná matica v uvedenom poradí. System 1 musí spĺňať nasledujúce obmedzenia na stav a vstup:

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

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.[32] 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

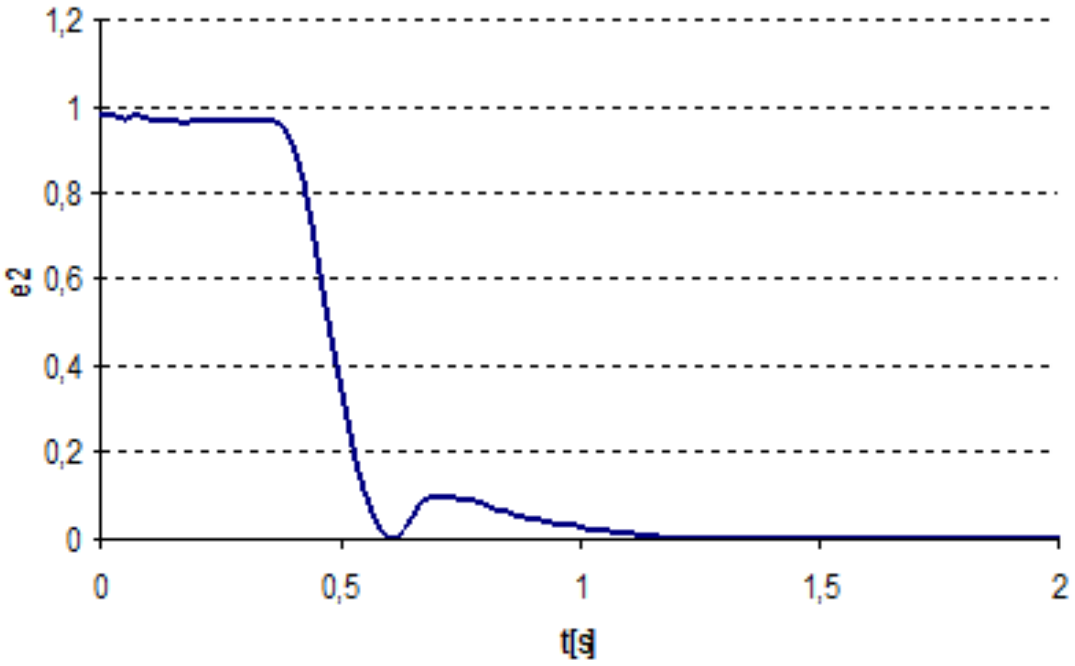
odvoditeľná z predošlého stavu a prvý stav x_0 je známy. Preto jedinou „neznámou“ ostáva sekvencia riadiacich zásahov. Jedna z otázok by mohla byť, prečo sa využíva kvadratické kritérium. Ako odpoveď je možné použiť príklad kvality riadenia. Pri hodnotení kvality riadenia sa používajú integrálne kritéria. 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.
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,



Obrázok 3: Časová závislosť kvadrátu regulačnej odchýlky

- pre vzorku k = 1 x(t+1)=$x_{(t+1)}$,
- atď.

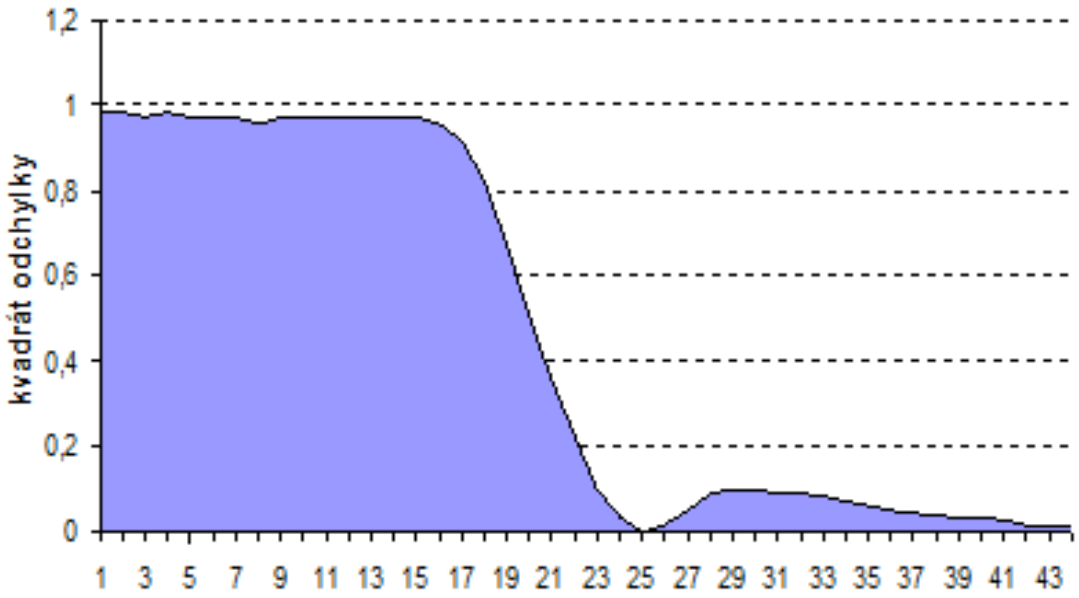
Kvadratické kritérium pre konečný horizont predikcie dĺžky N je:

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

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



Obrázok 4: Plocha kvadrátu regulačnej odchýlky

v tomto prípade aj analytické riešenie: [34]

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

Vyjadrenie predikovaného stavu je potom:

$$[x_{t+1}^T, \ldots, x_{t+N}^T]^T = V x_0 + T u_{t,N} \tag{6}$$

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:

$$V = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \\ 2 & 1 \\ 1 & 0 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} A^1 \\ A^2 \\ A^3 \end{bmatrix}$$

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

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 $\bigotimes$ označí ako kroneckerovo násobenie matíc a $I_j \in R^{(j \times j)}$, jednotková matica s príslušnou dimenziou, tak platí:

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

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:

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

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

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

kde:

$$\begin{aligned} x_{k+1}&=Ax_k+Bu_k,\\ x_0&=x\left(t\right),\\ E_cx_k+G_cu_k&\leq m_c,\quad k=1\ldots N\\ Q=Q^T\succcurlyeq 0,\ Q_N=Q_N^T\succcurlyeq 0,\ R&=R^T\succ 0 \end{aligned} \tag{17}$$

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:

$$\begin{aligned} J\left(x\left(t\right),U_t\right)=\ \frac{1}{2}{u_{t,N}}^THu_{t,N}+x^T\left(t\right)FU_t+\frac{1}{2}x^T\left(t\right)\mathrm{Yx}\left(t\right),\\ Gu_{t,N}\leq w+Ex(t) \end{aligned} \tag{18}$$

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

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

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:

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

matice G, E a vektor w budú vyzerajú:

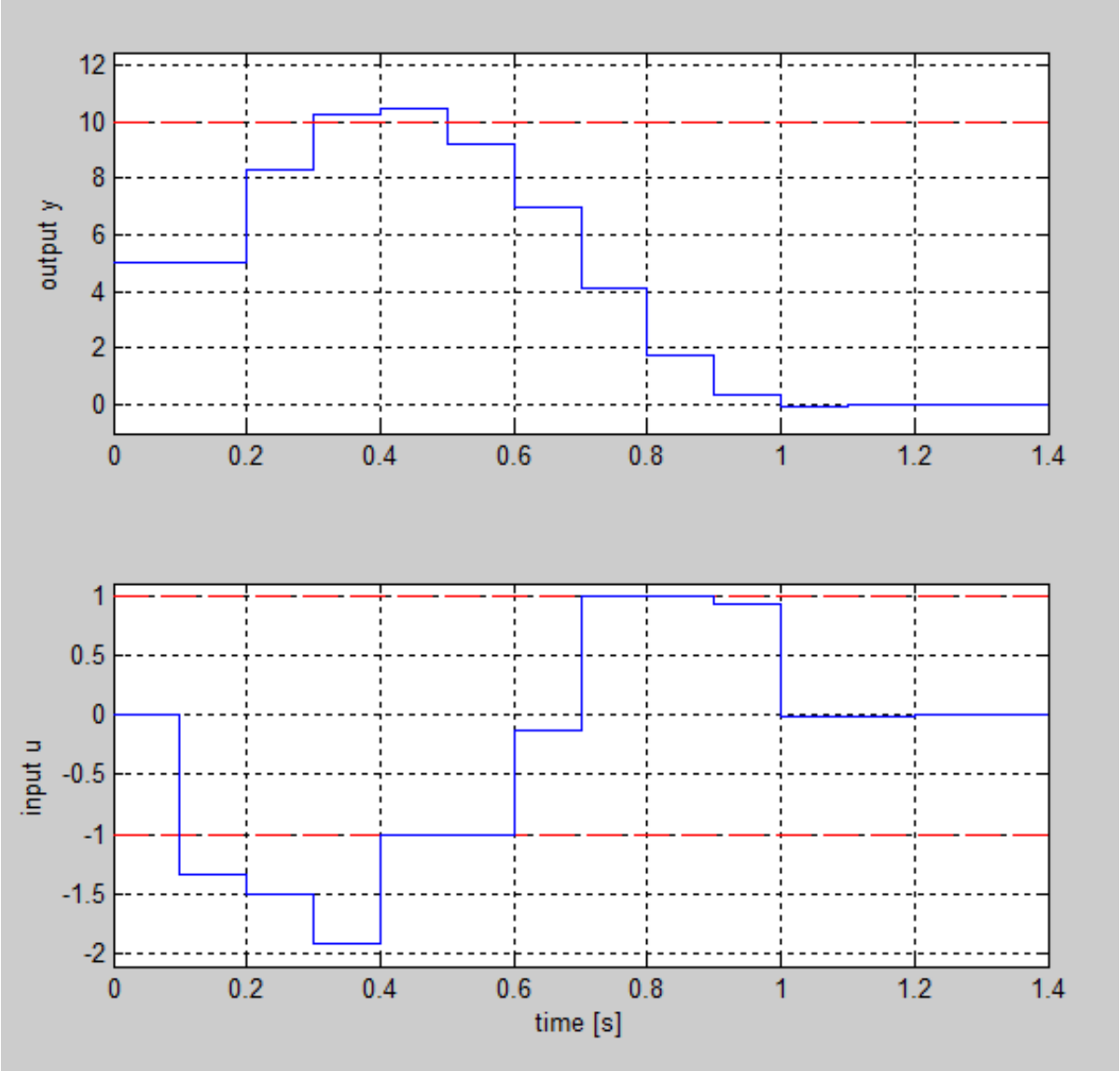
$$G = \begin{bmatrix} -I_3 \\ I_3 \\ -(I_3 \bigotimes CT) \\ (I_3 \bigotimes CT) \end{bmatrix}, \; E = \begin{bmatrix} 0 \\ 0 \\ -(I_3 \bigotimes CV) \\ (I_3 \bigotimes CV) \\ \vdots \end{bmatrix}, \; m_c = \begin{bmatrix} -(1_3 \bigotimes u_{\min}) \\ (1_3 \bigotimes u_{\max}) \\ -(1_3 \bigotimes y_{\min})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

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

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.

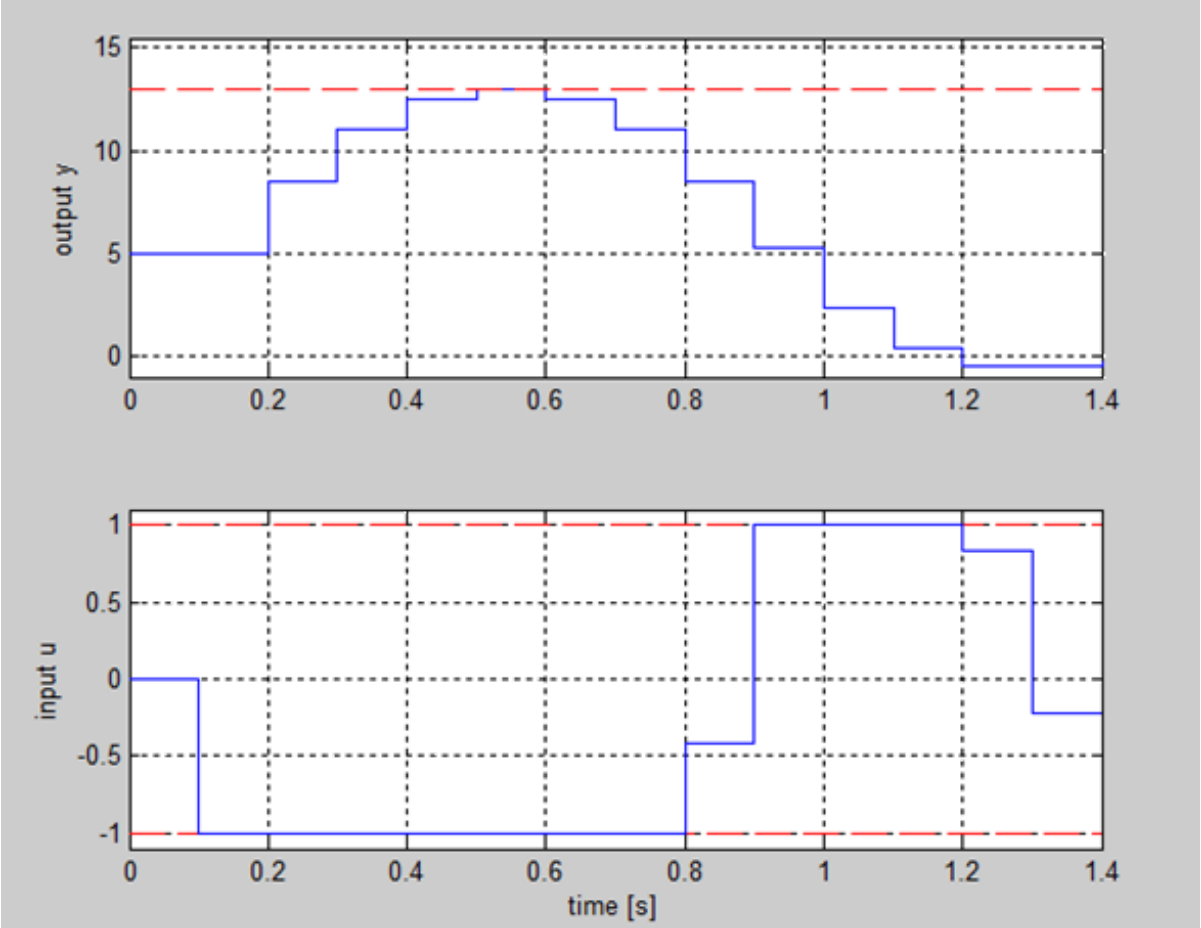


Obrázok 5: Časový priebeh výstupnej veličiny a riadiaceho zásahu pre neriešiteľný kvadratický problém. Červená čiara predstavuje obmedzenie na danú veličinu

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



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

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

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

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).

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. [34]

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

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

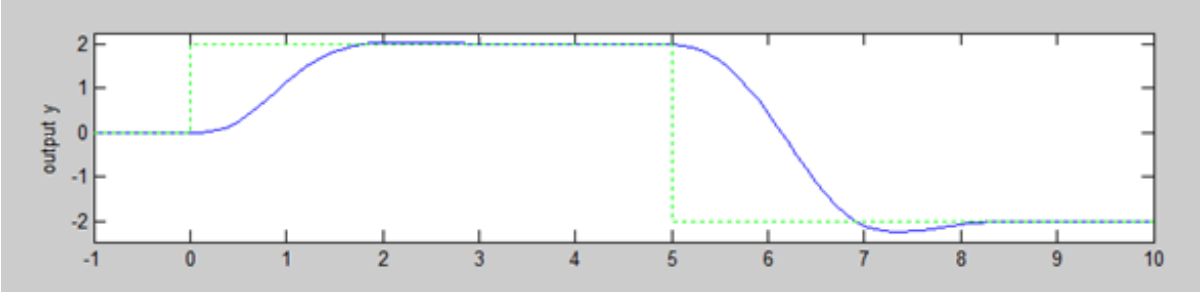
je regulačná odchýlka a

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

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 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:

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

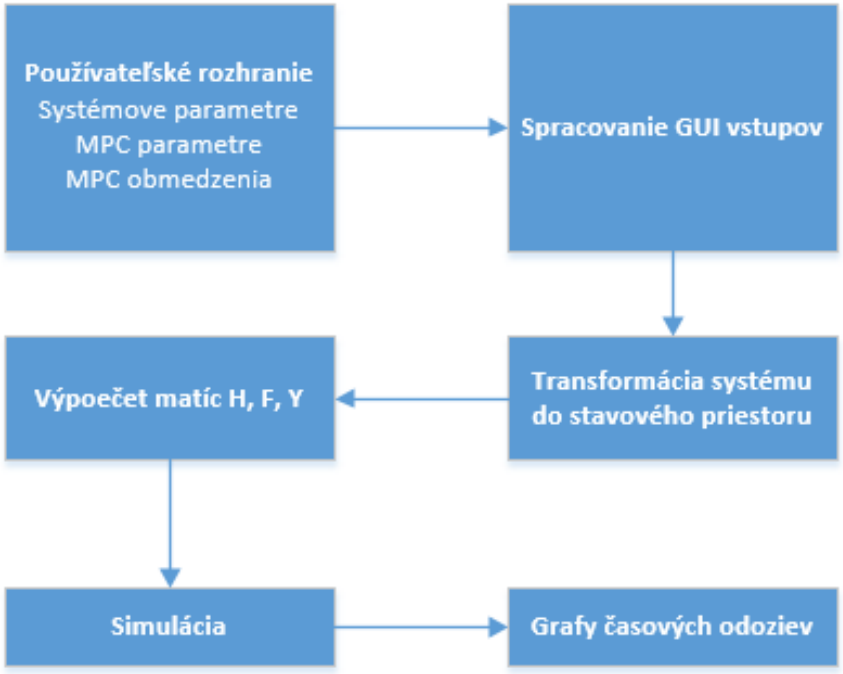
a spojitá kvadratická funkcia

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

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 . [34]
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. Programbol 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 a skladá sú znázornené na obrázku 8.



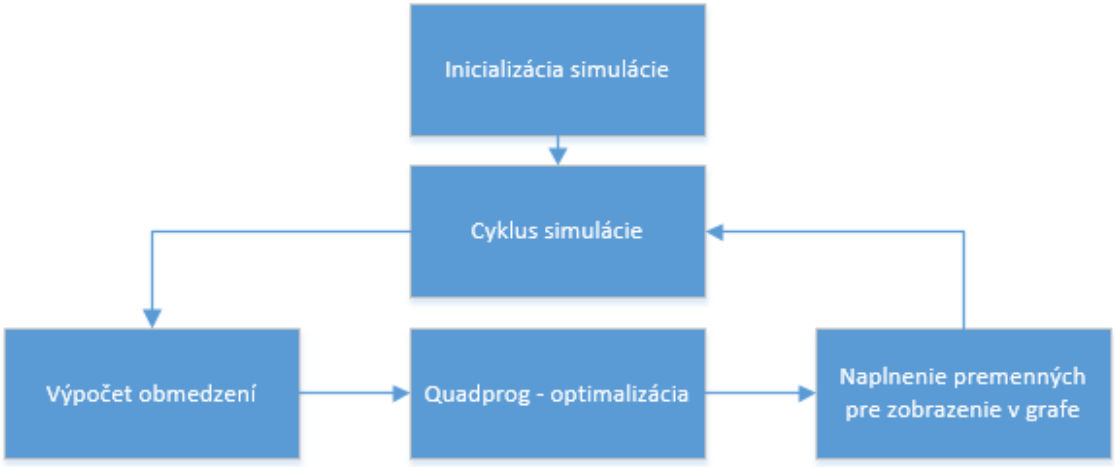
Obrázok 8: Základné komponenty

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.

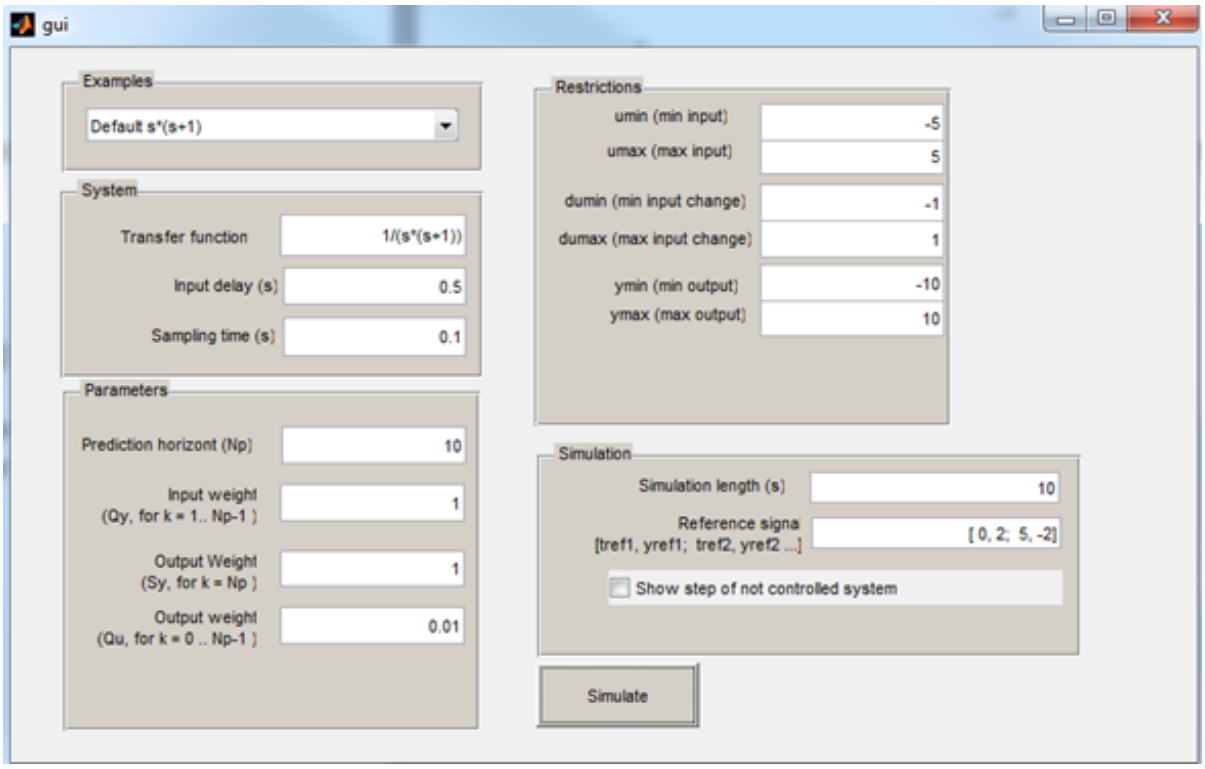
3. V prvej fáze simulačného cyklu sa prepočítajú obmedzenia systému podľa rovnice 19.

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

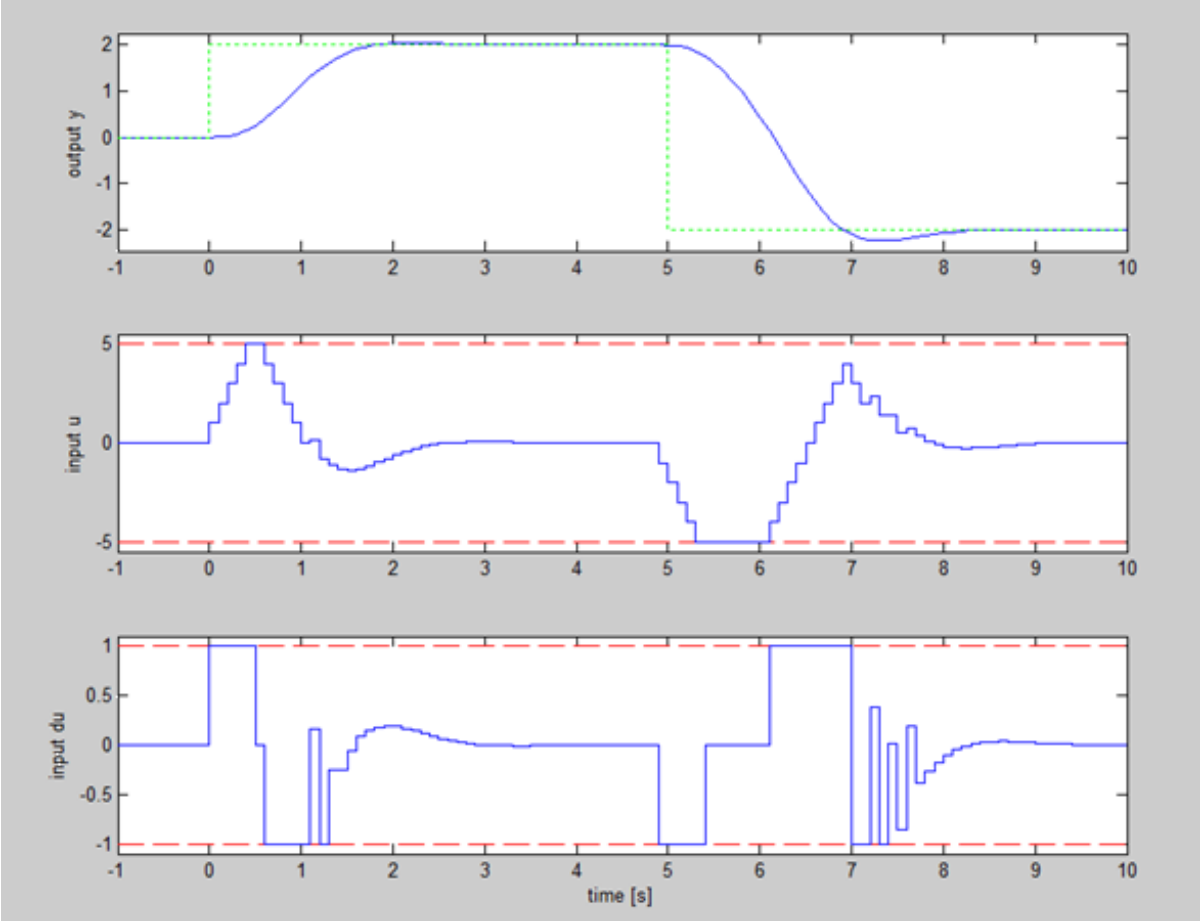
- Parametre simulácie
  - čas simulácie
  - referenčný signál zadávaný vo forme vektor dvojice hodnôt, kde prvá identifikuje čas, kedy ma zmena nastať a druhá hodnota veľkosti skoku.

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 [24]

- 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.



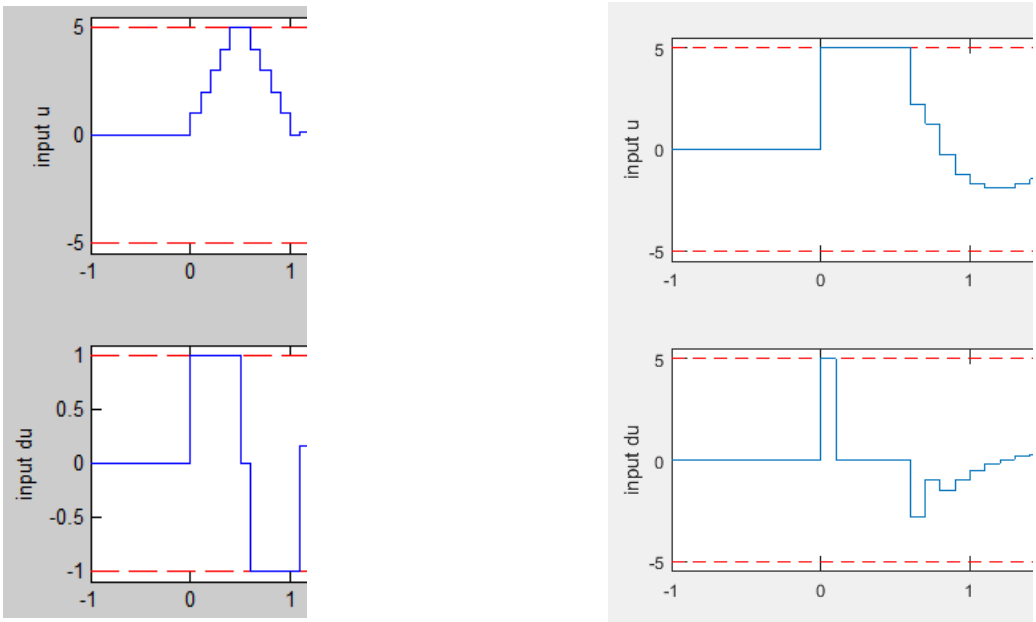
Obrázok 11: Časové priebehy výstupu systému, riadiaceho zásahu a zmeny riadiaceho zázsahu.

- 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 si na tomto mieste zdôrazniť je znázornenie ako vplývajú obmedzenia na riadenie systému.

- **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.



(a) Zapnuté obmedzenie

(b) Vypnuté obmedzenie.

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

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 sa obmedzenie akčného zásahu musí špeciálne riešiť.

Ďalší nástroj na overenie MPC algoritmu je voľne stiahnuteľný a použiteľný v prostredí Matlab. Názov je MPT toolbox. Je to dielo inštitútu pre automatizáciu vo Švajčiarsku. Tento program je jeden z najpoužívanejších v prostredí Matlab. Nasleduje overenie algoritmu na reálnom systéme z praxe, ktorý je popísaný v nasledujúcej podkapitole. Následne je identifikovaný matematický model systému a overenie funkčnosti MPC princípov. Treba spomenúť, že overenie je stále prostredníctvom simulácie v prostredí Matlab.

1.3.1 Opis systému riadenia plynovej turbíny

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.[35]

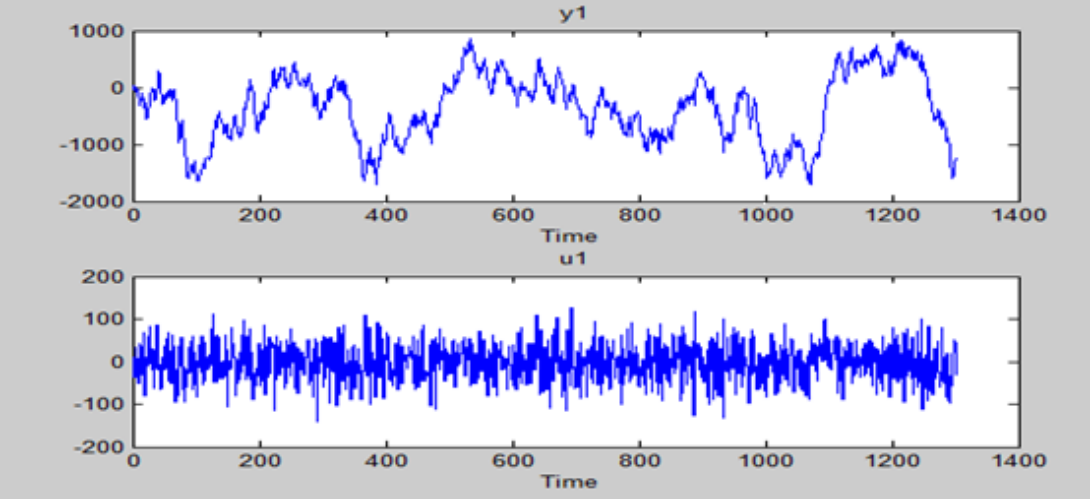
1.3.2 Identifikácia a riadenie systému plynovej turbíny

Vstupné dáta na identifikáciu systému sú zobrazené na obrázku 13.

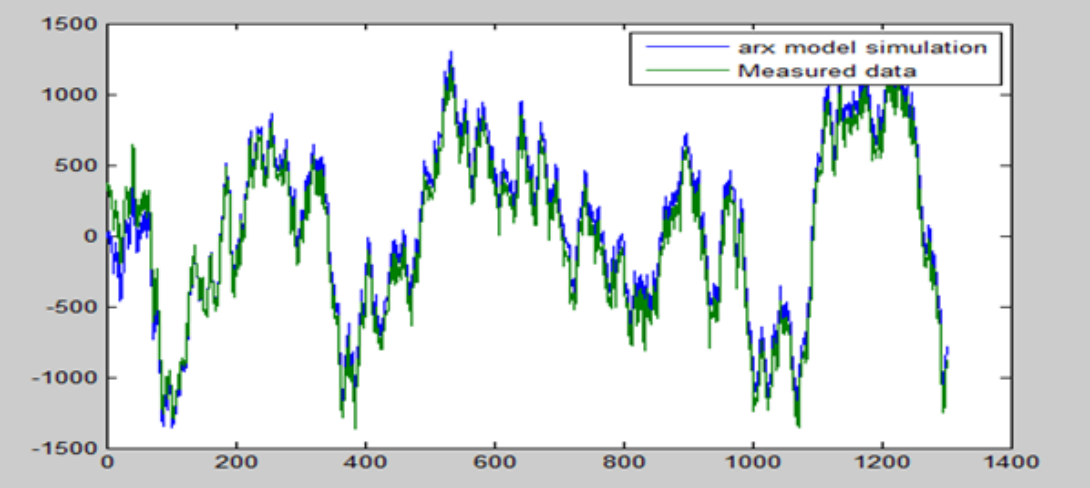
Premenná y_1 reprezentuje výstup - otáčky motora a u_1 vstup systému - natočenie klapky. Na identifikáciu systému bol použitý Matlab nástroj *ident*. Pri zisťovaní matematického modelu bolo vyskúšaných viacero metód. Najlepšie výsledky - percento zhody nameraných a simulovaných dát dal arx model. Porovnanie nameraných a simulovaných dát sú na obrázku 14.

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

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



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



Obrázok 14: Porovnanie simulovaných a nameraných údajov

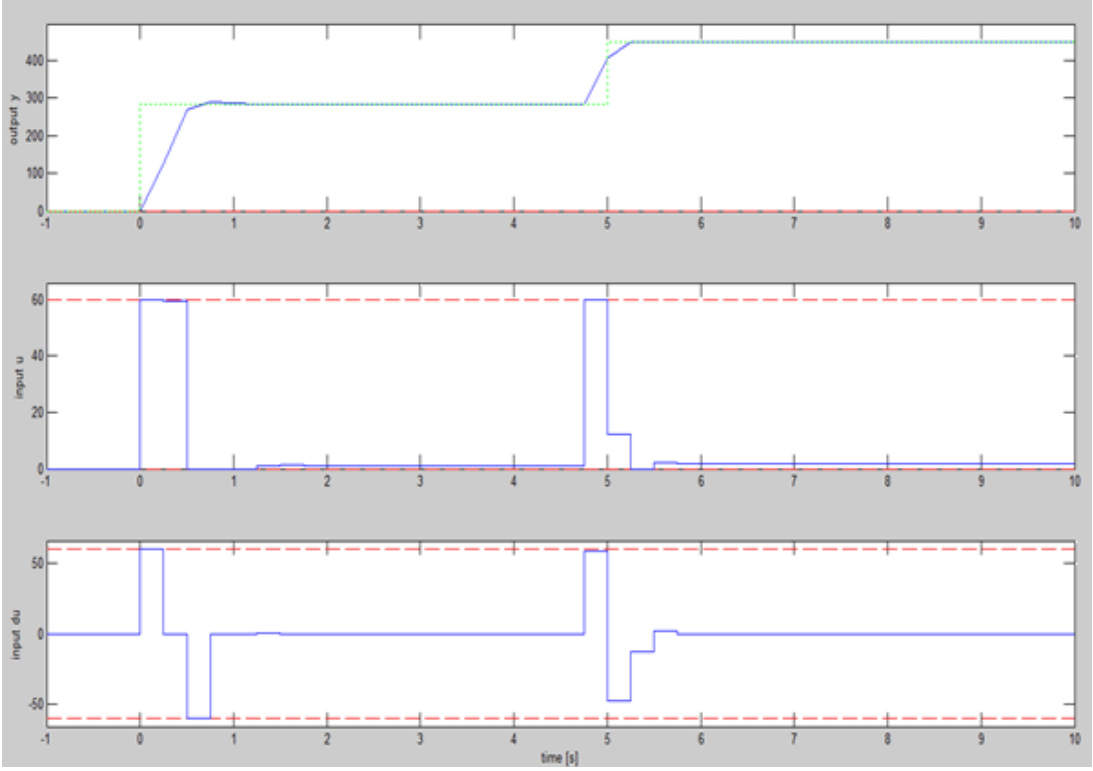
kde

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

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. Ze-

lená 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

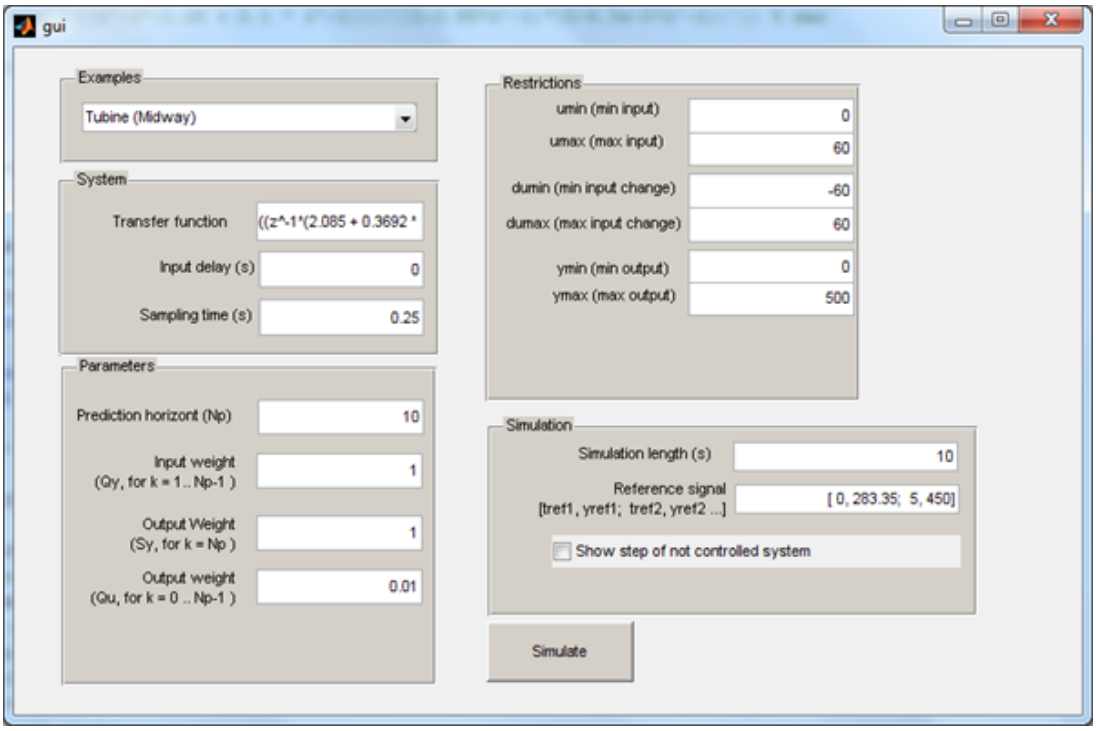
renosová funkcia 39 sa zadala do grafického rozhrania do pola „transfer function“. Horizont predikcie bol nastavený na 10 vzoriek dopredu. Zo simulácii vyplýva, že je to dostatočné a efektívne, berúc do úvahy odozvu systému a kvalitu riadenia. Váhy boli nastavené na prednastavené hodnoty. Obmedzenia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

- 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.[36]

Týmto končí simulačné overovanie algoritmu MPC regulátora vytvoreného na základe



Obrázok 16: Parametre simulácie

teoretických princípov popísaných v úvodných kapitolách MPC regulátora a nasleduje druhá časť práce, ktorá pripraví podklady pre praktický experiment, ktorý zahŕňa definíciu prostredia IoT.

2 Softvérové Architektúry

Téma softvérovej architektúry (Software architectrue) zahŕňa veľa súčastí. Práca sa zameriava len na tie časti, ktoré vedú do problematiky internetu vecí a teda nepojednáva o všetkých aspektoch tejto problematiky. Nasleduje niekoľko definícií a vymenovanie typov aplikácii.

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:

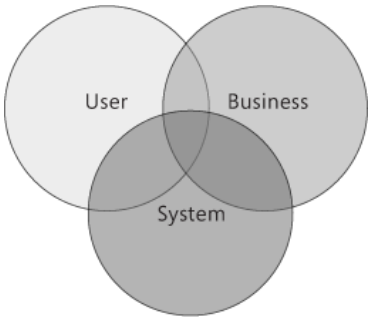
Softvérová architektúra zahŕňa sadu dôležitých rozhodnutí o usporiadaní softvérového systému vrátane voľby stavebných prvkov a ich rozhraní, pomocou ktorých je systém zložený. Rozhodnutie o voľbe správania, ktoré je definované spoluprácou medzi uvedenými prvkami. Rozhodnutie o spájaní týchto štrukturálnych 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.

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]

2.1 Základne typy architektúr

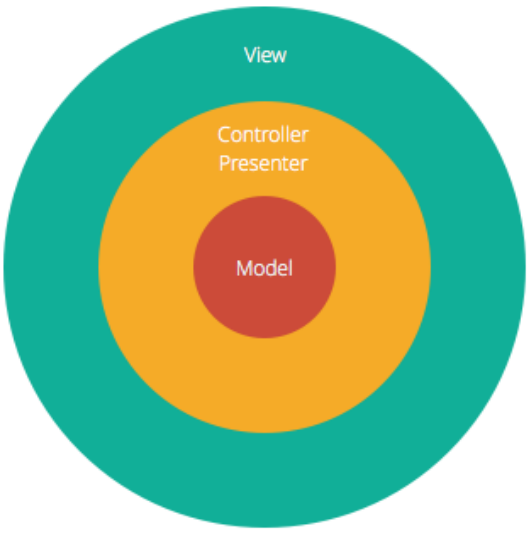
Jedno z kľúčových rozhodnutí je voľba typu architektúry. Tie najpoužívanejšie sú vymenované nižšie podľa článku [11] napísanom na základe knihy[37], ktorú napísal Mark



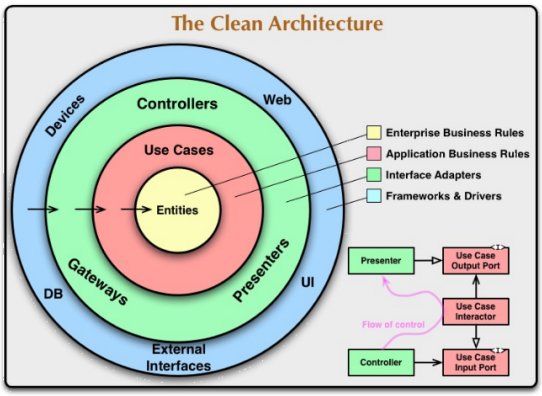
Obrázok 17: Oblasti, potrebné zvažovať pri návrhu[2]

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 [17]



(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

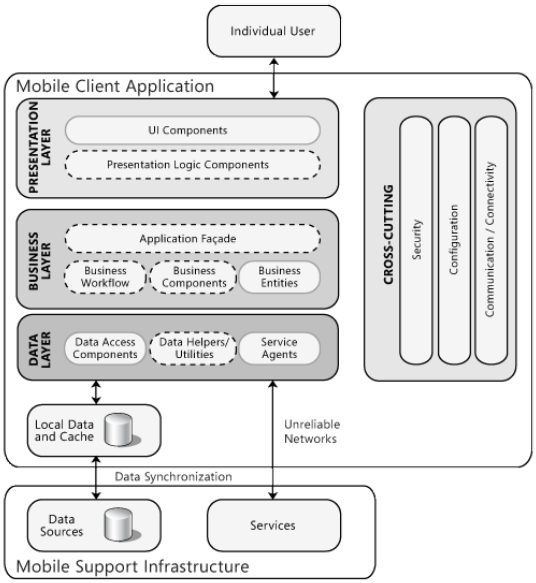
Ďalšie z kľúčových rozhodnutí je voľba typu aplikácie. Na výber podľa článku [3] sú:

- **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.

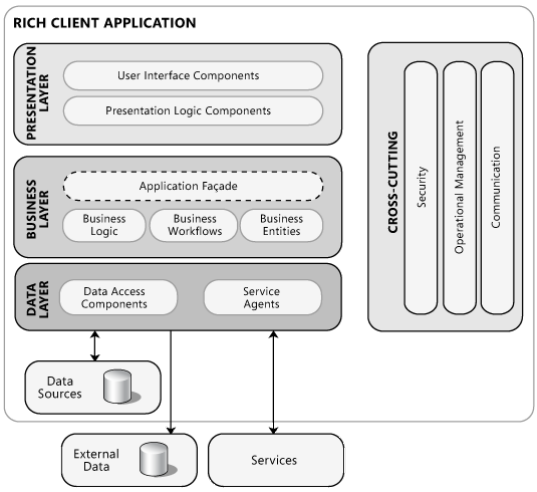
  Na zváženie:
  - Limity ohľadne zadávania údajov a navigácie v aplikácii.
  - Limitovaná šírka zobrazovanej plochy.

- **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 inštalačných možnosti ako ClickOnce, Windows Installer a XCOPY je dostupných.
  - Problém s nasadzovaním nových verzií v čase.
  - Závislé od platformy.



(a) Mobilná aplikácia [3]



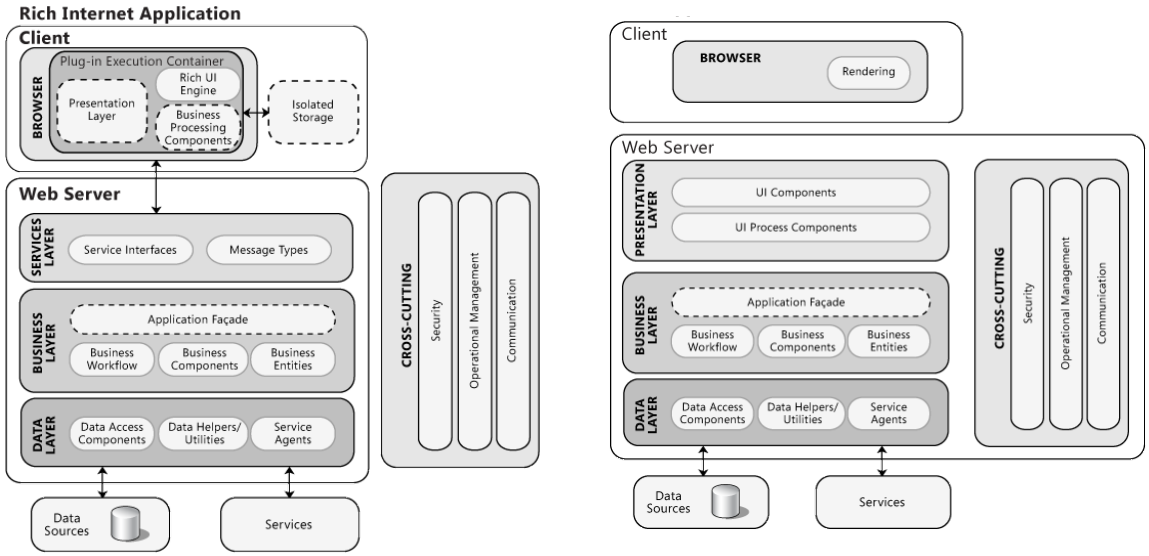
(b) Clientská aplikácia [3]

Obrázok 19

- **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 obmedzený 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.
- **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



(a) Tučná internet aplikácia [3]

(b) Web aplikácia [3]

Obrázok 20

väzieb medzi klientom a serverom.
Výhody:

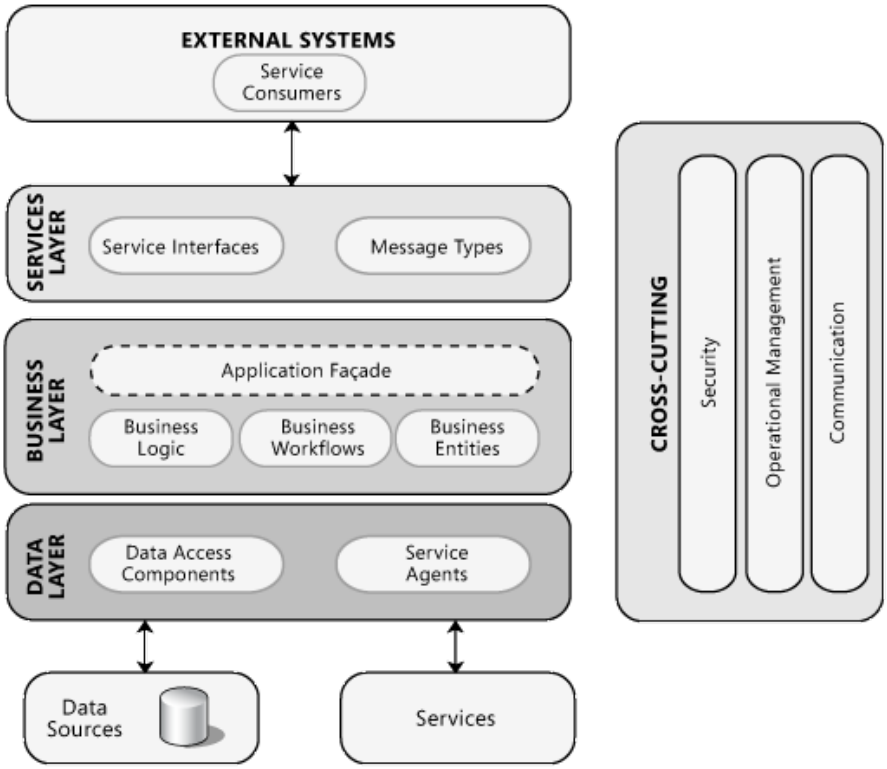
- Interakcie medzi klientom a serverom je prostredníctvom voľných väzieb.
- Môže byť použitá rôznymi nezávislými aplikáciami.
- Podpora pre interoperabilitu.

Na zváženie:

- Nie je podpora pomocou užívateľského rozhrania.
- Závisí od sieťového pripojenia.

2.2.1 Servisná aplikácia

Servisným aplikáciam je v práci venovaná samostatná kapitola, pretože v implementácii je použitý takýto typ aplikácie. 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 [27]. Inými slovami ide o architektúru podnikových informačných systémov, ktoré zvyčajne pozostávajú z viacerých súčasti. Príkladom týchto súčasti môže byť:



Obrázok 21: Aplikácia poskytujúca službu [3]

- **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

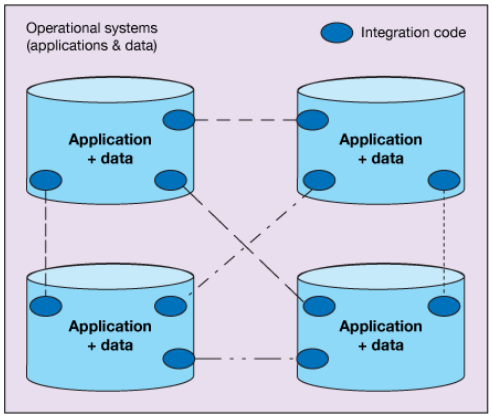
- **Účtovná aplikácia.** Aplikácia na vedenie účtovníctva

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

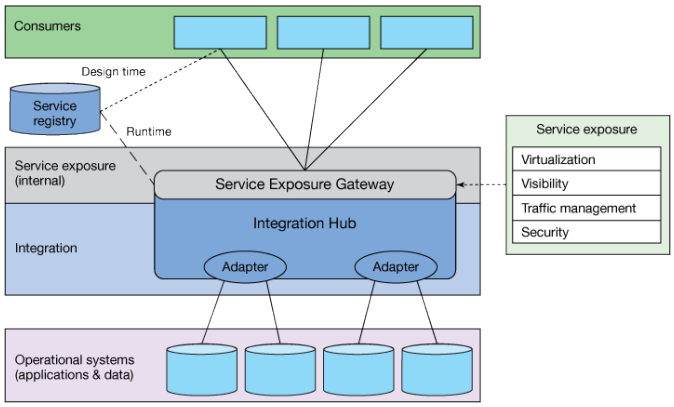
- **ERP aplikácia.** (Plánovanie zdrojov v podniku je pojem používaný v priemysle

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.[28]

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** s centrálnym bodom vystavovania služieb tzv. ESB - Enterprise service bus. Existuje viacero spôsobov ako SOA implementovať.

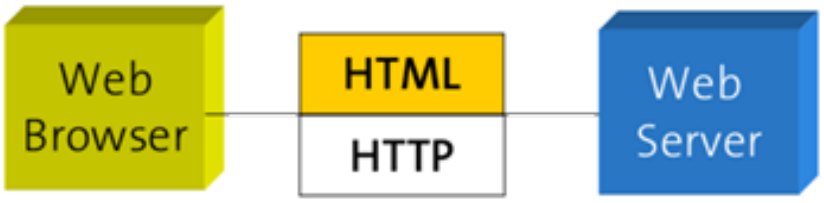


(a) Point to Point [13]



(b) SOA [13]

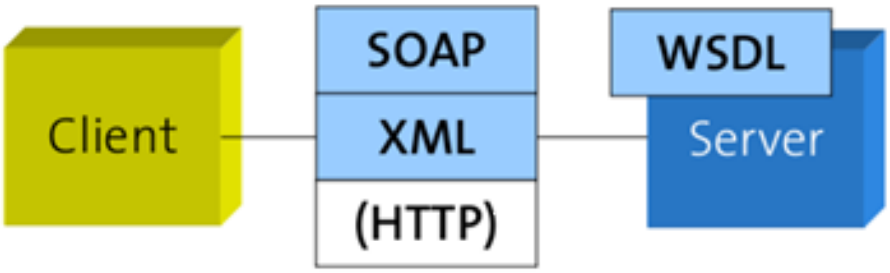
- Najznámejší spôsob implementácie SOA, ktorí dnes používa väčšina užívateľov informačných technológií je implementácia protokolu WWW - **World Wide Web** v skratke často označovaný len WEB. Obrázok 23 znázorňuje princíp fungovania. Prehliadač (web browser) je v pozícii klienta, konzumera služby, ktorú 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



Obrázok 23: WWW [12]

tomto prípade rozumie poskytnutie HTML dokumentu prostredníctvom aplikačného komunikačného protokolu HTTP.

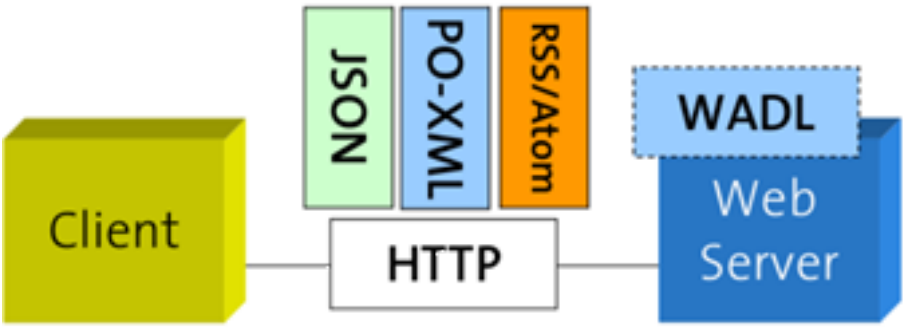
- Ďalší rozšírený protokol, ktorý implementuje SOA architektúru je **protokol SOAP**. Princíp fungovania je na obrázku 24. Klient je v tomto prípade generickejší. Zvyča-



Obrázok 24: SOAP [12]

jne 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 špecifickým 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.

- Ostatná 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. 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 je využívaný nie len ako



Obrázok 25: REST [12]

  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. Okrem využívania servisných aplikácii v enterprise architektúrach podnikov je tento typ vo veľkom využívaný aj v IoT prostredí. Preto sa pri implementácii zvolil takýto typ rozhrania.

2.3 Internet vecí - IoT

Organizácia IEEE vydala celý dokument s názvom „Smerom k definícii Internetu vecí“. Cieľom dokumentu je podať plnohodnotná 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 [21]. 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“ [29]. Viacero ďalších technologických spoločností majú ich definíciu. Cieľom tejto práce nie vytvoriť ďalšiu definíciu, ale identifikovať spoločné črty väčšiny definícií.

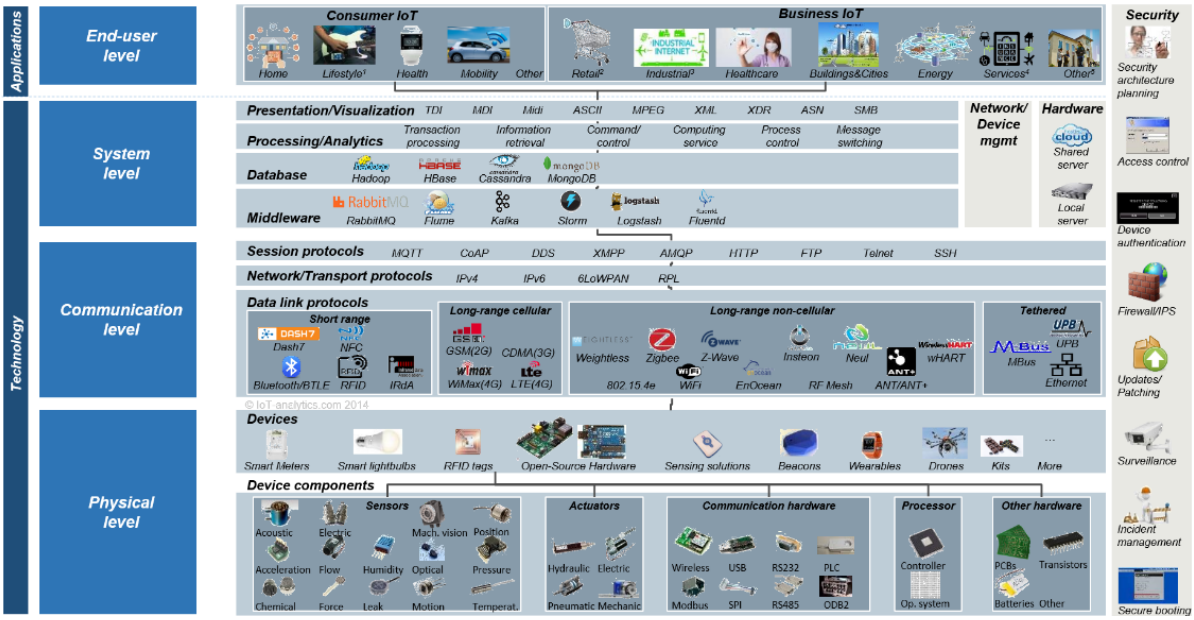
- Prvá základná črta definícii je, že v nej vystupujú objekty, ktoré môžu **snímať a ovplyvňovať okolie**. Na dosiahnutie tejto črty je potrebné, aby objekty mali senzory a akčné členy, čo je jeden z hlavných záujmov odboru automatizácia, pre ktorý to nie je nič nové. Dostupnosť a klesajúca cena snímačov 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, ktoré umožňujú zber a posielanie dát do dátových centier nazývaných - edge device, gateway alebo agregátor.

- 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

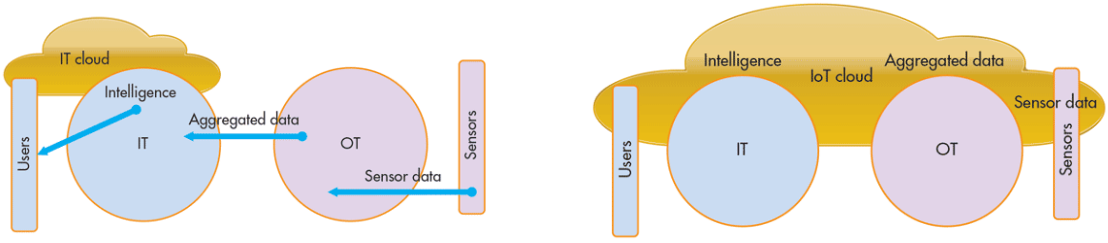
čase alebo kvázi reálnom čase, tak je spätne možnosť ovplyvniť akčný člen, čo vo veľkom môže znamenať vytvorenie inteligentného prostredia. Zabezpečenie tejto črty je predmetom oblasti ukladania a spracovania dát. V tomto bode sa IoT často spája s pojmom Big Data. Definícia tohto pojmu má viacero verzií 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 [26].

Zhrnutie technológii, ktoré môže IoT zhŕňať je ďalej na obrázku 26. V čom sa IoT líši od



Obrázok 26: IoT súhrn techológií [30]

tradičnej architektúry priemyselných systémov je znázornené na obrázkoch 27a a 27b.



(a) Tradičná architektúra [16]

(b) IoT architektúra [16]

Obrázok 27

Tradičná priemyselná architektúra jednoducho používa vstavaný softvér na platforme

obslužných technológií (Operational Technology - OT) na posunutie dát do platformy informačných technológií (Information Technology - IT), ktorá vie robiť, analýzu dát, poskytovať ich používateľovi aj vystavovať do cloud prostredia. IoT architektúra posúva väčšiu 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. [16]. Základné ciele IoT architektúr je 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. Veľké firmy typu Microsoft, Amazon, HP 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.

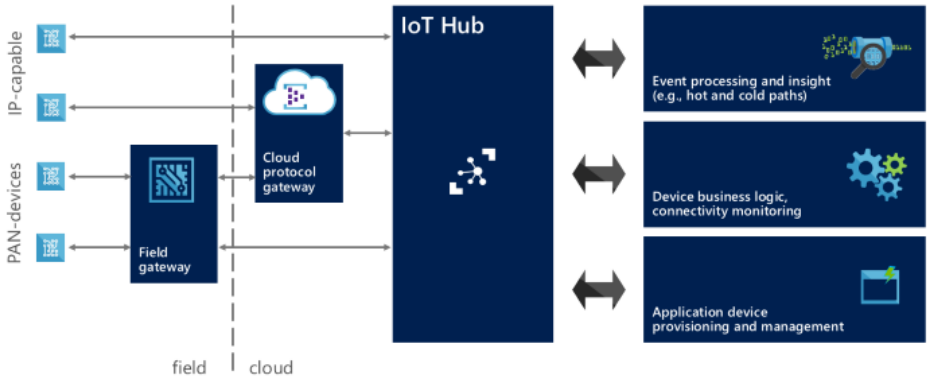
Tabuľka 1: Porovnanie dvoch IOT platforiem [15]

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

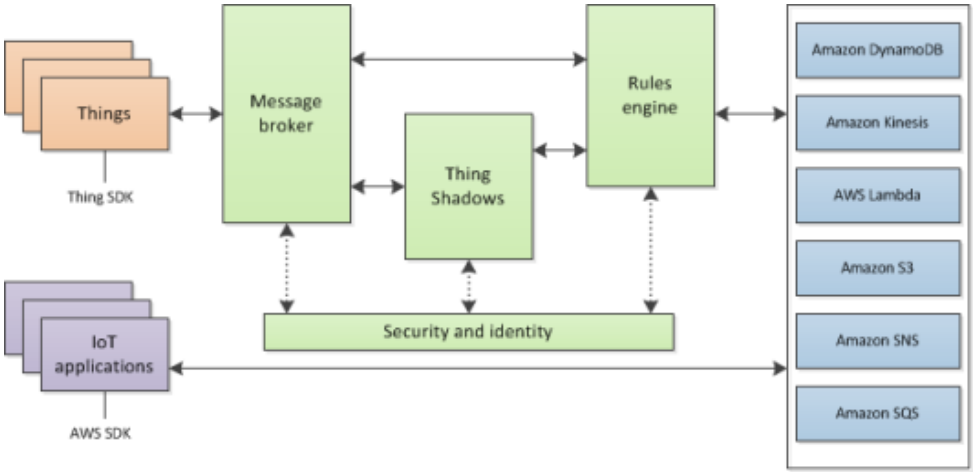
Na portály Devexperience a stránke [15], kde bolo porovnanie vykonané, definujú tieto súčasti IoT architektúry:

Kompletné IoT riešenie pozostáva z viacero častí. Ako prvé je potrebné prijať všetky udalosti a dáta poslané zo zariadení a to je tak 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



(a) Azure IOT HUB [15]



(b) AWS IOT [15]

Obrázok 28

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

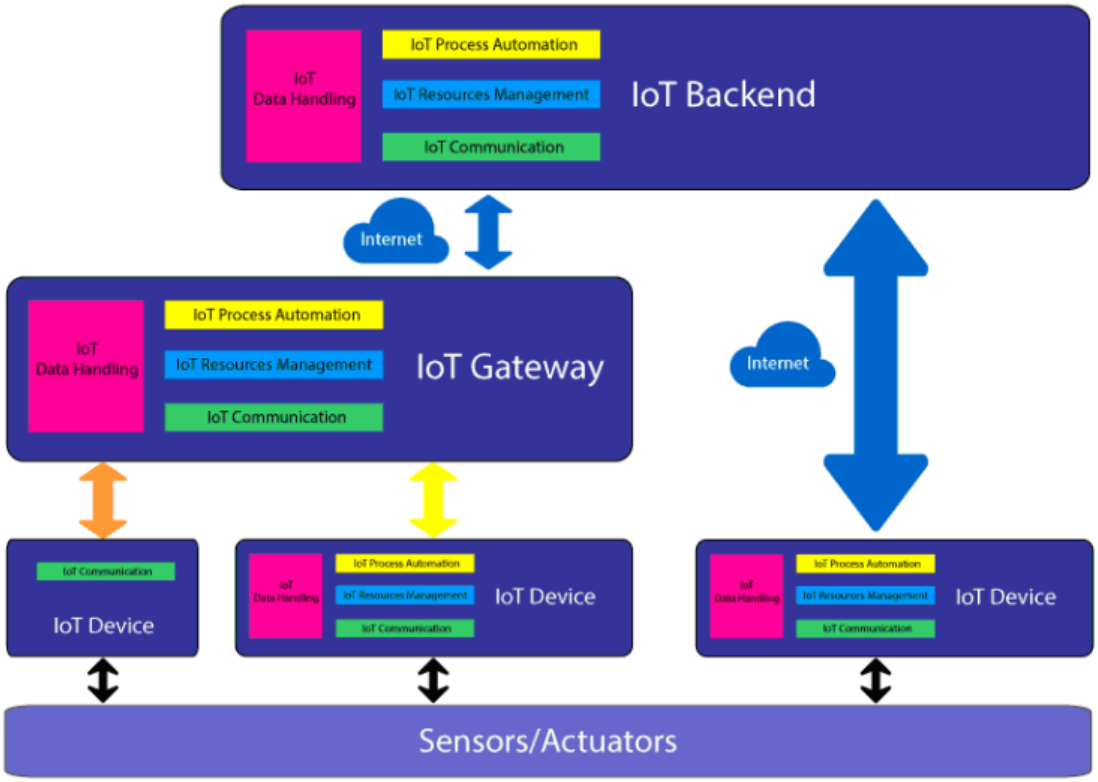
sa často používajú tzv. brány na komunikáciu zo zariadenia do cloudu a naopak často označovaný ako Cloud gateway.

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 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. [15] Cieľom porovnania nie je zaoberať sa detailami implementácie jednotlivých riešení, ale poukázať na principiálnu podobnosť architektúr. Obrázok 29 zovšeobecňuje a znázorňuje všetky možnosti zapojenia hlavných súčastí IoT riešení, ktoré sú uvedené v predošlej citácií a rovnako sa uvádzajú vo väčšine literatúr. Finálne zovšeobecnenie jednotlivých súčasti 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.

2.3.1 Oblasti využitia

Po zadefinovaní hlavných súčasti architektúry IoT riešení, práca popisuje oblasti využitia IoT. Ako podklad pre vymenovanie je dokument [22].

- 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, ...



Obrázok 29: IoT súčasti - všeobecne[23]

- 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, ...

TODO: Inteligentná budova (500) TODO: dalsie oblasti = obrazok oblasti + citat google

2.4 Činnosti súvisiace so softvér architektúrou

TODO: preco kapitola TODO: architektura aplikacie zahrna tieto cinnosti (1000)

2.4.1 Návrh

TODO: nástroje UML, enterprise + pohlady TODO: rozdiely is a iot = aj hw design

2.4.2 Vývoj

TODO: jazyky Java, C#, PHP, JavaScript, Python TODO: Trendy angular a ine technologie na RIA a ine TODO: rozdiely is a iot - pripajanie na senzory a pripajanie na Big Data

2.4.3 Údržba

TODO: ake nastroje nagios a dalsie google TODO: rozdiely starost o hw zariadenia TODO: co je continuous integration a ako zabezpecit

3 Implementácia

Praktická časť druhej fázy štúdia je zameraná na experimenty s IoT prostredím a architektúrou. Ide o IoT architektúru v inteligentnej domácnosti. Do existujúcej architektúry je zavedený nový pojem controller as a service (CaaS) - regulátor ako služba, ktorý je inšpirovaný inými 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 ovplyvňovanie akčného zásahu na základe výpočtu na serveri. Viac je popísane v časti 3.2, teraz nasleduje popis IoT prostredia a architektúry.

3.1 Popis experimentálneho IoT prostredia

Vychádzajúc z obrázkov 26 a 29 prostredie pozostáva z týchto prvkov.

- Fyzická vrstva - snímače.
  - Svetelný senzor.
    - Aeotec MultiSensor 4-in-1.
    - Slim Multi-Sensor PSM01.
  - 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
- Fyzická vrstva - Zariadenia.
  - Raspberry Pi model A+
  - Raspberry Pi 2 model B
  - Vera Control VeraLite
- Komunikačná vrstva smer IoT zariadenie.

  - Z-Wave
  - RSS (HTTP)

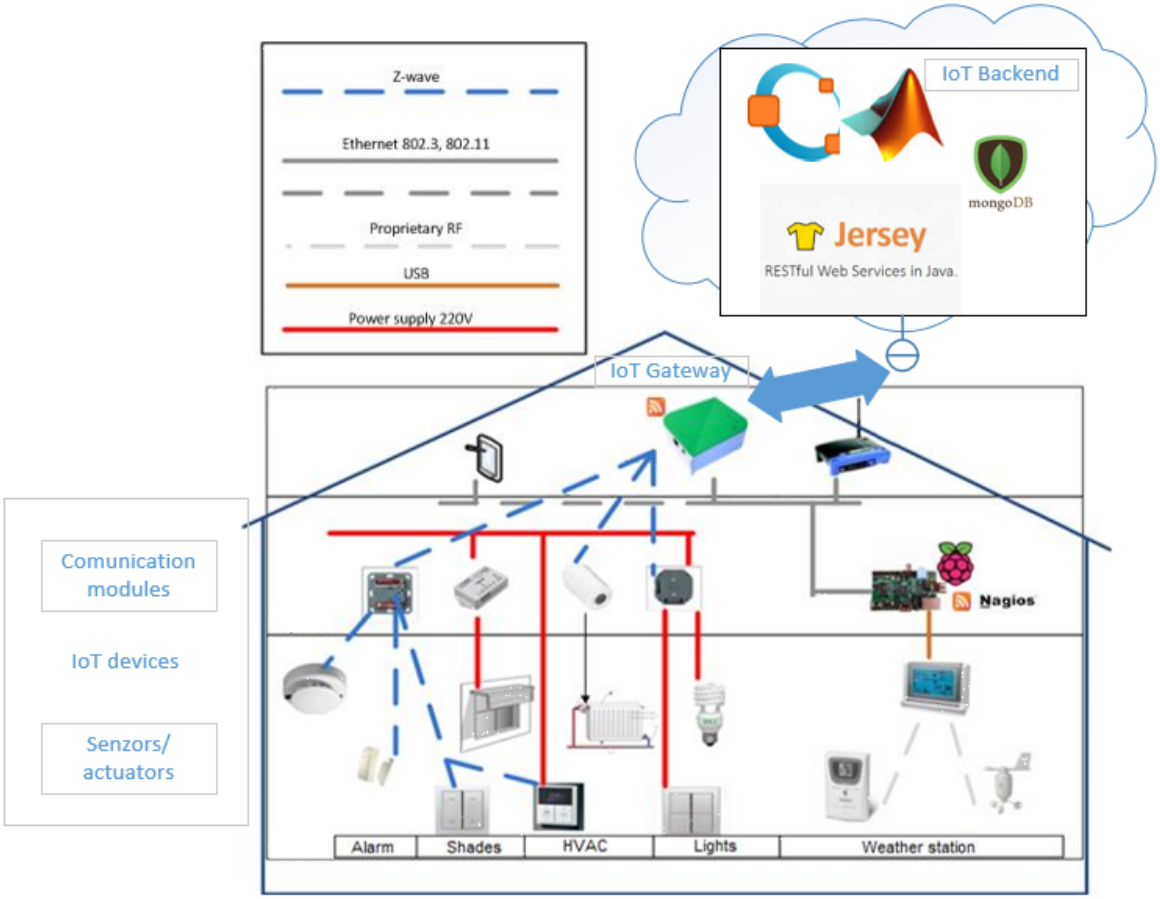
- Komunikačná vrstva smer IoT Backend.
  - REST (HTTP)

- Systémová vrstva.
  - MongoDB databáza
  - Jersey REST interface
  - Matlab/Octave nástroje na výpočty
  - JSON komunikačné rozhranie

- Používateľská vrstva.
  - Správa inteligentnej domácnosti
  - Vizualizácia nameraných údajov

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, ktorý potrebuje na svoje fungovanie jeden hlavný uzol, ktorý spravuje sieť a identifikáciu zariadení. Hlavný uzol v experimentálnom prostredí je zariadenie VeraLite. Existuje viacero alternatív ku tomuto zariadeniu, ktoré majú svoje výhody a nevýhody. Argumenty pre voľbu tohto zariadenia boli stabilita, možnosť programovania 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 [25] 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, 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. 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



Obrázok 30: Experimentálne prostredie

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 tomto 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, vonkajší snímač teploty, vlhkosti a atmosferického tlaku, merač úhrnu zrážok a anemometer na meranie intenzity a smeru vetra. Komunikácia medzi snímačmi a vnútornou jednotkou je prostredníctvom rádiového signálu na frekvencii 433 MHz. Na Raspberry 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 jednotke, ktorá ukladá posledných 24 hodín, ale nie je to úložisko, na ktoré sa možno spoľahnúť. Iné zariadenia, ktoré je možné pripojiť do IoT systému, ale dlhodobo nie sú pripojené sú plošné spoje s rôznym typom senzorov iných ako boli vymenované. Tieto sú pripojiteľné do Raspberry pi pomocou rozhrania GPIO. Na pripájanie takýchto zariadení je v IoT systéme Raspberry pi 2 Model B+. Táto časť je pripravená na jednorázové experimenty. Ďalej nasleduje popis komunikácie medzi zariadeniami.

TODO: nevyhody vo OneNote

3.1.1 Tvorba modulov

Na komunikáciu medzi zariadeniami, ako už bolo spomenuté, je komunikačný protokol Z-Wave. Ten v tejto kapitole nie je zaujímavý. Táto časť popisuje

- spôsob komunikácie zariadení Raspberry Pi s IoT bránou (VeraLite).
- Spôsob programovania v zariadení VeraLite
- a spôsob vytvárania softvérových zariadení v ňom.

Výrobcovia zariadení rodiny Vera si ako základný prvok postavili protokol UPnP ako priemyselný štandard na ovládanie zariadení a skriptovací jazyk Lua. Framework, ktorý vytvorili, sa volá *Luup* [20]. Vďaka UPnP je možné definovať zariadenia, jeho služby, stavové premenné služby, operácie služby a parametre na vstupe a výstupe operácie. Ako príklad je uvedené 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 a jej hodnotu 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*. Kód na zistenie stavu stmievaného svetla vo frameworku *Luup* je zobrazený v algoritme 2. Na

Algoritmus 2 Načítanie stavu stmievaného svetla

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

základe tejto generickosti UPnP protokolu sa v práci definovali tri nové typy zariadení.

- Zariadenie typu barometer s ID *urn:schemas-micasaverde-com:device:Barometer:1* a službou *urn:upnp-org:serviceId:Barometer1*, ktorá bola vytvorená na základe zariadenia pre snímanie teploty so zmenou mena stavovej premennej na *CurrentAirPressure*

- 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 pre nemá a ani nemá mať možnosť zadávať hodnotu, iba ju zo senzora čítať.

- Postupne sa prechádza k vysvetleniu spôsobu komunikácie medzi Raspberry Pi a VeraLite. Pre tento účel sa vytvorilo zariadenie typu RSS Reader s ID *urn:demo-micasaverde-com:device:RssReader:1*.

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. Do tela noviniek generovaných aplikáciou *wview* sa nakonfigurovalo, aby sa tam vkladalo XML v špecifickom tvare, v ktorom sú uložené dáta zo senzorov. Ukážka časti konfiguračného súboru je v algoritme 3.

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>
...
```

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 serverová strana. 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 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“ RSS Reader zariadenia, 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 obrázku je zariade-

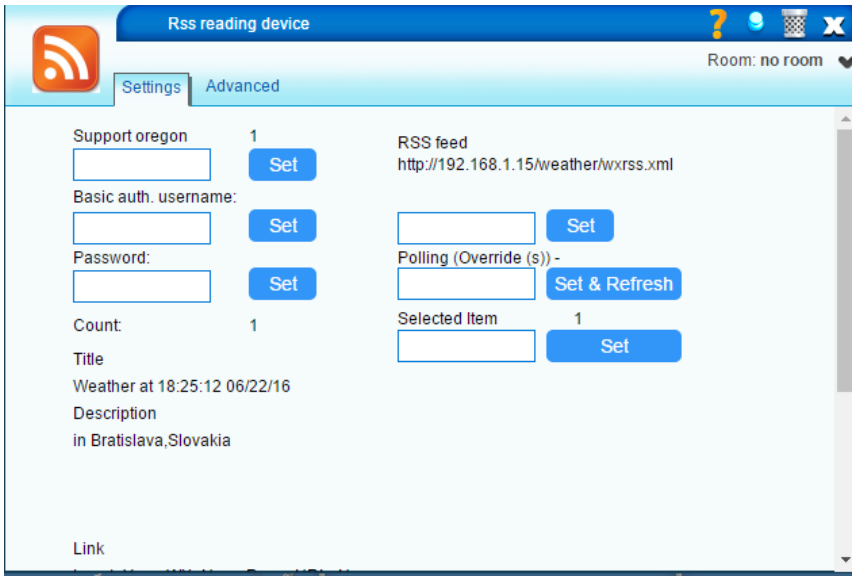
Obrázok 31: Vera GUI - vytvorené softvérové zariadenie

nie 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 Vera 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 authentification spôsobom. 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 už 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.



Obrázok 32: Nastavenia RSS Reader zariadenia

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 a 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. Toto bola ukážka ako programovať do špecifickej IoT brány, ktorá okrem obsluhy Z-Wave zariadení zvláda aj zariadenia komunikujúce s TCP protokolom a jeho nadstavbami (HTTP, RSS, ...), vďaka svojej mikrokernel (plugin) architektúre, kde hlavné jadro je implementované a kto čo potrebuje si môže v podobe pluginu naprogramovať a pridať do mikrokernel prostredia. Ďalej to bola ukážka otvorenosti UPnP protokolu a možnosti definovania vlastných zariadení pomocou neho. Čo v sumáre predstavuje štandardom podporenú mikrokernel architektúru. Zariadenia vytvorené v práci sú dostupné pre využívanie používateľmi Vera zariadenia, prostredníctvom github platformy.

3.2 CaaS

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 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 nedáva zmysel, iba to že môže byť veľa inštancií. Dáva zmysel pre regulátory s veľkými požiadavkami na výkon. Medzi takéto patrí práve online metóda MPC algoritmu, ktorý si vyžaduje, ako je v prvých kapitolách naznačené, prácu s veľkými maticami ich násobenie, inverzia... Práve postup v práci priviedol 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 implementovaný explicitný - offline MPC regulátor. Tejto problematike sa venuje iná dizertačná práca. Predchádzajúce fakty a štúdium IoT architektúr priviedli na myšlienku caas. 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 na toto pripravená. Služba na online identifikáciu pre experiment nebola pripravená, kvôli svojej komplexnosti. Najskôr bolo 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 vysokým výpočtovým výkonom a pamäťou.

3.2.1 Implementácia

Po vysvetlení myšlienky a jej motivácie nasleduje popis implementácie. Štúdiom architektúr a architektonických princípov bol zvolený aplikácie služby (service application) bez grafického rozhrania na základe mikroservice architektúry. Na rovnakom princípe je plán robiť službu identifikácie. Sú to ľahko oddeliteľné funkcionality s potrebou rozdielnej škálovateľnosti, preto je microservice architektúra 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 pokusoch o vytvorenie C programu, 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. Takže nasadzovaní si používateľ môže zvoliť, ktorú aplikáciu chce.

- 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 len ukladá 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 to transformuje. Takže inými slovami robí preklad z JSON notácie na jednotlivé 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 4. Možnosti zadania systému

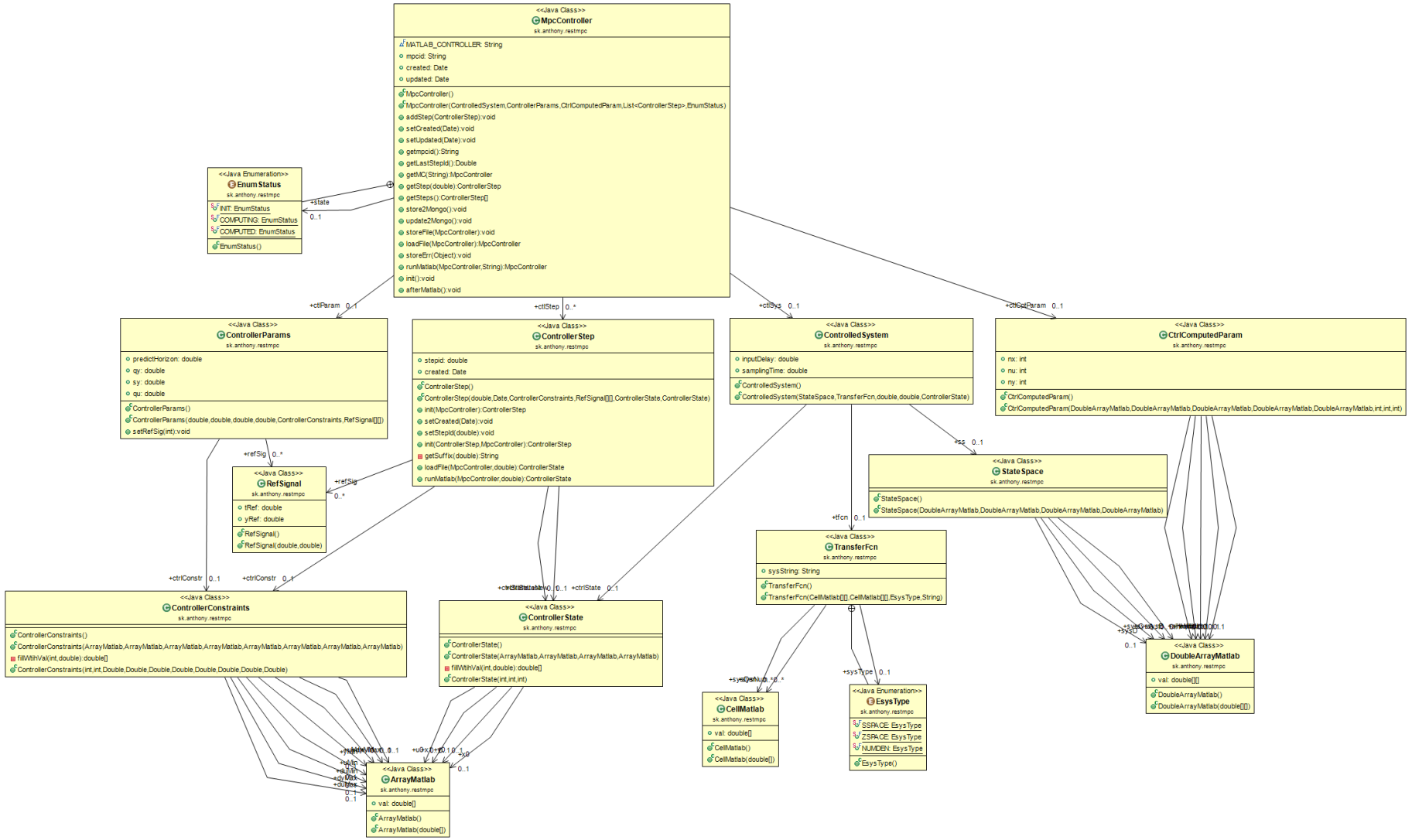
Algoritmus 4 Príklad tela HTTP požiadavky na definovanie MPC

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

sú viaceré. Či už prostredníctvom stavového modelu alebo prenosovej funkcie zadanej reťazcom alebo vektormi čitateľa a menovateľa. Ostatné parametre ako 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 33.



Obrázok 33: Diagram tried vstupného rozhrania.

Funkcionalita, ktorú musí rozhranie spraviť 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 [19], ktorý vie s JSON objektu spraviť Matlab/Octave dátové typy a naopak.
5. Nasleduje krok v Matlab/Octave nástroji, kde prebehne výpočet podľa 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 so 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 tam bol len jeden prvok. Pri ukladaní do súboru sa vynechala vektorová notácia pri a vytváraní objektu vektorová notácia bola očakávaná.

Problem bol s volaním Matlab, ktorý sa nepodarilo spustiť pod používateľom, ktorý je prihláseny ako vlastník procesu aplikačného serveru, na ktorom je 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 zo 14 sekúnd v priemere na 4 sekundy v priemere. 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 je potrebný test na reálnych cloud zariadeniach. Pri načítaní regulátora z databázy je výsledok okolo 400 ms na slabšom zo spomínaných prostrediach. Preto sa dajú očakávať lepšie časy odpovede a teda je možné riadiť procesy s periódou vzorkovania rádovo desiatky milisekúnd. Spôsob

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

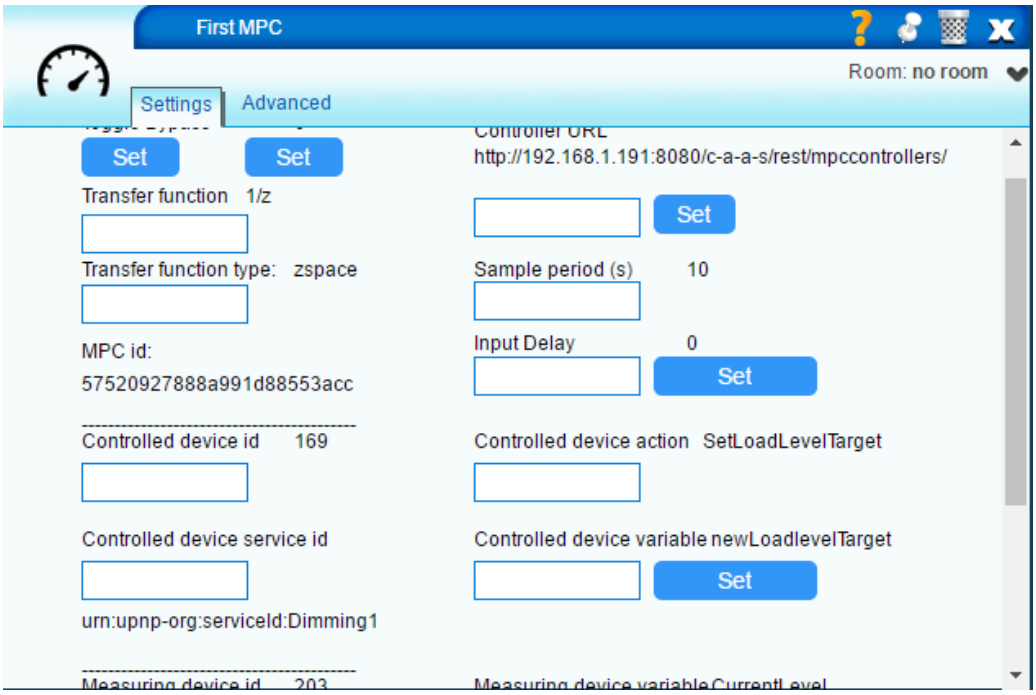
garancie odpovede do určitého času a núdzový režim v prípade sieťového výpadku nie je v práci riešený.

3.2.2 Integrácia

Pre službu regulátora popísanú v predošlej časti a vystavenú v časti IoT Backend je v rámci práce vytvorený klient v prostredí popísanom v kapitole 3.1. Postup vytvorenia klienta aj súčasti potrebné na implementáciu sú podobné. Rýchle opakovanie a popis a zmien v súčastiach oproti RSS Reader zariadeniu. Je potrebná časť na komunikáciu a operácie s textom. Teraz nie je XML správa, ale JSON, ku ktorému tiež na internete existuje knižnica [18], ktorá bola použitá na spracovanie JSON odpovedí. Princíp ostáva podobný, vyskladať a zavolať požiadavku a spracovať odpoveď 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 *urn:demo-micasaverde-com:device:MpcClient:1* so službou *urn:demo-micasaverde-com:serviceId:MpcControl1*. Stavových premenných je tu viac ako v prípade RSS Reader zariadenia. Venovať sa bude 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 viacero akčných členov. Rozhranie na zadávanie parametrov a spustenie riadenia je na obrázku 34. Nachádza sa tam

- vstup pre prenosovú funkciu.
- Vstup pre zadanie URL služby.
- Zobrazenie ID regulátora.
- Tlačítko na vypnutie/povolenie riadenia.
- Vstup pre zadanie periódy vzorkovania, ktorá zároveň identifikuje ako často sa bude volať služba na načítanie akčného zásahu.



Obrázok 34: Rozhranie na zadávanie parametrov regulátora.

- 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 vytvorí job, ktorý pravidelne v čase periódy vzorkovania zavolá IoT Backend, aby vrátil 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.

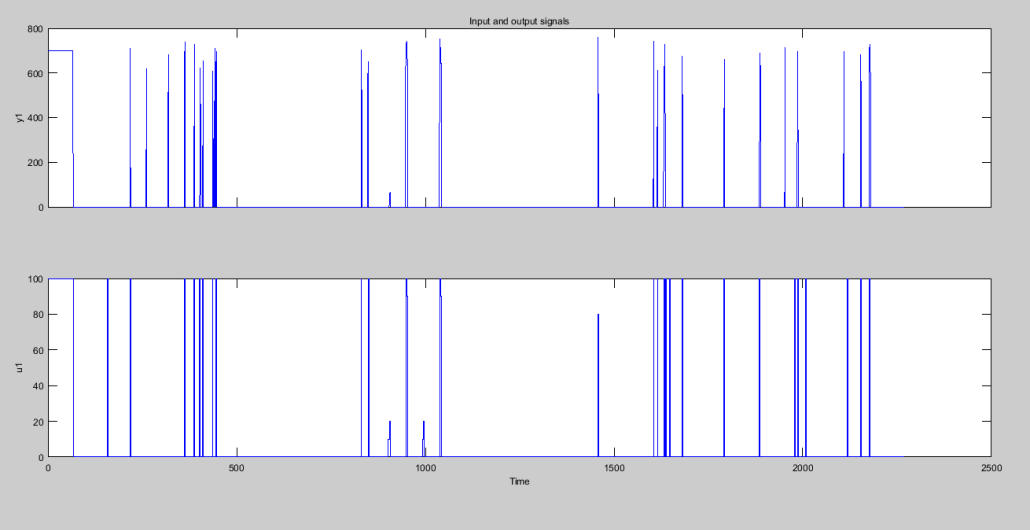
3.2.3 Experiment

Overenie myšlienky je uskutočnené v IoT systéme popísanom v kapitole 3.1. Využíva sa IoT Backend popísaný v kapitole 3.2.1 a klient v podobe softvérového zariadenia implementovaného do Vera IoT brány popísaný v kapitole 3.2.2. Ide o riadenie procesu udržiavania žiadanej intenzity svetla pomocou svetelného zdroja, ktorý je ovplyvňovaný univerzálnym stmievačom FIBARO FGD 211 a snímača intenzity svetla Aeotec Multi-Sensor 4-in-1. Ako podklad dát na identifikáciu procesu poslúžilo softvérove zariadenie dataMine [5] - plugin do Vera zariadenia, ktoré umožňuje ukladať dáta zo zariadení a robiť z nich grafy. Forma ukladania dát je v podobe časová značka, hodnota. Dáta na identifikáciu systému je potrebné ešte upraviť. Keďže každé zariadenie má inú periódu vzorkovania, je potrebné ju zjednotiť. Nevýhoda snímača intenzity svetla bola, že minimálna vzorka bola každé 4 minúty, kvôli úspore batérie. Preto proces mohol mať minimálnu periódu vzorkovania 4 minúty, ktorá bola aj nastavená do softvérového zariadenia. Identifikácia dát prebehla manuálne. Dáta vytiahnuté z dataMine sa vložili do Matlab nástroja. Tam sa dáta upravili jednoduchým algoritmom 5, ktorý prešiel vzorku dát vstupných časov

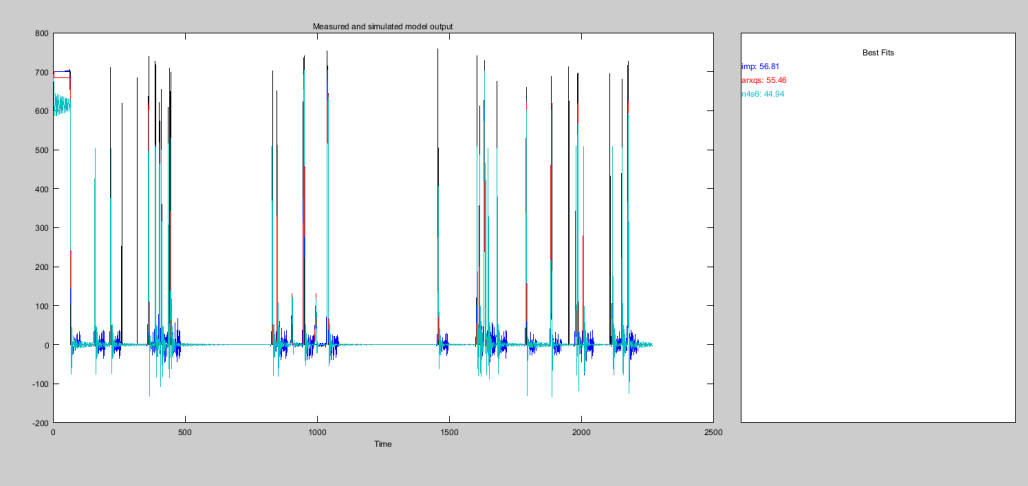
Algoritmus 5 Príprava dát na identifikáciu

```
1  function output=fillval(inputtime,inputvalue)
2  output = zeros(1,floor(inputtime(length(inputtime))));
3  for j=1:floor(inputtime(1))
4      output(j)=inputvalue(1);
5  end
6
7  for i=2:length(inputtime)
8      for j=floor(inputtime(i-1)):floor(inputtime(i))
9          output(j)=inputvalue(i-1);
10     end
11 end
```

a vstupných hodnôt a do každej „medzery“ medzi hodnotami vložil poslednú nameranú hodnotu. To isté sa spravilo aj pre výstupný vektor časov a hodnôt, takže sa spravili vstupné a výstupné dáta s rovnakou periódou vzorkovania 1 minúta. Výstupom tohto procesu úpravy dát je graf 35. Vstupná premenná je percento nastavené na stmievači hodnoty od 0 po 100%. Výstup je hodnota zo snímača, ktorá dosiahla hodnoty 0 po približne 750 luxov. Tieto dáta sa dali do systému Matlab nástroja ident na identifikáciu procesu. Výsledok identifikácie je naznačený v grafe 36. Z identifikovaných systémov sa



Obrázok 35: Namerané dáta experimentu.



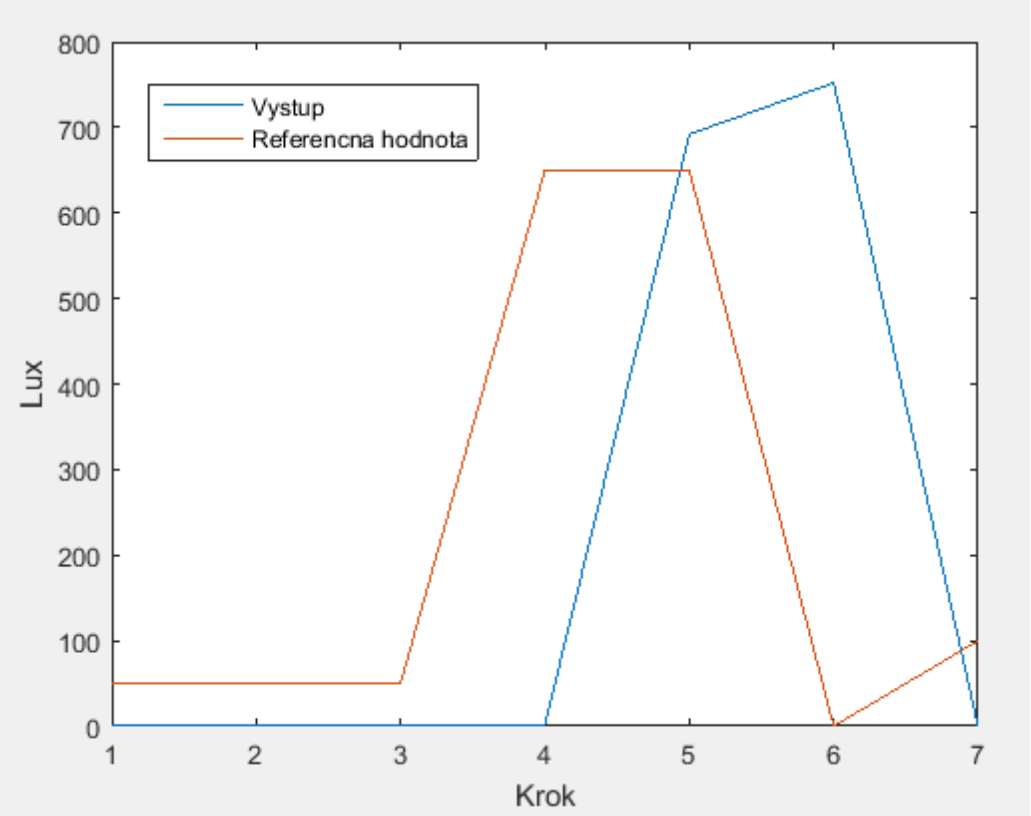
Obrázok 36: Identifikácia systému.

použil systém ns4s6, pretože bol najnižšieho rádu v približne rovnakom percente zhody s ostatnými. Prenosová funkcia tohto systému je:

$$G = \frac{5.088s^5 + 0.9643s^4 + 2.52s^3 + 4.551s^2 + 1.321s + 2.269}{s^6 + 0.01112s^5 + 0.2885s^4 + 0.9443s^3 + 0.1878s^2 + 0.1482s + 0.1376} \tag{40}$$

Výstup riadenia je znázornený v grafe 37. TODO: zmenit graf TODO: pridat hodnotenie kvality riadenia.

Využitie tohto princípu je možné využiť napríklad na udržiavanie intenzity svetla so zapadajúcim slnkom. Rovnako je to možné využiť na regulovanie teploty a aj akýchkoľvek procesov, ktoré majú periódu vzorkovania 4 sekundy a viac. Ukážka prebehla na systéme so svetlom a nie s teplotou, pretože systéme je v byte a tam v čase pripravovania ukážky



Obrázok 37: Graf výstupnej veličiny a žiadanej hodnoty.

boli radiátory vypnuté.

Záver

zhodnotenie

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 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).

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

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

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

[19] 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).

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

[21] 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).

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

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

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

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

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

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

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

[29] 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).

[30] 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).

[31] 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).

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

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

[34] 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).

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

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

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

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