VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

Optimalizovaná kompenzace účiníku v napájecí síti pomocí kondenzátorových bank

Optimized Power Factor Compensation Using Capacitor Banks

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Ján Kseňak
AUTHOR

VEDOUCÍ PRÁCE — Ing. Radek Poliščuk, Ph.D.
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Ján Kseňak**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**Ing. Radek Poliščuk, Ph.D.**
Akademický rok:	2023/24

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Optimalizovaná kompenzace účiníku v napájecí síti pomocí kondenzátorových bank

Stručná charakteristika problematiky úkolu:

Návrh softwarové regulace účiníku v napájecí síti pomocí řízeného vybíjení kondenzátorových bank. Řídicí software poběží na PLC ABB AC500, které bude řešit úlohu softwarově simulovanou na platformě DigSoft PowerFactory.

Cíle diplomové práce:

Popis problematiky a možností řešení kvality kompenzace účiníku v síti
Tvorba OPC UA interface mezi řídicím systémem a simulačnm softwarem DigSoft PowerFactory.
Tvorba testovacího simulačního projektu v PowerFactoy,
Tvorba řídicího regulačního softwaru pro PLC ABB AC500, včetně HMI.

Seznam doporučené literatury:

BERTSEKAS, Dimitri. Dynamic Programming and Optimal Control Volume 1. Third Edition. MIT, 2005. ISBN 1-886529-26-4.

FRANKLIN, Gene F.; POWELL J. David; EMAMI-NAEINI, Abbas. Feedback Control of Dynamic Systems, Eighth Edition. Pearson Education, 2019. ISBN 978-0-13-468571-7

[3] ABB, s.r.o. PLC Automation, Automation Builder, programmable logic controlers, control panels Manual. ABB, s.r.o. , 2023.

[4] DIgSILENT Gmbh. DIgSILENT Power Factory Version 2023 User Manual. DIgSILENT Gmbh , 2023.

[5] TŮMA, Jiří; FARANA, Radim; , Renata Wagnerová; LANDRYOVÁ, Lenka a WAGNEROVÁ, Renata. Základy automatizace. Ediční středisko VŠB – TUO, 2007. ISBN 978-80-248-1523-7.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2023/24

V Brně, dne

L. S.

Ing. Pavel Heriban, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Cieľom diplomovej práce je návrh regulátora účinníku pre elektrickú sieť, vrátane HMI. Teoretická časť práce je rozdelená na dva logické celky. Prvá časť sa zaoberá problematikou korekcie účinníka v lineárnom elektrickom prostredí, bez uvažovania harmonického skreslenia. Druhá polovica teoretickej časti je venovaná technológiám, ktoré boli použité pri tvorbe kooperatívneho simulačného prostredia. V kooperatívnom simulačnom prostredí následne prebehlo virtuálne uvedenie do prevádzky celého systému. V empirickej časti práce je popísaná architektúra vytvorené kooperatívneho simulačného prostredia postaveného na frameworku Mosaik. Ďalej sa pokračuje v popise samotného regulátora, jeho komponent, programu a HMI. V rámci poslednej kapitoly bol celý systém uvedený do virtuálne prevádzky, pri ktorej prebehlo aj funkčné testovanie. Regulátor bol úspešne otestovaný na zmenu požadovanej hodnoty účinníku a na zmenu systému bez zmeny požadovanej hodnoty účinníku.

ABSTRACT

The aim of the thesis is the design of a power factor controller for the power grid, including HMI. The theoretical part of the thesis is divided into two logical units. The first unit explain the problem of power factor correction in a linear electrical environment, without considering harmonic distortion. The second theoretical unit describes the technologies, applied in cooperative simulation environment. A virtual commissioning of the whole system was performed in the simulated environment. The empirical part of the thesis describes the architecture of the Mosaik simulation framework. Then folows description of the controller, its integral components and the HMI. In the last chapter, the whole system was put into virtual operation, in order to perform function tests. The controller was tested on change of the power factor setpoint and to test the reaction on system change, without losing the power factor compenation.

KLÍČOVÁ SLOVA

Kompenzácia účinníka, Stupňovitá regulácia účinníka, Kooperatívna simulácia, Virtuálne uvedenie do prevádzky, Mosaik, DigSilent PowerFactory, OPC UA, ABB AC500 V3

KEYWORDS

Power factor compensation, Power factor step regulations, Co-simulation, Virtual commissioning, Mosaik, DigSilent PowerFactory, OPC UA, ABB AC500 V3

ÚSTAV AUTOMATIZACE A INFORMATIKY

2024

BIBLIOGRAFICKÁ CITACE

KSEŇAK, Ján. Optimalizovaná kompenzace účiníku v napájecí síti pomocí kondenzátorových bank. Brno, 2024. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/157674. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Radek Poliščuk

POĎAKOVANIE

Rád by som poďakoval vedúcemu tejto diplomovej práce, pánovi Dr. Radkovi Poliščukovi, Ing. Zdeňkovi Obornému a Ing. Rostislavovi Martiňakovi zo spoločnosti ABB, za ich odborné rady, priateľský prístup a cenné pripomienky pri práci na diplomovej práci.

ČESTNÉ PREHLÁSENIE

Prehlasujem, že táto práca je mojím pôvodným dielom, vypracoval som ju samostatne pod vedením vedúceho práce a s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry.

Ako autor uvedenej práce danej prehlasujem, že v súvislosti s vytvorením tejto práce som neporušil žiadne autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích osobnostných práv a som si plne vedomý následkov porušenia ustanovenia § 11a nasledujúcich autorského zákona c. 121/2000 Sb., vrátane možných trestne právnych následkov.

V Brne dňa 24. 5. 2024 ……

Bc. Ján Kseňak

1 ÚVOD

V súčasnej dobe, ktorá sa zameriava efektívne využívanie energetických zdrojov sa paralelná kompenzácia účinníku stala jednou z dôležitých techník v oblasti riadenia kvality elektrickej energie, čo vedie k optimalizácii využitia energetických zdrojov a znižovaniu strát v elektrických sústavách.

Cieľom tejto diplomovej práce je preskúmať a analyzovať problematiku kvality elektrickej energie so zameraním na paralelnú kompenzáciu účinníku. Obsah práce sa nezaoberá len teoretickým aspektom paralelnej kompenzácie, ale aj praktickej aplikácií korekcie účinníka pomocou navrhnutého regulátora, ktorý beží na platforme PLC AC500 V3.

Zároveň sa práca zaoberá aj virtuálnym uvedením do prevádzky elektrického systému, na ktorom bude prebiehať riadená korekcia účinníku. Virtuálne uvedenie do prevádzky poskytuje možnosť testovania a simulácie nových riešení bez výrazných vstupných nákladov spojených s fyzickou implantáciou. Týmto spôsobom môžeme získať lepší prehľad o výkonnosti a efektivite nových technológií ešte pred ich skutočným nasadením.

Virtuálne uvedenie do prevádzky funguje na princípe kooperatívnej simulácie elektrickej siete v simulačnom programe DigSilent PowerFactory a reálneho regulátora voel forme PLC. DigSilent PowerFactory je špecializovaný softvér určený pre analýzu a simulácie elektrických sietí a systémov. Jadro kooperatívnej simulácie využíva framework Mosaik. Mosaik slúži na kooperatívne simulácie, čo umožňuje efektívnu integráciu rôznych simulačných prostredí a modelov. Tieto nástroje nám umožnia simulovať a testovať rôzne scenáre paralelnej kompenzácie účinníkov a ich dopad na výkonnosť a efektivitu elektrických sietí.

Navrhnutý regulátor vychádza z riešenia stupňovitej regulácie účinníku. Regulátor na riadenie účinníka elektrickej siete predstavenej v prípadovej štúdii analyzuje elektrické veličiny a na základe nich vyhodnocuje potrebné množstvo jalového výkonu, ktoré treba dodať alebo odobrať zo systému, tak aby meraná hodnota účinníku dosiahla požadovanú hodnotu. Riadiaci systém obsahuje tiež funkcionalitu na optimálne rozdelenie odchýlky jalového výkonu medzi dve kompenzačne skupiny, ktoré sú pripojené na samostatných podružných termináloch, ako aj a na optimálny výber kompenzačných stupňov. Navrhnutý regulátor tiež dokáže zaručiť optimálny výber kompenzačných stupňov so zameraním na prevádzkové hodnoty, aby boli rovnomerne opotrebované.

K navrhnutému regulátoru bola vytvorená aj HMI (Human Machine Interface) komponenta, ktorá slúži ako grafické rozhranie medzi regulátor, riadeným systémom a operátorom prevádzky. V grafickom prostredí môže operátor sledovať aktuálny stav systému, nastavovať rôzne parametre regulátora, sledovať trendy meraných veličín alebo sledovať vyvolané alarmy spôsobené prípadnou poruchou systému.

2 PREHĽAD SÚČASNÉHO STAVU POZNANIA

2.1 Elektrický výkon v lineárnom prostredí

Pre jednoduchosť a názornosť budú použité vzťahy výkonov pre jednofázové lineárne prvky napájané harmonickým napätím s frekvenciou 50 Hz. Pri uvážení trojfázovej sústavy aj jej symetrického zaťaženia môže jednofázové výkony vynásobiť tromi alebo pri nesymetrickej záťaži jednofázové výkony sčítať.

Reálnymi lineárnymi prvkami sú rezistor, kondenzátor a cievka. U lineárnych prvkov platí lineárna závislosť medzi napätím a prúdom, a VA charakteristika je teda priamka. Z toho vyplýva, že pri napájaní sínusovým napätím bude záťaž odoberať sínusový prúd.

Napätie v harmonickom ustálenom striedavom stave sa môže vyjadriť jednoduchým vzťahom:

$$u(t) = U_m \cdot \sin(\omega t) \qquad (V;\ V) \tag{1}$$

a pripojená záťaž bude zo zdroja odoberať okamžitý výkon s hodnotou danú vzťahom:

$$p(t) = u(t) \cdot i(t) \qquad (W;\ V, A) \tag{2}$$

Cez pripojený lineárny spotrebič charakterizovaný impedanciou $Z = \mathrm{R} + \mathrm{jX}$ bude pretekať elektrický prúd podľa vzťahu:

$$i(t) = \frac{u(t)}{Z} = \frac{u(t)}{R + jX} = I_m \cdot \sin(\omega t - \varphi) \qquad (A;\ V, \Omega) \tag{3}$$

Pri uvážení nulového fázového posunu pri harmonickom prúde sa efektívna hodnota harmonického prúdu a napätia dá odvodiť nasledujúcimi vzťahmi:

$$U = \frac{1}{T}\sqrt{\int_0^T u(t)^2 dt} = \frac{1}{T}\sqrt{\int_0^T (U_m^2 \cdot \sin^2(\omega t))dt} = \frac{U_m}{\sqrt{2}} \tag{4}$$

$$I = \frac{1}{T}\sqrt{\int_0^T i(t)^2 dt} = \frac{1}{T}\sqrt{\int_0^T (I_m^2 \cdot \sin^2(\omega t))dt} = \frac{I_m}{\sqrt{2}} \tag{5}$$

Po dosadení vzťahov (1) a (3) do vzťahu (2) a úprave goniometrických výrazov získame nasledujúci vzťah pre okamžitý výkon:

$$p(t) = U_m \cdot \sin(\omega t) \cdot I_m \cdot \sin(\omega t - \varphi) = \frac{U_m I_m}{2} \cdot [\cos\varphi + \cos(2\omega t + \varphi)] \tag{6}$$

Vzťah (6) môžeme rozdeliť na dve časti. Prvá je nezávislá na čase a nazýva sa *činný výkon* a je vyjadrený ako:

$$P = \frac{U_m \cdot I_m}{2} \cdot \cos\varphi = U \cdot I \cdot \cos\varphi \qquad (W) \tag{7}$$

V elektrických systémoch činný výkon (reálny výkon) sa nenávratne spotrebuje a premení na inú formu energie alebo užitočnú prácu, ako napríklad pohon strojov, teplo alebo osvetlenie.

Druhá polovica vzťahu je časovo závislá a kmitá s dvojnásobnou frekvenciou, veľkosť amplitúdy je rovná súčinu efektívnych hodnôt napätia a prúdu. Stredná hodnota tejto zložky výkonu počas jednej periódy je rovná nule.

V krajných prípadoch, kedy majú napätie a prúd rovnaký fázový posun (nachádzajú sa vo fáze) dostaneme výkon, ktorý sa nazýva zdanlivý. To v reálnych prípadoch sa skoro nikdy nestane. Ak je v obvode pripojený nejaký reálny prvok s impedanciou a s obecným induktívnym alebo kapacitným charakterom, bude hodnota fázového posunutia nenulová. To znamená, že hodnota činného výkonu bude vždy menšia ako hodnota zdanlivého výkonu. Zdanlivý výkon je vyjadrený ako:

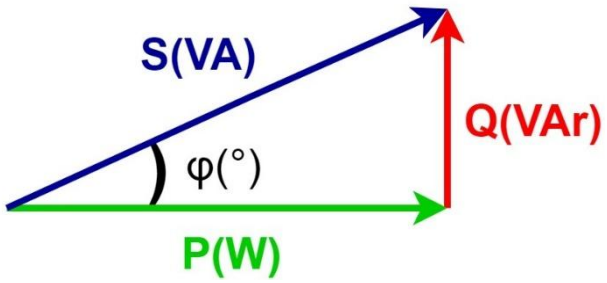
$$S = U \cdot I \qquad (VA;\ V, A) \tag{8}$$

Ako už bolo spomenuté, kmitavá zložka okamžitého výkonu má nulovú strednú hodnotu počas celej jednej periódy. Žiadna užitočná práca sa teda nekoná. Avšak elektrický prúd tejto zložky pri motoroch generuje magnetické pole, ktoré je nevyhnutné pre prevádzku. Tento prúd sa nazýva reaktívny elektrický prúd a výkon, ktorý je prenášaný sa taktiež nazýva *reaktívny výkon*, ale môžeme sa stretnúť s názvom ako *jalový výkon*. Tým, že jalový výkon nekoná žiadnu užitočnú prácu, je často pri presnom vyjadrení fyzikálnych zákonoch považovať jalový výkon za nepodstatný. Jalový výkon je vyjadrený vzťahom:

$$Q = U \cdot I \cdot \sin\varphi \qquad (VAr;\ V, A) \tag{9}$$

Vzájomné vzťahy medzi činným, zdanlivým a jalovým výkonom je možné vyjadriť graficky ako vektory. Táto grafická reprezentácia sa tiež nazýva aj *výkonový trojuholník* a jeho analytické vyjadrenie sa dá jednoducho odvodiť pomocou Pytagorovej vety:

$$S = \sqrt{P^2 + Q^2} \qquad (VA;\ W, VAr) \tag{10}$$



Obr. 1: Výkonový trojuholník.

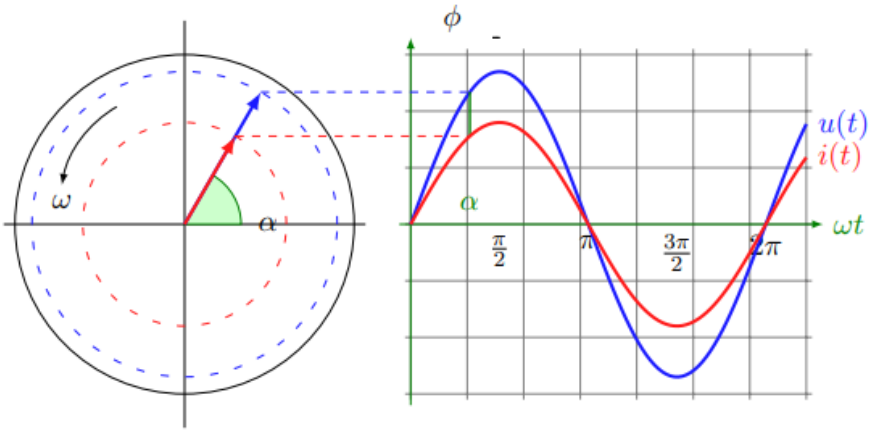
Hodnota uhlu impedancie spotrebiča φ je zároveň hodnota fázového posunu medzi napätím a prúdom. Kosínus tohto uhla sa nazýva *účinník* (v anglickej literatúre *Power Factor*, v skratke PF). Táto hodnota je definovaná ako pomer medzi činným výkonom a zdanlivým výkonom. Čím menší je uhol, tým väčšia je hodnota účinníku a prenos výkonu je preto efektívnejší. Vzhľadom na to, že je účinník párna funkcia, nadobúda rovnaké hodnoty pre záporné a kladné hodnoty fázového posunu. Rozlišuje sa však charakter účinníku na *indukčný charakter* alebo *kapacitný charakter*. Vzťah pre účinník je nasledovný:

$$\cos\varphi = \frac{P}{S} \quad (-;\ W, VAr) \tag{11}$$

Je zrejmé, že hodnota účinníku je priamo zviazaná s typom záťaže. Každú lineárnu záťaž môžeme vyjadriť sériovým, paralelným alebo kombinovaným zapojením ideálnych lineárnych prvkov a tými sú:

- **Rezistor**

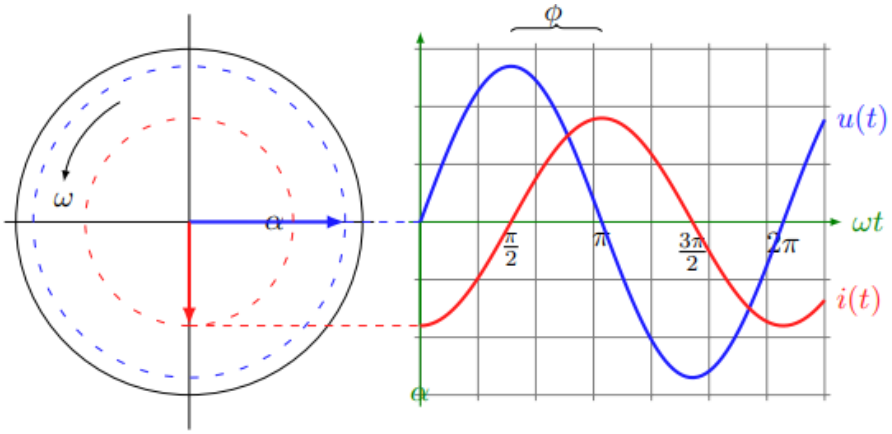
Po pripojení ideálneho rezistora s odporom R na zdroj harmonického napätia, začne rezistorom pretekať harmonický elektrický prúd. Okamžité hodnoty napätia a prúdu budú mať rovnakú frekvenciu a medzi nimi nebude žiaden fázový posun ($\varphi = 0$). Okamžitý výkon, vypočítaný podľa vzorca (6) bude mať dve zložky. Konštantnú ($U \cdot I$). a kmitavú $U \cdot I \cdot \cos(2\omega t)$. Z toho vyplýva, že okamžitý výkon bude kmitať okolo konštantnej zložky, ktorá je aj jeho stredná hodnota. Okamžitý výkon bude vždy kladný, pretože aj súčin napätia a prúdu je vždy kladný. Rezistor je teda iba spotrebičom elektrickej energie. Veľkosť činného a zdanlivého výkonu je rovnaká, pretože fázový posun je nulový.



Obr. 2: Okamžitý priebeh el. napätia a prúdu na ideálnom rezistore. [20]

- **Induktor**

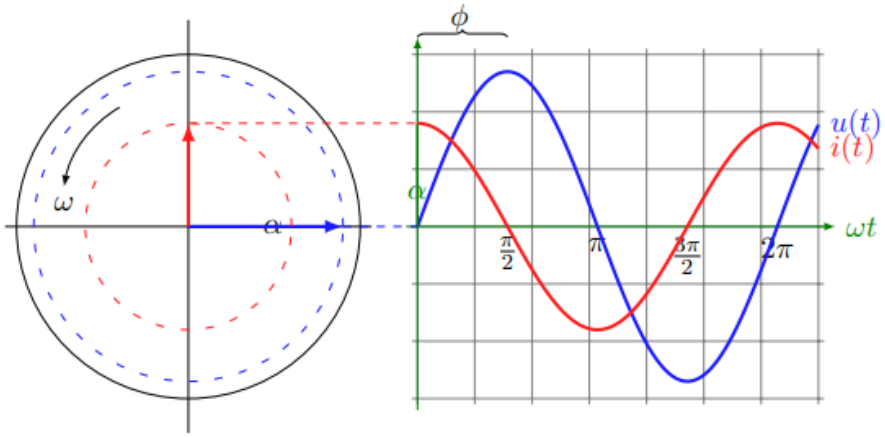
Priebeh okamžitého prúdu bude mať rovnakú frekvenciu ako má napätie, avšak bude zaostávať voči napätiu o $\frac{\pi}{2}$. Amplitúda prúdu bude $I_m = \frac{U_m}{\omega L}$, kde súčin ωL sa taktiež označuje ako X_L. Nazýva sa to ako *indukčná reaktancia* alebo *indukčný jalový odpor*. Okamžitý výkon podľa vzorca (6) bude mať len jednu kmitavú zložku $U \cdot I \cdot \sin(2\omega t)$. Z toho vyplýva, že dochádza k periodickej výmene energie medzi zdrojom a magnetickým poľom induktora. Hodnota stredného výkonu je nulová. Týmto nedôjde ku žiadnej premene elektrickej energie na iný druhu energie v ideálnom induktore. Oscilácia výkonu však spôsobuje zaťaženie prenosového vedenia. To znamená, že jalový výkon induktorom je nevyhnutný na prevádzku väčšiny elektrických zariadení.



Obr. 3: Okamžitý priebeh el. napätia a prúdu na ideálnom induktore. [20]

- **Kapacitor**

Ak je kapacitor pripojený na zdroj harmonického napätia, tak ním začne pretekať harmonický prúd s rovnakou frekvenciu, ale bude predbiehať napätie o uhol $\frac{\pi}{2}$. Amplitúda prúdu bude mať hodnotu $I_m = \omega \cdot C \cdot U_m$, kde obrátená hodnota súčinu $\frac{1}{\omega \cdot C}$ sa nazýva *kapacitná reaktancia* alebo *kapacitný jalový odpor* a označuje sa X_C. Priebeh okamžitého výkonu sa správa inverzne voči induktorom, pričom sa znova nie vykonaná žiadna užitočná práca. Integrál strednej hodnoty výkonu je nulový.



Obr. 4: Okamžitý priebeh el. napätia a prúdu na ideálnom kapacitore. [20]

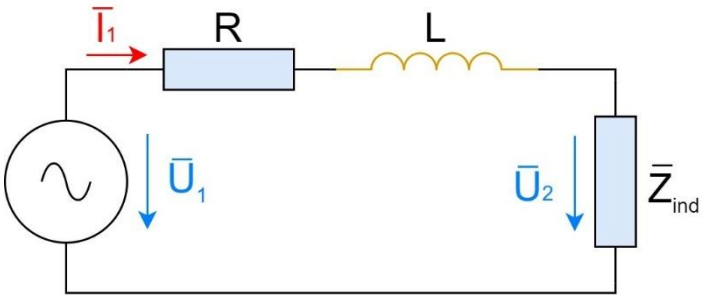
Popísane správanie ideálnych prvkov v reálnom svete neexistuje. Každý reálny elektrický prvok obsahuje nejaký parazitný odpor, kapacitu a indukčnosť. Popísaná výkonová teória funguje iba v prípade v prostredí bez harmonického skreslenia. [1]

2.1.1 Vplyv účinníku na kvalitu elektrickej energie

Technické a ekonomické dopady na prevádzkovanie zariadení s nízkou hodnotou účinníku môžu byť významné. V kapitole bude rozobrané základne technické a ekonomické dopady nevhodnej hodnoty účinníku. V rámci technických dopadov na prevádzkovanie zariadení s nízkym účinníkom je množstvo dostupnej elektrickej energie v závode u distribučného transformátoru významne znížené vplyvom jalovej energie, ktorú musí transformátor niesť. Ďalším technickým dopadom sú výkonové straty na vedení pri prevádzke $\cos\varphi < 1$. Toto tvrdenie môže popísať vzťahom 12, ale aj tým, že prenosové vedenie je zaťažené okrem činného výkonu aj jalovým:

$$\Delta P = 2 \cdot R \cdot \left(I_č^2 + I_j^2\right) = 2 \cdot R \cdot \left(\frac{I_č^2}{\cos^2\varphi}\right) \tag{12}$$

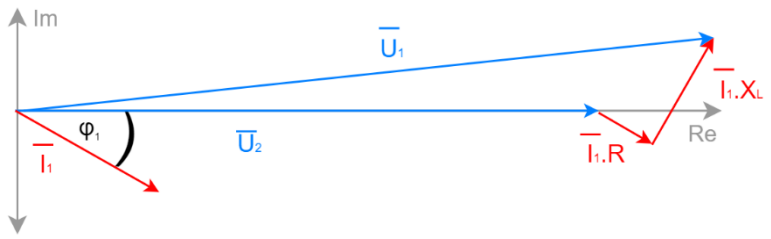
Ďalším vplyvom je úbytok napätia na prenosovom vedení, ktorý je charakterizovaný vlastným odporom a induktívnou reaktanciou, pozri náhradne schéma na obrázku 5.



Obr. 5: Náhradné schéma prenosového vedenia a záťaže.

Úbytok napätia daný vzťahom 13, ide vyjadriť aj formou *fázorového diagramu*, ktorý sa nachádza na obrázku 6 :

$$\Delta U = R \cdot I_č + X_L \cdot I_j = R \cdot I_č + X_L \cdot I_č \cdot \tan\varphi \tag{13}$$



Obr. 6: Fázorový diagram náhradnej schémy.

Najčastejšie sú prevádzkované induktívne spotrebiče (napr. asynchrónny motor), tie vedú k úbytku napätia. V prípade kapacitného charakteru spotrebiča dôjde k nárastu napätia.

Posledným technickým vplyvom, ktorý predstavíme je nižšia zaťažiteľnosť elektrickej sústavy, ktorá je daná vzťahom 14. Zo vzťahu je evidované, že pri nižšej hodnote účinníku narastie hodnota pravej strany rovnice. To môže spôsobiť nežiaduce výpadky istiacich prvkov alebo spôsobiť preťaženie distribučného transformátora.

$$I_{DOV} \geq \frac{P}{\sqrt{3} \cdot U_N \cdot \cos \varphi} \tag{14}$$

Všetky spomenuté dopady vnímajú distribučné spoločnosti ako negatíva a obvykle prenášajú na zákazníkov náklady na prevádzky zariadení s nízkym účinníkom. Platby za nízku hodnotu účinníku sa dajú účtovať ako jeden z možných spôsobov, alebo ich kombinácia:

- Pokuta za hodnotu účinníku pod predom stanovenú hodnotu alebo nad predom stanovenú hodnotu.
- Zväčšovanie pokuty so znížením hodnoty účinníka.
- Poplatok sa mesačné hodiny kVAR.
- Priamy poplatok za maximálnu hodnotu kVA spotrebovanú v priebehu mesiaca.

V Českej republike podľa Energetického zákona č. 458/2000 Sb. je odberateľ elektrickej energie povinný dodržiavať účinník v pásme 0,95-1, pokiaľ nie je dopredu dohodnuté inak. Riešenie maloodberateľov je zastrešené individuálnou distribučnou spoločnosťou.

2.1.2 Paralelná kompenzácia účinníku

Korekcia účinníku prebieha dvoma spôsobmi:

- Znížením množstva jalovej energie tým, že odstránime zariadenia s nízkou hodnotou účinníku, ako napríklad motory alebo transformátory.
- Pripojením externých kompenzačných kondenzátorov alebo iných zariadení, ktoré slúžia na korekciu účinníku.

Najčastejším aj najekonomickejším riešením je paralelné pripojenie kondenzátora. Tieto kondenzátory majú obvykle hodnoty v kVAR a sú dostupné pre rôzne typy elektrických sústav. Zvyčajne je potrebný viac ako jeden kondenzátor, aby sa dosiahla škálovateľnosť pre požadovanú hodnota účinníku. Návrh vhodných kompenzačných kondenzátorov je samostatná technická disciplína a nie je predmetom tejto práce. Inštalácia kondenzátorov môže byť realizovaná spôsobmi:

- Individuálne kondenzátory pre každé elektrické zariadenie.
- Kondenzátorová banka inštalovaná pre určitú oblasť v prevádzke.
- Kombinácia predchádzajúcich spôsobov.

Spomenutá technika kompenzácie účinníka pripojením kondenzátorov, spadá do kategórie statických kompenzátorov. Pripojovanie jednotlivých kondenzátorov môže prebiehať pomocou:

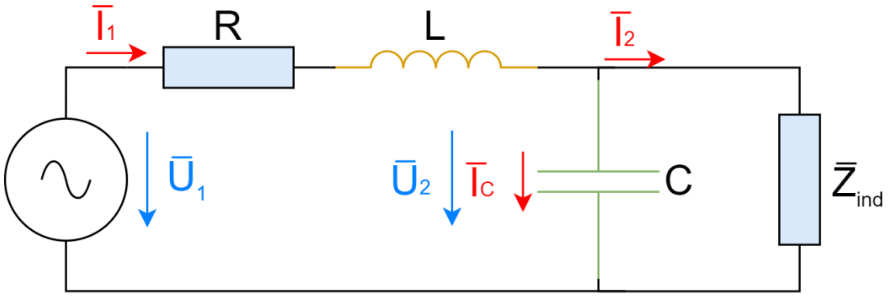
- **Kontaktného spínania**

Na pripojenie kondenzátorov sa používajú stýkače. Moderné stýkače majú vlastnosť k potláčaniu prúdových rázov, ktoré môžu spôsobiť pripojenie kondenzátora. Stýkače sú elektromechanické systémy, ktoré sú schopné spínať s periódou rádovo v sekundách. Tento spôsob sa používa iba v prípadoch s pomalšou dynamikou. Ak je frekvencia zmien jalového výkonu na úrovni frekvencie spínania stýkačov, je potrebné použiť spôsob bezkontaktného spínania.

- **Bezkontaktného spínania**

V systémoch s rýchlou dynamikou (rýchla zmena jalového výkonu) sa používa bezkontaktné spínanie. To je realizované pomocou tyristorov. Nedochádza tu k prechodovým dejom, pretože spínanie prebieha pri prechode napätia nulou a môžu spínať v časoch jednej periódy sieťovej frekvencie. Toto však vyžaduje rýchlejšie regulátory, ktoré dokážu spínať s požadovanou rýchlosťou.

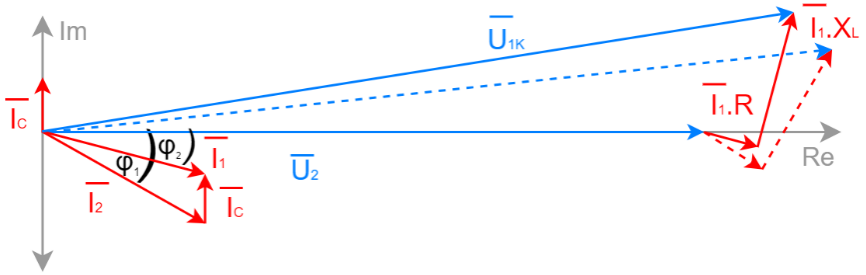
Na obrázku 7 je ukážkový príklad paralelnej kompenzácie. Do náhradnej schémy prenosového vedia z obrázku 5 je pripojený kompenzačný kondenzátor.



Obr. 7: Náhradné schéma prenosového vedenia a záťaže po kompenzácií.

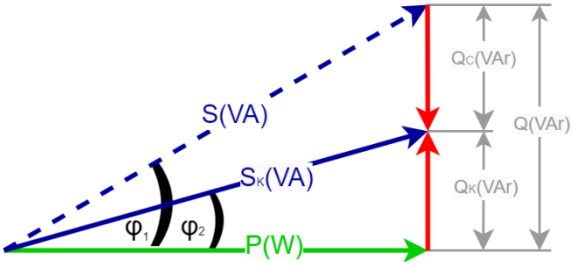
Na fázorovom diagrame a v dôsledku vzťahu 15 môžeme sledovať tieto pozitívne efekty. Úbytok napätia sa zmenšil a účinník sa zvýšil.

$$\Delta U = R \cdot I_{č} + X_L \cdot I_j - X_C \cdot I_j \tag{15}$$



Obr. 8: Fázorový diagram náhradnej schémy po kompenzácií.

Taktiež dodaním kapacitného jalového výkonu Q_C, pomocou kompenzačných kondenzátorov sa celkový prenášaný výkon S sa zmenšil na hodnotu S_K. Súčasne sa zmenšil uhol medzi zdanlivým výkonom a činným výkonom, inak povedané zväčšila sa hodnota účinníku. Prenášaný činný výkon ostáva nezmenený.[2; 3]



Obr. 9: Výkonový trojuholník po kompenzácií.

2.1.3 Stupňovitá regulácia účinníku

Pri tomto spôsobe regulácie regulátor automaticky pripojuje jeden alebo viacero kompenzačných stupňov. Dodaný kompenzačný výkon teda rastie alebo klesá stupňovito na základe spínania jednotlivých stupňov. Pripojenie stupňov môže prebiehať kontaktným aj bezkontaktným spôsobom. Regulátorom, ktorý spína jednotlivé stupne je často mikroprocesorová technika.

Všeobecne sa počet a veľkosť stupňov volí tak, aby kompenzačná banka disponovala určitým jalovým výkonom Q_C a dostatočnou presnosťou regulácie. Pri nevhodnom výbere kompenzačných stupňov by mohlo dôjsť k nedostatočnej kompenzácií alebo k nadmernej kompenzácií.

V praxi výber jednotlivých kompenzačných stupňov často podlieha skúsenostiam realizátora a ide o kompromis medzi potrebnou citlivosťou regulácie a cenou kompenzačnej jednotky.[3]

2.2 Kooperatívna simulácia a virtuálne uvedenie do prevádzky

Budúcnosť energetických systémov čelí niekoľkým výzvam. Jednou z nich je výrazná zmena v spôsobe využívania elektrickej energie, s dôrazom na efektivitu a znižovanie elektrických strát. Preto je modernizácia infraštruktúry energetickej siete nevyhnutná na podporu inteligentnej prevádzky a zabezpečenie spoľahlivosti, kvality a výkonnosti v budúcnosti. Inteligentná sieť by mala byť schopná efektívne zvládať decentralizovanú a vysoko premenlivú výrobu obnoviteľnej energie.
Implementácia inteligentných riešení si vyžaduje aj realistické, masívne online simulácie v reálnom čase, aby sa predpokladané problémy sieti mohli riadiť a riešiť včas.

V kapitole je podrobne predstavený *simulačný framework Mosaik*, ktorého úlohou je riadenie viacerých simulátorov, ich synchronizácia a výmena dát medzi simulátormi. Kapitola taktiež predstavuje simulačný program *DIgSILENT PowerFactory*. V tejto súvislosti je DIgSILENT PowerFactory dôležitý nástroj, ktorý umožňuje modelovanie, simuláciu a analýzu inteligentných sietí. Poskytuje tiež funkcie pre monitorovanie,

vizualizáciu, hodnotenie dynamických vlastností v reálnom čase, rôzne typy meraní a rozhraní, ktoré slúžia na tvorbu kooperatívnych simulačných scenárov.

Virtuálne uvedenie do prevádzky

Emulácia systému na overenie regulačnej logiky sa označuje ako virtuálne uvedenie do prevádzky. Pod tým rozumieme proces, ktorý zahŕňa tvorbu digitálnej repliky jedného alebo viacerých prvkov systému so softwarovými nástrojmi (zvyčajne pre systém v rámci návrhu alebo ďalšieho testovania). Cieľom je navrhnúť prostredie, ktoré napodobňuje reálny automatizačný hardware.

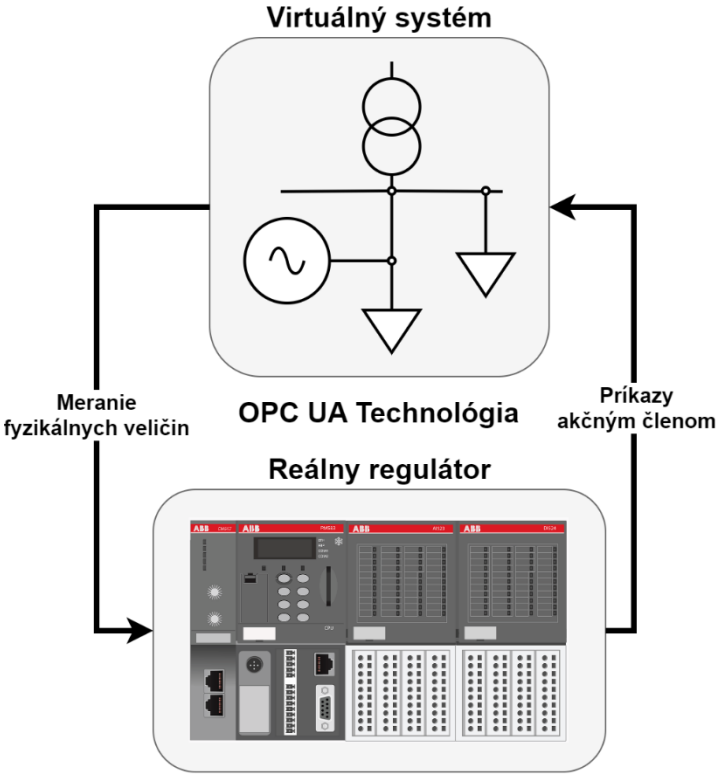
Výsledkom je simulácia prostredia výroby alebo iných procesov, ktoré sa následne využijú na overenie navrhnutých riadiacich mechanizmov pred testovaním systému v prostredí skutočného výrobného závodu, čím sa zlepší kvalita a umožní sa bezproblémový prechod z virtuálneho do fyzického prostredia.[4]

Virtuálne uvedenie do prevádzky umožňuje úplné overenie výrobného systému vykonaním koordinovanej simulácie, ktorá zahrňuje virtuálny riadený systém a skutočný regulátor. To vyžaduje, aby model virtuálneho systému bol plne opísaný na úrovni snímačov a akčných členov. Klasické uvedenie do prevádzky požaduje reálne zariadenia a regulátory, čo je veľmi nákladné, časovo náročné a býva nevyhnutné pozastavenie výrobných procesov. Významnou úlohou virtuálneho uvedenia do prevádzky je identifikácia a riešenie konštrukčných chýb a prevádzkových porúch bez reálnych zariadení a regulátorov, čím sa dosiahnu významné úspory v skutočnej implementácií do výrobného systému.

Vzhľadom na realizáciu procesov uvedenia systémov do prevádzky existujú štyri možnosti rôznych prístupov:

- Reálne uvedenie do prevádzky s reálnym systémom a reálnym regulátorom
- Virtuálne uvedenie do prevádzky, ktoré zahŕňa digitálne dvojča systému a reálny regulátor (HIL - hardware-in-the-loop)
- Uvedenie do prevádzky v reálnom prostredí so skutočným systémom a virtuálnym regulátorom (RIL - reality-in-the-loop)
- Konštruktívne uvedenie do prevádzky zahŕňa virtuálny systém a virtuálny regulátor.

Často sa technici, ktorí vykonávajú uvedenie systémov do prevádzky, zameriavajú na druhý a štvrtý prístup. Väčšina automatizovaných výrobných systémov je riadená pomocou programovateľných logických kontrolérov (PLC), čo je v súčasnosti najvhodnejšia a najrozšírenejšia priemyselná riadiaca technológia. [5]



Obr. 10: Hardware-in-the-loop (HIL).

2.2.1 Softwarový nástroj DigSilent Powerfactory

PowerFactory od spoločnosti DIgSILENT je softvérový nástroj navrhnutý pre inžiniersku analýzu elektrických systémov. Poskytuje vhodnú objektovo orientovanú platformu na modelovanie a simuláciu prenosových, distribučných, a priemyselných elektrických systémov.

Jeho prístup ku modelovaniu využíva hierarchickú schému, ktorá kombinuje grafické a skriptovacie modelovacie techniky. To zahŕňa štandardné modely a možnosť tvorby užívateľsky definovaných modelov pomocou jazyka DSL (DIgSILENT Simulation Language). Program taktiež poskytuje funkcionality, ako napríklad výpočet toku záťaže, výpočet optimálneho toku výkonu, odhad stavov, návrh ochrán, RMS/EMT simulácie a množstvo ďalších. Powerfactory tiež podporuje skupinu rozhraní s ostatnými softwarovými nástrojmi, ako napríklad Matlab alebo Python na výmenu dát a vzdialené ovládanie. Bližšie informácie o všetkých funkcionalitách obsahuje užívateľský manuál. [6]

Keďže sú energetické systémy konštruované s použitím štandardizovaných materiálov a komponentov, Powerfactory poskytuje modely komponentov, ktoré môžeme rozdeliť do skupín:

- **Terminály :** Základný prvok elektrickej siete slúžiaci na prepájanie spotrebičov, zdrojov napätia a zároveň na ňom prebiehajú rôzne výpočty podľa danej potreby, napr. výpočet veľkosti činného výkonu. V závislosti na ich použití môžu terminály predstavovať pripájacie svorkovnice, spoje alebo vnútorne uzly. Na terminály môžu

byť inštalované odpojovacie zariadenia, ako ističe alebo odpojovače. Názov objektu v PowerFactory pre tento model je *ElmTerm*.

- **Zdroje energie :** Generátory alebo zdroje elektrickej energie sú modelované ako napäťovo ovládané zdroje. Tie dodávajú elektrické napätie a výkon do systému prostredníctvom pripojenia na terminály. Powerfactory poskytuje modely pre synchrónne stroje (*ElmSym*), externá elektrická sieť(*ElmXnet*) alebo zdroje striedavého napätia(*ElmVac*)
- **Koncové komponenty:** Tieto komponenty predstavujú záťaže, transformátory alebo pasívne filtre. Záťaž (*ElmLod*) po pripojení na terminál (PQ zbernica[1]) odoberá dodanú elektrickú energiu. Rovnako môžu dodávať do siete činnú a jalovú zložku elektrického výkonu. Pasívne filtre (*ElmShnt*) sú modelované ako rezistor, induktor a kapacitorový filter alebo ich sériová kombinácia.
- **Sériové vetvy :** Elektrické komponenty (*ElmLne*), ktoré slúžia na prepojenie dvoch terminálov na vytvorenie elektrickej siete. [6]

2.2.2 RMS Simulácia

Jedným z dvoch typov simulácie v časovej doméne, ktoré PowerFactory umožňuje, je *RMS simulácia*. RMS simulácia využíva rôzne veľké integračné kroky v rozmedzí mikrosekúnd do milisekúnd alebo minút. RMS simulácie môžu byť symetrické alebo nesymetrické. Symetrická RMS simulácia berie do úvahy dynamiku elektromechanických dejov, ovládanie a tepelné vlastnosti zariadení. Využíva symetrické ustálené znázornenie pasívnej elektrickej siete, kde sa zohľadňujú iba základné zložky napätí a prúdov.

PowerFactory podporuje rôzne typy udalostí, ktoré sa môžu vyskytnúť v elektrických systémoch, ako napríklad odpájanie terminálov a transformátorov, chyby, skokové zmeny regulátorov a oveľa viac. Ďalšou dôležitou udalosťou, ktorú je potrebné spomenúť, je *udalosť parametru*. Pomocou tejto udalosti môžeme špecifikovať parameter kontraktného modelu, ktorý má byť upravený a použitý na komunikáciu s externým prostredím. Príkladom môže byť poskytnutie merania elektrických veličín a prijatie riadiaceho signálu z riadiaceho systému.[6]

2.2.3 Aplikačné rozhrania DigSilent Powerfactory

Prepojenie s ostatnými externými aplikáciami je nevyhnutné na zvládnutie zvyšnej komplexnosti elektrických systémov, ktoré interagujú s inými odvetviami, ako sú informačné alebo telekomunikačné technológie. Ďalšia možnosť, ktorá sa ponúka, je využitie už existujúcich modelov alebo fyzických komponentov, ako sú ochranné prvky alebo iné spôsoby riadenia.

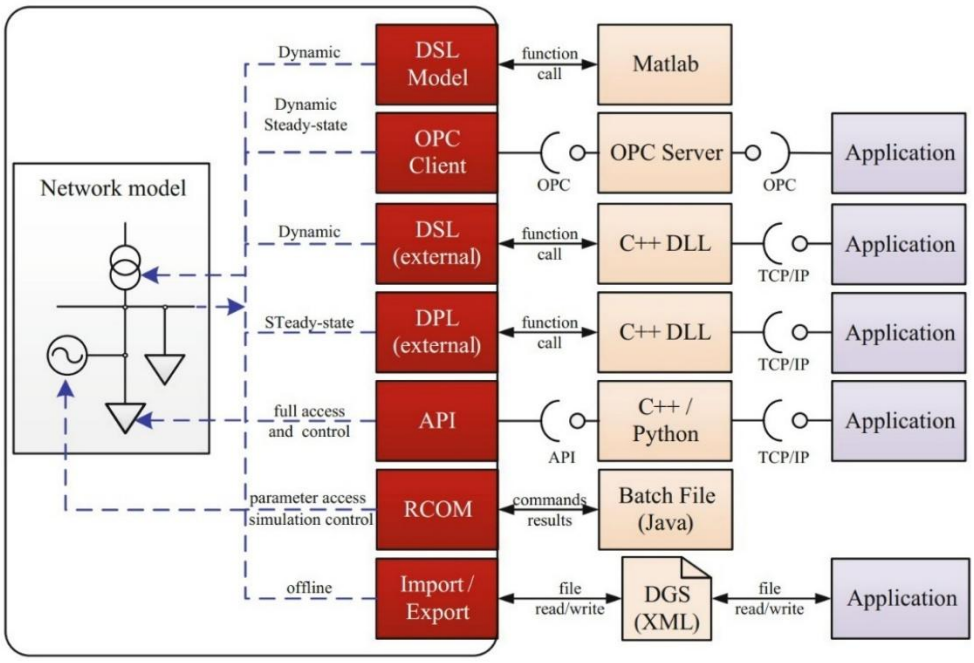
Kooperatívna simulácia s externými modelmi alebo riadením vyžaduje rozhranie, ktoré dokáže komunikovať so simuláciou v PowerFactory. Príkladom externej aplikácie

[1] Nazývame ich zbernice PQ, pretože na týchto zberniciach používame rovnice pre súčet reálneho výkon (P) a jalového výkonu (Q) a neznáme veličiny sú potom napätie a uhol napätia. [17]

môžu byť riadiaci program, ktorý je navrhnutý v prostredí ako je Matlab alebo za využitia iného programovacieho jazyka ako C++ alebo Python, ktoré možno v budnúcnosti nahrať na produkčný systém.

Existuje rôzne prístupy na prepojenie simulácií v PowerFactory s externými aplikáciami. Niektoré príklady budú v krátkosti predstavené a tie, ktoré sa využili pri návrhu kooperatívneho simulačného prostredia, budú popísané detailnejšie.

- **Matlab**: Vstavaný modul PowerFactory pre komunikáciu s prostredím Matlab.
- **DLL**: Možná integrácia externých C/C++ DLL súborov cez DSL komponenty (dostupné len pre RMS simuláciu) volaním externých funkcií (napr. pomocou TCP-IP socketov).
- **OPC-UA**: priemyselné štandardné rozhranie. Prenos dát je synchronizovaný s RMS simuláciou a jej frekvenciou aktualizácie. Často používané práve na HIL aplikácie alebo iné multiagentné systémy.
- **RCOM**: Vzdialená komunikácia (remote communication). Najčastejšie používanie na aplikácie s typológiou client-server. Tieto procedúry dovoľujú používanie PowerFactory v „Engine“ móde.
- **API**: Aplikačné rozhranie, ktoré zapuzdruje vnútorné dáta, modely a simulačné funkcie. Taktiež poskytuje priamu kontrolu nad PowerFactory, a ktorý poskytuje pokročilé funkcionality.
- **DGS**: súborový formát na prenos offline dát medzi aplikáciami. [7]



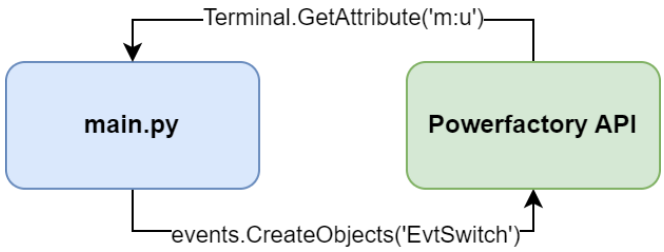
Obr. 11: Aplikačné rozhrania DigSilent PowerFactory. [4]

Python API a Powerfactory Engine mode

Dynamický Python modul je možné použiť ako rozhranie pre externé aplikácie a PowerFactory simuláciu. Rozhranie umožňuje prístup ku všetkým objektom, vlastnostiam a príkazom v PowerFactory. Obmedzeniami sú, že nie je možné meniť hodnoty premenných počas vykonávania RMS simulácie.

Jedinými predpokladmi na použitie API je kompatibilný Python interpreter a správny import tohto modulu vlastného skriptu. Python dynamický modul sa nachádza v súbore *powerfoctory. pyd*, ktorý obsahuje inštalačná zložka PowerFactory.

V Engine režime Powerfactory funguje bez grafického rozhrania a všetky jeho funkcionality sa dajú importovať do externého Python skriptu a následne ich spúšťať. Tento režim umožňuje integrovať funkcie Powerfactory do vlastného programu v jazyku Python a následne ich používať podľa potreby. Jediné obmedzenie ktoré existuje je, že nemôžu prebiehať dve inštancie programu Powerfactory, čím sa myslí súbežný beh normálneho módu s grafickým rozhraním a Engine módu. [8]



Obr. 12: Externé ovládanie RMS simulácie.

V práci bol použitý PowerFactory ako softwarová náhrada hardwaru vo virtuálnom uvedení do prevádzky HIL riadiaceho systému a Python API použitá na integrovanie do kooperatívneho simulačného prostredia, ktoré bolo vytvorené prostredníctvom frameworku Mosaik.

2.2.4 Mosaik ekosystém

Mosaik je open source, co-simulačný nástroj, ktorý pracuje v diskrétnom čase alebo v udalostiach. Bol vytvorený v programovacom jazyku Python, ale jeho aplikačné rozhranie (API) je úplne nezávisle na programovacom jazyku. Hlavným cieľom Mosaiku je použitie existujúcich simulátorov v spoločnom kontexte na vykonanie synchronizovanej simulácie.

To zahŕňa všetky simulátory alebo iné podobné nástroje ako HIL spojenie do jednej koordinovanej simulácie, kde každá súčasť tohto prostredia spustená vo vlastnom procese. Mosaik sa snaží tieto procesy synchronizovať a riadi tok dát medzi jednotlivými časťami tohto simulačného prostredia. Aj keď jednou z hlavných úloh Mosaiku je výmena dát, Masaik neobsluhuje výmenu dát v rámci jedného simulátoru.

Aby sa to všetko dalo realizovať, Masaik obsahuje tieto funkcionality:

- API pre simulátory na komunikáciu s Mosaikom.
- Implementuje rôzne obslužné programy pre rôzne druhy procesov jednotlivých simulátorov.
- Dovoľuje definovanie a modelovanie rôznych simulačných scenárov, ktoré zahrnujú rôzne simulátory.
- Pánuje postupné vykonávanie krokov rôznych simulátorov a riadi výmenu dát medzi nimi.[9]

Obr. 13: Mosaik ekosystém. [9]

Mosaik sám o sebe nezvládne skoro nič. Potrebuje simulátory, ktoré implementujú spomenuté API. Mosaik funguje ako transparentný server, ktorý riadi výmenu dát a zabezpečuje časovú synchronizáciu. To znamená, že ak jeden simulátor vyžaduje dáta z druhého simulátora v konkrétnom simulačnom čase, tak tieto dáta budú dostupné (iný simulátor postúpil v dostatočne ďaleko v čase a aby poskytol tieto údaje).

Mosaik pozostáva zo štyroch hlavných komponentov, ktoré budú rozobrané v nasledujúcich kapitolách.[10]

2.2.5 Mosaik API

Aplikačné rozhranie popisuje komunikáciu medzi Mosaikom a jednotlivými simulátormi, ktoré spája do jedného simulačného prostredia. Rozlišuje dve úrovne API, nízko úrovňová a vysoko úrovňová verzia API. Nízka úroveň používa čisté TCP sockety na výmenu správ, kódovaných do formátu JSON.

Vysoká úroveň[2] používa implementáciu nízkej úrovne API v konkrétnom programovacom jazyku. Zahrňuje všetky časti súvisiace so sieťovou komunikáciou. Zahŕňa to napríklad, obsah socketov, serializáciu, deserializáciu správ a cyklus udalostí. Vysoká úroveň obsahuje abstraktnú triedu s niekoľkými povinnými metódami, ktoré musia byť implantované pre správne fungovanie požadovaného simulátora.

[2] V súčasnej stave poskytuje Mosaik vyššiu úroveň API v programovacích jazykoch Python a Java.

Popis týchto metód je nasledovný :

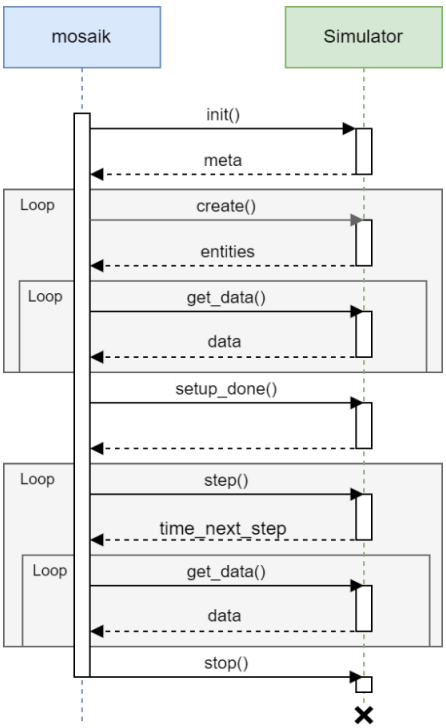
- **Init:** Inicializuje jednotlivé simulátory a vracia meta dáta s modelmi a ich atribútmi.
- **Create:** Vytvára simulačné modely a vracia list s ich unikátnymi názvami a dátovými typmi.
- **Step:** Vykoná jeden simulačný krok a vracia čas, v ktorom sa má vykonať nasledujúci krok
- **Get_data:** Vracia späť vyžadované dáta o simulačnom modelu a jeho entít.

Pri nadviazaní komunikácie medzi simulátorom a Mosaikom, Mosaik najprv zavolá metódu *init()* a predá simulátoru globálne parametre. Metóda vracia späť meta dáta, ktoré popisujú samotný simulátor. Následne môže Mosaik niekoľkokrát zavolať metódu *create(),* aby aspoň raz vytvoril simulačný model, ktorý simulátor implementuje. Na konci fázy vytvorenia a na začiatku simulačnej fázy sa zavolá metóda *setup_done().* V tomto bode sú vytvorené všetky entity simulačného prostredia, ba dokonca aj prepojenia medzi nimi.

Po ukončení metódy *create()* alebo *step()* môže Mosaik zavolať niekoľkokrát metódu *get_data(),* v ktorej Mosaik požaduje aktuálne hodnoty atribútov určitých entít.

Pri štarte simulácie Mosaik cyklicky volá metódu *step()*. To dovoľuje simulácii postúpiť dopredu v čase. Metoda *step()* vracia čas, v ktorom chce vykonať nasledujúci krok simulácie.

Na konci simulácie Mosaik pošle *stop()* správu, ktorá každej simulácií vynúti jej ukončenie. [11; 9]



Obr. 14: Aplikačné rozhranie Mosaik.

2.2.6 Popis simulačného scenára

Simulačný scenár je popis systému, ktorý plánujeme simulovať. To zahŕňa použité modely a ich vzťahy, stavy jednotlivých modelov a ich dáta. Modelovanie a zostavovanie simulačného scenára prostredníctvom Mosaiku pozostáva z troch krokov:

1. Spustenie simulátorov
2. Inštanciovanie modelov v rámci simulátorov
3. Prepojenie inštancií modelov rôznych simulátorov s cieľom vytvoriť tok dát medzi nimi.

Počiatočné nastavenia a spustenie simulátorov

Centrálna trieda na vytvorenie simulačného scenára je *mosaik.scenario.World*. Táto trieda ukladá všetky dáta a stavy patriace scenáru a kooperatívnej simulácie. Zároveň poskytuje rôzne metódy, ktoré dovoľujú spustenie simulátorov a nadviazania dátovej komunikácie medzi nimi.

Na spustenie simulácie pomocou tejto triedy musí trieda *World* vedieť, ktoré simulátory sú dostupné, ako si ich prajeme spustiť. To všetko obsahuje simulačná konfigurácia, ktorá je vysvetlená v kapitole 2.2.7.

Spustenie jednotlivých simulátorov prebieha zavolaním metódy *start(),* ktorá ako parameter požaduje jedinečný názov simulátoru. Mosaik vyhľadá tento názov v simulačnej konfigurácií a spustí daný simulátor. Táto metóda vracia objekt *ModelFactory*, ktorý dovoľuje vytvoriť inštancie modelov v rámci simulátora. V tomto kroku sa taktiež volá *init()* metóda simulátora, ktorej môže odovzdať určité parametre.

Inštanciovanie modelov simulátora

Simulátory obsahujú zoznam špecifikovaných verejne dostupných modelov a ich meta dáta. Tieto modely sú sprístupnené cez objekt *ModelFactory.* Sú tým myslené klasické Python triedy, ktoré môžeme štandartne inštanciovať a predávať im počiatočné nastavenia, ak je to potrebné.

Objekt modelu sa popisuje ako entita v rámci simulačného scenára. Každá z entít obsahuje jedinečný identifikátor v rámci simulátora a spojenie identifikátora simulátora a entita sa nazýva plný identifikátor. Plný identifikátor je globálne jedinečný.

Prepojenie entít

Aby výmena dát medzi modelmi a ich entitami vôbec bola možná, musí ich Mosaik nejakým spôsobom prepojiť. Na to slúži metóda *connect()* triedy *World*. Táto metóda prepája atribúty entít simulátorov prepojením výstupných atribútov zdrojovej entity a vstupných atribútovou cieľovej entity. Príklad takého prepojenia je nasledujúca *connect(A,B,('A_vystup','B_vstup'))*.

Toto prepojenie je však jednosmerné a nezaručuje cyklickú výmenu dát. Taktiež nie je možné jednoduchou výmenou entít zabezpečiť cyklickú výmenu dát. Existuje niekoľko spôsobov, ako to zabezpečiť, závisí to ale implementáciu nejakej stratégie riadenia simulácie. Mosaik sa zaoberá dvoma typmi výmeny dát medzi simulátormi a každý atribút môže používať iba jeden typ správania.

Typy výmeny dát sú:

1. *Hodnoty meraní*, ktoré máme v každom časovom okamihu. Príkladom sú všetky fyzikálne merania, ako napätie siete, spotreba elektrickej energie a podobne.
2. *Udalosti*, ktoré sa stanú len v určitom časovom bode. Typickým príkladom je správa o zmene požadovanej hodnoty regulátora, ktorý riadi nejaký systém.

Po spustení simulátorov, vytvorení modelov a ich prepojení je možné spustiť simuláciu s určitým konečným časom. Na to slúži metóda *run()*. Ak ide o časovú simuláciu, tá začne v čase 0 a pokračuje až do definovaného časového horizontu.[9] Bližší popis priebehu simulácie je popísaný v kapitole 2.2.7

2.2.7 Simulačný správca

Simulačný správca spúšťa a obsluhuje procesy externých simulátorov zapojených do jednej spoločnej kooperatívnej simulácie, rovnako spravuje komunikáciu s nimi.

Najčastejšie sa externé simulátory spúšťajú ako samostatné podprocesy, ktoré sa potom len pripoja do Mosaiku prostredníctvom TCP socketov. Tento prístup so sebou nesie nasledujúce výhody:

- Simulátory môžu byť vytvorené v akomkoľvek jazyku.
- Simulačné kroky sa môžu vykonávať paralelne, ak dva procesy nie sú navzájom závislé.

Spôsoby akým môžeme definovať spustenie jednotlivých simulátorov sú tieto:

- **python:** Prikáže Mosaiku spustiť simulátor v rovnakom procese. Musíme špecifikovať, ktorý Python modul a názov triedy popisuje daný simulátor.
- **cmd:** Mosaik spustí simulátor ako nový podproces. Tento spôsob dovoľuje definovať Python interpreter alebo virtuálne prostredie. Na vytvorenie prepojenia pomocou TCP socket musí simulátor poznať adresu socketu Mosaik servera. Mosaik túto adresu odovzdá v tvare *host:port* ako argument príkazového riadka.
- **connect:** prikáže Mosaiku nadviazať sieťové spojenie s už bežiacim simulátorom.

Simulačný manažér prevezme túto konfiguráciu ako argument inštancie triedy World(). Konfigurácia obsahuje názvy simulátorov a spôsob, ako ich spustiť. Konkrétny príklad sa nachádza v kapitole 3.2.4., ktorý popisuje konfiguráciu navrhnutej simulácie.[9]

2.2.8 Plánovanie a vykonávanie simulácie

Po štarte simulačného prostredia začne byť aktívny aj Mosaik plánovač. Jeho úlohou je exekúcia všetkých zahrnutých simulátorov, udržovať ich v synchronizácií a spravovať komunikáciu medzi nimi.

Plánovač vykonáva simuláciu spôsobom, že jednotlivé pripojené simulátory postúpia o krok v čase. Mosaik využíva celočíselnú reprezentáciu času. Jeho jednotky (koľko sekúnd trvá jedna simulačná jednotka kroku) môžu byť definované v inicializačnej fáze.

Podporované sú simulátory v diskrétnom čase a simulácie s diskrétnymi udalosťami a ich kombinácia. Mosaik využíva nasledovné názvoslovie: časový simulátor, simulátor založený na udalostiach a hybridný simulátor pre popis podporovaných druhov simulátorov.

Časové simulátory sú viac spojené reálnym fyzikálnym svetom, kde sú stavy, vstupy a výstupy systémov spojité (napr. činný výkon elektrického spotrebiča). Simulátory založené na udalostiach súvisia s kybernetickým svetom, kde sa stavy systému môžu okamžite meniť a vstupy s výstupmi sa vyskytujú len v určitom časovom bode.

Príkladom môže byť prijímanie a odosielanie správ v komunikáciách. Prirodzený spôsob postupu v čase potom spočíva v preskakovaní medzi všetkými vyskytujúcimi sa udalosťami. Hybridné simulátory môžu kombinovať akýkoľvek druh časových systémov a systémov založených na udalostiach.

Postup v čase

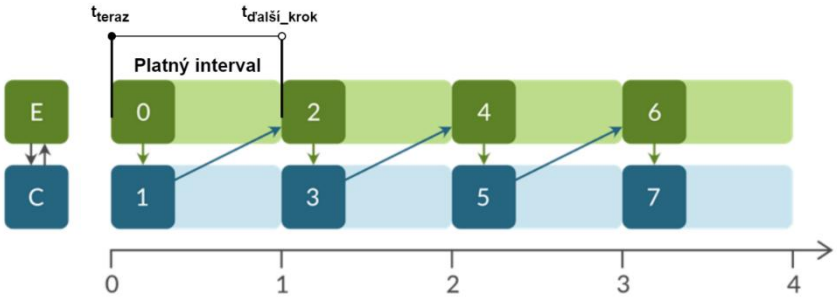
Mosaik plánovač sleduje simulačný čas pre každý simulátor individuálne. Ako simulátor postupuje v čase, záleží na type simulátoru.

Pri štarte časových a hybridných simulátorov sa simulátor nachádza v čase nula. Po žiadosti vykonať ďalší krok, odovzdá mu simulácia svoj aktuálny čas. Po ukončení kroku simulátor vráti čas, v ktorom chce vykonať nasledujúci krok. Veľkosť kroku simulátora nemusí byť konštantná, ale môže sa počas simulácie meniť podľa potreby. Údaje poskytnuté časovými simulátormi sú vypočítané počas simulačného kroku a sú platné v otvorenom intervale od aktuálneho času do času nasledujúceho kroku.

Postup v čase simulátoroch založených na udalostiach je výrazne odlišný. Tieto simulátory sú krokované vo všetkých časoch, v ktorých je udalosť vytvorená. Udalosť môže byť vytvorená prepojením atribútov alebo udalosť môže vyvolať samotný simulátor. Výstup týchto simulátorov je platný len pre určitý časový okamih, zvyčajne pre aktuálny čas kroku alebo pre akýkoľvek neskorší čas, ak je to explicitne definované. Ďalšou vlastnosťou výstupov je, že ich poskytovanie nie je povinné. Dôsledok toho je, že simulátor prepojením k určitému atribútu spustí krok len vtedy, ak je výstup skutočné poskytnutý.

Cyklická výmena dát

V niektorých simulačných aplikáciách simulovaný systém vyžaduje cyklickú výmenu dát medzi komponentmi. Príkladom môže byť regulačný mechanizmus (C) a iný systém (E), ktorý je ním ovládaný na základe jeho stavu. Nie je možné vykonávať obe dátové výmeny súčasne, pretože sú navzájom závislé. Tento problém sa vyrieši spôsobom, že najprv sa vykoná krok regulovaného systému. Stav regulovaného systému následne využije regulátor ako vstup pre svoj simulačný krok v tom istom platnom časovom intervale. Príkazy, ktoré regulátor vygeneruje, sa potom použijú v ďalšom kroku regulovaného Systému. Výsledkom je sériové vykonávanie, ktoré sa tiež nazýva *Gaussová-Siedelova schéma* [9].



Obr. 15: Schéma cyklickej výmeny dát.

Popis reálneho príkladu tohto cyklu, ktorý funguje v skutočnom svete je, že riadiaca jednotka by merala údaje z riadeného systému v určitom časovom okamihu *t*. Potom by vykonala určitý výpočet, ktorý by trval nejaký čas Δt. Výpočet s príkazmi by sa odoslal do riadeného systému v okamihu $t+\Delta t$.

V simulačnom prostredí Mosaik sa môže tento výsledok dostať pomocou časovo posunutého spojenia. Spojenie hovorí Mosaiku, že výstup regulátora sa má použiť v nasledujúcom kroku regulovaného systému. Keďže v prvom kroku simulácie regulovaný systém nemôže poskytnúť merané údaje regulátoru, musíme mu nastaviť nejaké počiatočné údaje. Druhým spôsobom je umožniť asynchrónne požiadavky pripojenia a použiť asynchrónne spätné volanie metódy *set_data()*, v implementácií kroku regulátora na odosielanie príkazov do regulovaného systému. Výhodou tohto prístupu je, že volanie metódy *set_data()* je nepovinné, čiže príkazy regulátora sa nemusia odosielať pri každom kroku.[9]

3 VLASTNÉ RIEŠENIE VIRTUÁLNEHO UVEDENIA DO PREVÁDZKY

Na začiatku kapitoly je predstavený elektrický systém, jeho vlastnosti a požiadavky na reguláciu. Na tento elektrický systém bola vypracovaná a dodaná prípadová štúdia v programe DigSilent PowerFactory technickým špecialistom spoločnosti ABB. Hlavným cieľom diplomovej práce bolo tento systém virtuálne uviesť do prevádzky.

To zahŕňalo úpravy prípadovej štúdie tak, aby bola schopná kooperatívnej simulácie, vytvorenie kooperatívneho simulačného prostredia a návrh regulačného programu, ktorý bude podľa požiadaviek riadiť daný systém.

Kooperatívne simulačné prostredie je postavené na ekosystéme Mosaik. Na druhej strane simulovaného systému je reálny regulátor vo forme PLC. Každému zo spomenutých častí je venovaná samostatná kapitola, ktorá popisuje detailný popis a realizáciu.

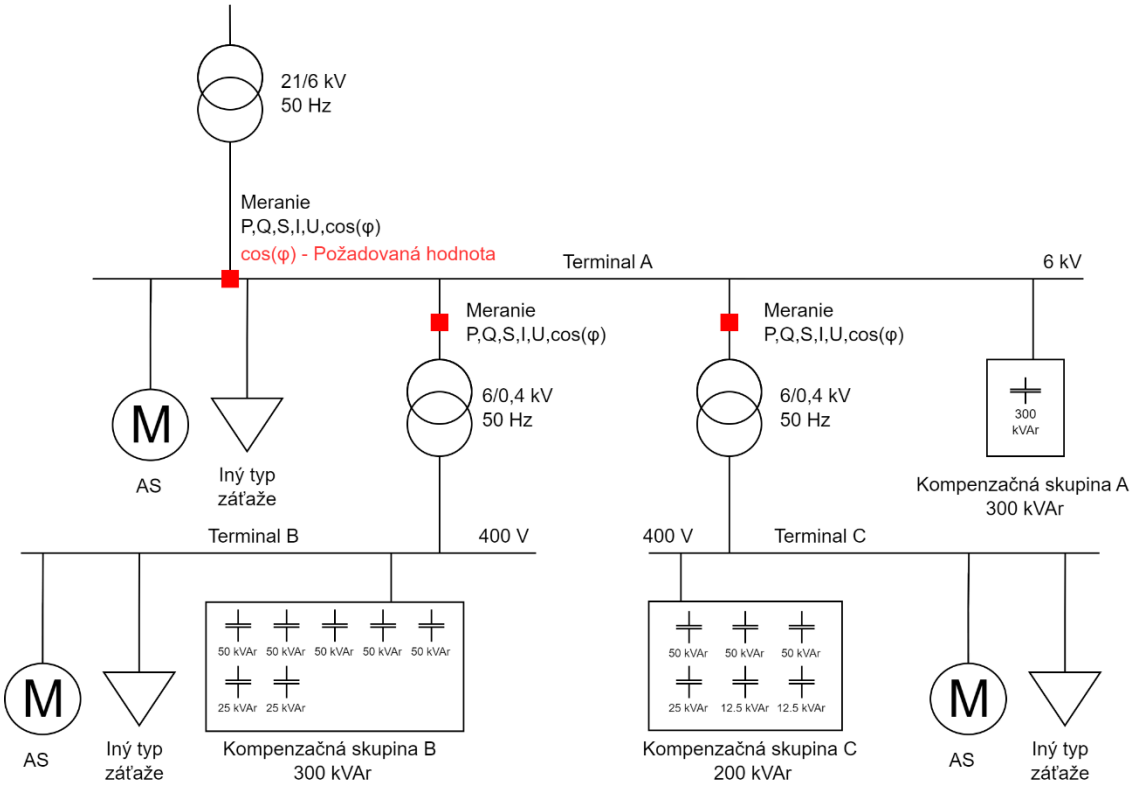
3.1 Popis prípadovej štúdie

Elektrický systém pozostáva z jedného hlavného terminálu a dvoch podružných terminálov. Schéma elektrického systému je zobrazená na obrázku 16. Pre jednoduchosť sme ich označili *terminál A, B* a *C*. Terminál A je pripojený na distribučnú elektrickú sieť pomocou napájacieho transformátora 21/6 kV. V mieste prívodu na *terminál A* sú merané nasledovné elektrické veličiny, činný výkon, jalový výkon, zdanlivý výkon, napätie, prúd a účinník. Tieto merané veličiny budú použité na výpočet potrebného kompenzačného jalového výkonu k regulácií účinníku na požadovanú hodnotu. Transformátor napája niekoľko motorov, dva transformátory 6/0,4 kV, ktoré napájajú podružné rozvádzače. Taktiež je k nemu prepojený jeden kompenzačný stupeň s veľkosťou 300kVar. Podmienky jeho pripojenia budú vysvetlené ďalej v kapitole.

Na primárnych vinutiach oboch transformátorov 6/0,4 kV bude prebiehať meranie elektrických veličín rovnakých ako na napájacom transformátore. Transformátory napájajú *terminály B* a *C*, na ktorých sú napojené rôzne typy záťaží a sú k ním pripojené kompenzačné skupiny s hodnotami 300 kVar (5x50 kVar a 2x25 kVar) a 200 kVar (3x50 kVar,1x25 kVar a 2x12,5 kVar). Požiadavky na reguláciu predstaveného systému môžeme formulovať takto:

- Reguluje sa na požadovanú hodnotu účinníku v mieste prívodu elektrickej energie na terminál A.
- Možnosť voľby účinníku v rozsahu $0{,}7_C$ a $0{,}7_L$.
- Optimálne rozdelenie potrebného kompenzačného výkonu medzi kompenzačné skupiny *B* a *C* tak, aby hodnoty účinníkov na termináli *B* a *C* mali čo najmenší rozdiel.

- Optimálny výber kompenzačných stupňov v rámci skupiny tak, aby boli rovnomerne zaťažené.
- Kompenzačný stupeň na termináli A má byť pripojený iba v prípade, ak regulácia vyžaduje viac ako 300 kVar, v opačnom prípade slúžia na dodanie jalového výkonu stupne na termináli *B* a *C*.



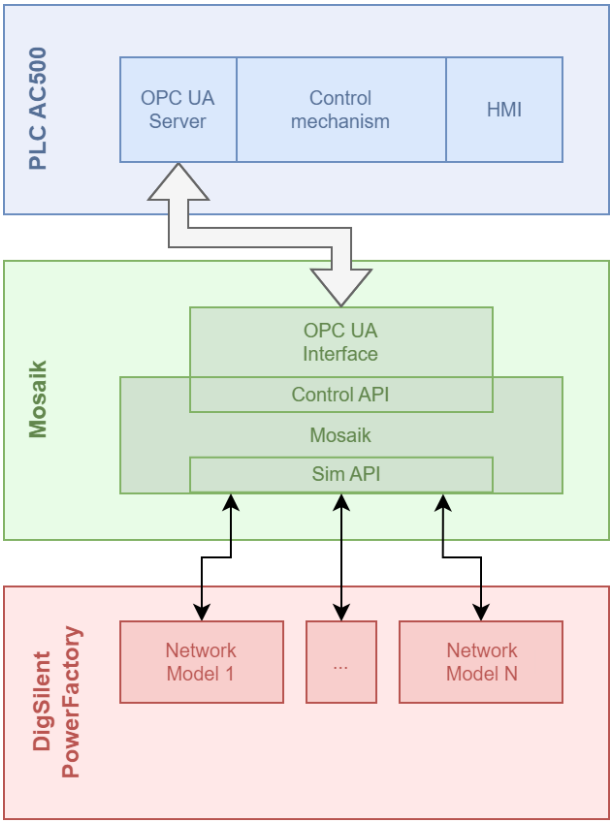
Obr. 16: Schéma regulovaného elektrického systému.

3.2 Návrh simulačného prostredia

Simulačné prostredie je postavené na ekosystéme Mosaik, predstavenom v kapitole 2.2.4. V Mosaik terminológií sa každý pripojený model simulovaného systému nazýva ako simulátor. V tomto prípade kooperatívne simulačné prostredie pozostáva z troch simulátorov a samotného jadra Mosaiku.

Prvým je PowerFactory simulátor, ktorý simuluje elektrický systém, ktorý plánujeme regulovať. Druhým simulátorom je OPC UA klient, ktorý je pripojený k OPC UA serveru bežiacom na PLC. V každom kroku kooperatívnej simulácie sú čítané dáta z PowerFactory simulátoru a po vyhodnotení dát sa v nasledujúcom kroku simulácie pošlú príkazy simulovaným akčným členom (stýkačom).

Tretím je jednoduchý kolektor dát, ktorý slúži na ukladanie CSV dát z regulovaného systému. Tieto dáta môžu byť následne uložené alebo zobrazené vo forme grafov pre ich hlbšiu analýzu.



Obr. 17: Architektúra kooperatívnej simulácie.

3.2.1 PowerFactory simulátor

Návrh integrácie softwaru DigSilent PowerFactory do ekosystému Mosaik vychádza z návrhu uvedenom v zdroji [11]. Pri predstavení softwaru DigSilent PowerFactory bolo spomenuté, že obsahuje aplikačné rozhranie Python, ktoré umožňuje externé ovládanie modelov a príkazov simulácie. V práci sa toto aplikačné rozhranie použilo na integrovanie do ekosystému Mosaik.

Úprava RMS simulácie

Pôvodnú vypracovanú štúdiu bolo potrebné upraviť v grafickom rozhraní softwaru DigSilent PowerFactory tak, aby sa nám v Engine režime ľahšie pracovalo. Všetkým potrebným modelom simulácie bol priradený prefix *Ext_* , aby sme neskôr podľa toho prefixu mohli filtrovať modely. Ďalej bolo nevyhnutné pridať externé merania fyzikálnych veličín pomocou objektu StaExtXmea, kde za X môžeme dosadiť typ merania. Napríklad StaExtpmea je objekt, ktorý slúži na externé meranie činného výkonu a obsahuje atribúty ako vypočítanú hodnotu merania (*e:Pcal*) alebo uzol, kde meranie prebiehalo (*e:pcubic*). Miesta merania sme sa snažili napodobniť reálnemu umiestneniu sieťového analyzátora. Miesta merania sú zobrazené na obrázku 16 červeným štvorcom.

Rovnako bol identifikátor každého odpojovacieho prvku terminálu plánovaného v kooperatívnej simulácií premenovaný tak, aby začínal reťazcom *Ext* pre jednoduchú manipuláciu.

Identifikátory stýkačov, ktoré budú spínať kompenzačné stupne, boli upravené spôsobom pre jednoduché prepojenie s odpovedajúcim prvkom poľa v programe PLC. Príkladom je stýkač v kompenzačnej skupine A *TerminalA_Compensation[1]*. Na odpojovacie prvky nebolo potrebné pridať žiadne ďalšie objekty, pretože objekt *StaSwitch*, ktorý predstavuje odpojovací prvok obsahuje všetky potrebné atribúty, ako polohu prvku (*e:isclosed*) alebo príkaz na spínanie (*e:on_off*).

V tomto bode sme mali simuláciu upravenú pre jednoduchú prácu v Engine režime. Posledným krokom je import dynamického python modulu PowerFactory, aby sme s ním mohli pracovať v externom Python skripte.

Simulačné aplikačné rozhranie

Navrhnutý PowerFactory simulátor pozostáva z tried, ktoré sú zobrazené na obrázku 18. Každá trieda simulátoru v ekosystéme Mosaik musí dediť z abstraktnej triedy Simulator, ktorý je poskytovaný Mosaikom. Simulátor musí implementovať štyri povinné metódy. *Init(), create(), step()* a *get_data()*. Poskytuje aj iné metódy, ale tie nie sú povinné. Náš RMS PowerFactory simulátor pozostáva z dvoch tried.

Prvou je abstraktná trieda PF_Sim, ktorej úlohou je základné ovládanie softwaru PowerFactory v „Engine“ režime. Tento simulátor je definovaný ako hybridný, pretože kombinuje časové správanie systémov merania fyzikálnych veličín a systémov založených na udalostiach (príkaz na zopnutie kompenzačného stupňa). Bližší popis podporovaných typov simulátorov sa nachádza v kapitole 2.2.8. Implantácia metód simulátora a ich fungovanie je nasledovné.

Init()

V tejto metóde prebieha spustenie software DigSilent PowerFactory v Engine režime. Predaním nastavení ako *názov projektu* a *názov štúdie* program vie, aký projekt a štúdiu má aktivovať. V prípade, ak štúdia nemá nastavený počiatočný čas, je tento čas nastavený na počiatočnú hodnotu. V tomto kroku sa tiež získajú objekty príkazov RMS simulácie, ako napríklad pozastavenie časovej simulácie (*ComSim*) alebo príkaz na inicializáciu počiatočných podmienok RMS simulácie (*ComInc*).

Následne sa preskenuje celý PowerFactory projekt pre všetky relevantné objekty. Tieto objekty a ich atribúty následne pridá do slovníka meta dát simulátora, aby tieto objekty boli prístupné v ekosystéme Mosaik.

Create()

Metóda *create()* mapuje všetky prvky objektu *ElmNet*. *ElmNet* je koreňom celej hierarchie modelu elektrického systému v PowerFactory a predstavuje elektrickú sieť.

Všetky ostatné prvky sú potomkami objektu *ElmNet*. Táto metóda nevytvára nové prvky, pretože prvky elektrickej siete sú už definované v PowerFactory. Z tohto dôvodu je možné inštanciovať len objekt *ElmNet*. Metóda vracia všetky relevantné prvky elektrickej siete definovanej v PowerFactori ako potomkov ElmNet vo vrátenom zozname entít.

Step()
Vstupným parametrom tejto metódy sú vstupné dáta pre entity elektrickej siete. Vstupné dáta sú dostupné iba v aktuálnom kroku simulácie. Po zapísaní nových dát sa zavolá abstraktná metóda *run_step()*, aby vykonala určité výpočty alebo vytvorila simulačné udalosti, ktoré ovplyvnia simuláciu. Potom sa spustí RMS simulácia, až kým sa neskončí krok simulácie.

Abstraktná metóda *run_step()* musí byť definovaná v dcérskej triede. V našom prípade v triede RMS_Sim. Metóda prinavracia relatívny simulačný čas, v ktorom sa má vykonať nasledujúci simulačný krok.

Get_data()
Metóda vracia požadované atribúty prvkov elektrickej siete vytvorenej v PowerFactory.

Finalize()
Táto metóda je volaná na konci kooperatívnej simulácie a slúži na resetovanie určitých nastavení do pôvodných hodnôt. Príkladom je zmena hodnoty času štúdie do jej pôvodnej hodnoty pred spustením simulácie.

Druhou triedou je trieda RMS_Sim. Jedná sa o potomka abstraktnej triedy PF_Sim a dedí jej základné funkcie a rozširuje ju o funkcie RMS simulácie. Práve tento druh simulácie sme využili v kooperatívnom simulačnom prostredí Mosaik. RMS simulácia umožňuje simuláciu v diskrétnej časovej doméne.

Na ovládanie simulácie a jej modelov boli implementované alebo rozšírené tieto metódy:

Init()
Okrem vykonina všetkých rutín, ktoré sú popísané v init() metóde rodičovskej triedy PF_Sim, sa táto metóda rozširuje o tieto nutné funkcie.

Na začiatku RMS simulácie je nutné definovať počiatočné hodnoty nastavení, ako napríklad veľkosť simulačného kroku, typ simulácie alebo čas v ktorom má simulácia začať. Tieto parametre sa predajú príkazu *ComInc*, ktorý slúži na inicializáciu počiatočných podmienok RMS simulácie.

Taktiež je nutné získať objekt udalosti *IntEvt*. Objekt obsahuje všetky udalosti, ktoré sú vytvorené a vykonané v danej simulácií. Udalosti vytvorené v našom simulačnom prostredí pri inicializácií simulátora sú vymazané, aby nová simulácia neobsahovala udalosti z predošlej simulácie.

Step()
Vstupné dáta, ktoré metóda prevezme, sa nastavia patričným entitám, vytvorením simulačných udalostí ktoré vykonajú aktualizáciu dát. V tomto prípade je pre každý odpojovací prvok vytvorená udalosť *EvtSwitch*. Udalostiam sú nastavené atribúty, ako názov prvku, jeho nová poloha alebo čas vykonania udalosti.

Po vytvorení udalosti sa spustí RMS simulácia a vytvorené udalosti sa aktivujú. RMS simulácia je ukončená v momente, keď je daný simulačný krok kooperatívneho

prostredia neplatný. Tento cyklus vytvárania nových udalostí pracuje až kým nie je ukončená kooperatívna simulácia. Metóda vracia relatívny simulačný čas, v ktorom sa má vykonať nasledujúci simulačný krok. Povinné metódy, ktoré neboli spomenuté, ostávajú nezmenené.

3.2.2 OPC UA klient simulátor

Protokol OPC UA (Open Platform Communications Unified Architecture) je štandardizovaný komunikačný protokol, navrhnutý pre spoľahlivú a bezpečnú výmenu dát medzi rôznymi systémami v priemyselných automatizačných aplikáciách. Je nezávislý od platformy a programovacieho jazyka a umožňuje prepojenie medzi rôznymi systémami. Celý popis protokolu sa nachádza v jeho dokumentácií [12].

Na vytvorenie OPC UA klienta bola použitá python knižnica *opcua-asyncio*. Knižnica ponúka nízkoúrovňové rozhranie na odosielanie a prijímanie všetkých definovaných UA štruktúr a vysokoúrovňové triedy, ktoré umožňujú napísať vlastný server alebo klienta v krátkom čase. Bližší popis obsahuje online dokumentácia [13] .

Simulátor slúži ako prostredník medzi PLC regulátorom a simuláciou PowerFactory. Na PLC beží OPC UA server, ku ktorému sa OPC UA klient ako simulátor pripája v inicializačnej fáze. Klient prijíma dáta cez Mosaik z PowerFactory a následne ich posiela PLC systému. Komunikácia funguje aj opačným smerom. Ak regulátor vyhodnotí, že je potrebné zopnúť určitý stýkač, PLC pošle príkaz na zopnutie stykača OPC UA klientovi. Klient ho následne predá Mosaiku. Mosaik tieto dáta odovzdá PowerFactory simulátoru ako vstupné dáta metódy *step()*, aby sa vykonala zmena v simulovanom systéme. Architektúra komunikácie je znázornená na obrázku 17. OPC UA klient je implementovaný ako simulátor bežiaci na udalostiach v prostredí Mosaik, a preto musí byť podtriedou abstraktnej triedy Simulator.

Zároveň musí implementovať už spomenuté povinné triedy. Implantácia metód simulátora a ich fungovanie je nasledovné:

Init()

Metóda *init()* prevezme a nastaví počiatočné nastavenia simulátora. Ako napríklad URL adresu OPC UA servera alebo veľkosť simulačného kroku. Následne je vytvorená inštancia synchrónnej verzie triedy *Client* . OPC UA klient sa potom pripojí na OPC UA server. Ako každý simulátor v ekosystéme Mosaik, aj táto metóda vracia meta dáta simulátora.

Create()

V tomto kroku je vytvorená v simulovanom prostredí jediná entita *OPC_UA_Client*. Aby nedošlo ku chybám pripojenia, je možné inštancovať iba jedno OPC UA rozhranie.

Na konci nastaví OPC UA uzol *MosaikStatus.Connected* na logickú hodnotu jedna, aby PLC systém vedel, že simulátor funguje. Metóda na konci vracia objekt klienta, kvôli dostupnosti v simulovanom prostredí.

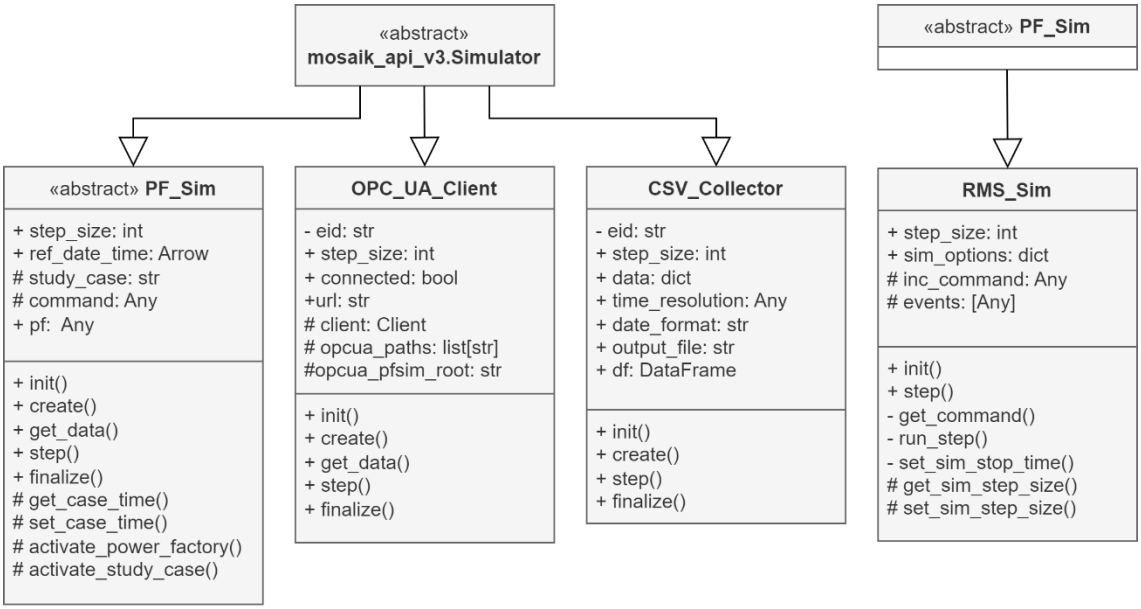
Step()
V každom simulačnom kroku simulátor prevezme aktuálne dáta z PowerFactory simulácie. Tieto sú následne zapísané do patričných uzlov OPC UA servera. Metóda nakoniec vracia relatívny simulačný čas, v ktorom sa má vykonať nasledujúci simulačný krok.

Get_data()
Metóda vracia všetky požadované hodnoty OPC UA uzlov a ich názvy vo forme Python slovníka. Tieto dáta sú potom dostupné v ekosystéme Mosaik. Týmto spôsobom sú aj predané dáta regulátora PowerFactory simulácií.

Finalize()
Pri ukončení kooperatívnej simulácie, simulátor pošle PLC systému informáciu o statuse pripojenia a následne sa klient ako simulátor odpojí od OPC UA servera bežiacom na PLC.

3.2.3 CSV Kolektor dát

Kolektor dát je pomocný simulátor, ktorý slúži na ukladanie všetkých simulačných dát (fyzikálne merania, polohy stýkačov, adť) do súboru CSV. Je veľmi podobný simulátoru OPC UA klient, len s tým rozdielom, že dáta nie sú komunikované cez OPC UA protokol ale sú ukladané do CSV súboru. Každý riadok predstavuje jeden simulačný krok a jeden stĺpec predstavuje hodnotu atribútu jednej entity v danom simulačnom kroku. Tieto uložené dáta slúžia sú spracovane knižnicami *pandas*[3] a *matplotlib*[4], ktoré slúžia na dátovú analýzu a vizualizáciu dát.



Obr. 18: Triedny diagram kooperatívneho simulačného prostredia

[3] Online dokumentácia ku knižnici pandas je dostupná na odkaze [18]
[4] Online dokumentácia ku knižnici matplotlib je dostupná na odkaze [19].

3.2.4 Simulačný scenár

Konfigurácia simulačného sveta alebo scenára obsahuje zoznam dostupných simulátorov a spôsob akým ho chceme zapnúť. Každý simulátor sa spúšťa v samostatnom procese.

Cez zástupné symboly *%(python)s* a *%(addrs)s* je simulátorom oznámené aký Python interpreter majú použiť, a aká je IP adresa aktuálneho TCP socketa, aby vedeli kam sa pripojiť. Manažment komunikácie potom prevezme Mosaik a nie je potrebné sa s ňou už ďalej zaoberať. Spomenuté parametre sa predávajú ako argumenty skriptu pri jeho spustení. Konfigurácia simulačného prostredia je aplikovaná na priloženom obrázku.

```
SIM_CONFIG = {
      'RMS_Sim': {
          'cmd': '%(python)s mosaik_pf//rms_sim.py %(addr)s',
      },
       'OPC_UA_Collector': {
          'cmd': '%(python)s mosaik_pf//opc_ua_collector.py %(addr)s',
      },
      'Collector': {
      'cmd': '%(python)s mosaik_pf//collector.py %(addr)s',
      }
  }
```

Obr. 19: Konfigurácia kooperatívneho simulačného prostredia.

Na začiatku skriptu sú definované konštanty, ktoré sa opakujú v celom skripte, ako napríklad dĺžka trvania celej kooperatívnej simulácie v sekundách.

Predaním simulačnej konfigurácie triede *World()* a jej inštanciou je vytvorené simulačné prostredie. Ďalším krokom bolo spustenie simulátorov v rámci celého simulačného prostredia. Jednotlivým simulátorom boli priradené určité počiatočné nastavania, ktoré sú vysvetlené v danej kapitole. Po spustení simulátorov nastalo filtrovanie potrebných entít, ich vytvorenie a prepojenie medzi sebou. Z PowerFactory RMS simulácie boli vyfiltrované potrebné entity. Zo simulácie elektrického systému nás hlavne zaujímali merania fyzikálnych veličín, poloha stýkačov a príkazy spínanie.

Jednotlivé merania sú v PowerFactory simulácii reprezentované objektom (*StaExtXmea*). Tieto merania sú jednosmerne pripojené k simulátoru klienta OPC UA cez pomocnú funkciu *connect_many_to_one()*. Tejto funkcií boli predané parametre v tomto poradí. Prvým je simulačný svet, druhým je zoznam zdrojových entít simulátora, tretím je entita cieľového simulátora a posledným je atribút zdrojovej entity, ktorú chceme predať cieľovému simulátoru. Príkladom je *connect_many_to_one(world,Pmeas, opc_ua_client, 'e:Pcal')*. Týmto spôsobom bolo vytvorené spojenie medzi entitami simulátoru PowerFactory a simulátoru OPC UA klienta.

Aby regulačná slučka mala zmysel a náš navrhnutý regulátor bol schopný spínať dané stýkače, bolo nutné vytvoriť cyklické prepojenie dát medzi príkazmi, spätnými väzbami regulátora a simulovanými stýkačmi. Toto prepojenie prebiehalo v dvoch krokoch. Prvým bolo prepojenie všetkých entít *StaSwitch*, ktoré patria PowerFactory

simulátoru a jej atribút *e:isclosed* s entitou *opc_ua_client* simulátora OPC UA Klient. Atribút *e:isclosed* obsahuje informáciu o aktuálnej polohe daného stýkača.

V druhom kroku sa prepojili príkazy regulátora k spínaniu daného stýkača k patričnému stýkaču v simulácii PowerFactory. Na spínanie stýkačov v simulácii PowerFactory slúži atribút *e:on_off* entity *StaSwitch*. Pre správne fungovanie cyklickej výmeny dát bolo nevyhnutné uviesť počiatočné polohy stýkačov a povoliť časový posun simulátorov pomocou parametru *time_shifted=true*. Dôvod tohto nastavenia je objasnený v kapitole 2.2.8 v sekcii cyklická výmena dát.

V tomto bode boli definované všetky potrebné vzťahy medzi entitami simulátorov a celé kooperatívne simulačné prostredie sa mohlo spustiť.

3.3 Návrh riadiaceho programu

V navrhnutom virtuálnom uvedení do prevádzky bude regulátor vo forme PLC systému rady AC500 V3 spoločnosti ABB. Táto rada PLC systémov poskytuje spoľahlivú a výkonnú platformu na vytváranie škálovateľných a flexibilných automatizačných riešení. Poskytuje veľkú škálu komunikačných protokolov, ako napríklad Modbus, Profinet alebo nami využívaný protokol OPC UA. podporuje najnovšiu verziu piatich programovacích jazykov normy IEC 61131-3, ktorá zahŕňa objektovo orientovaný prístup vývoja aplikácií. Vývoj aplikácií prebieha v prostredí programu *Automation Builder*, ktorý je postavený na open source platforme *Codesys*.[14]

Obr. 20: PLC AC500 V3. [14]

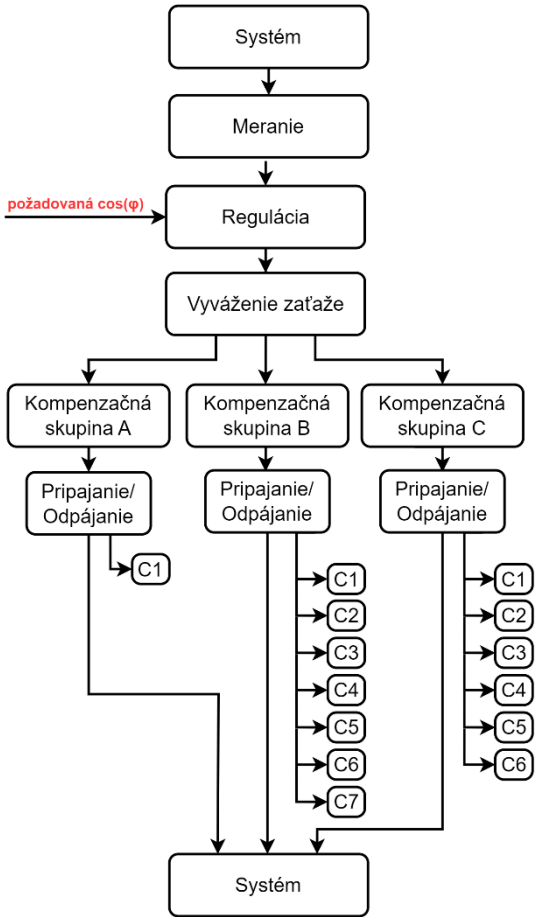
Navrhnutý riadiaci program bude bežať na PLC AC500 a vychádza z objektovo orientovaného programovania. Táto paradigma bola zvolená hlavne preto, pretože sme pri návrhu objektov kládli dôraz na ich znova použiteľnosť a funkčnosť.

Na začiatku návrhu sa analyzovali jednotlivé komponenty systému. Pre každý komponent systému bol vytvorený samostatný funkčný blok a grafický objekt. Realizácia funkčných blokov a programových jednotiek prebiehala v dvoch vrstvách. Prvú, nižšiu vrstvu popisuje vytvorená knižnica funkčných blokov. Funkčné bloky boli

naprogramované v jazyku ST (Structured text). Na vyššej vrstve sú vytvorené funkčné bloky prepojené prostredníctvom grafického jazyka FBD (Function Block Diagram). Tento jazyk bol zvolený hlavne pre svoju prehľadnosť a názornosť.

Hlavný cyklus regulácie pozostáva z troch hlavných častí. Schéma regulačného cyklu je znázornená na obrázku 21. Prvou časťou je regulátor, ktorý z meraných fyzikálnych veličín vyhodnotí či je nutný zásah do regulovaného systému, a ak áno, koľko jalového výkonu treba dodať poprípade odobrať.

Požadovaná hodnota jalového výkonu je následne predaná funkčnému bloku s názvom *vyváženie záťaže*. Tento funkčný blok slúži na optimálne prerozdelenie požadovanej hodnoty jalového výkonu medzi podružné termináli B a C a kompenzačný stupeň na termináli A . Popis algoritmu je vysvetlený v kapitole 3.3.7. Po optimálnom rozdelení požadovanej hodnoty jalového výkonu je riešenie odovzdané funkčnému bloku kompenzačná skupina.



Obr. 21: Schéma riadiaceho programu.

Funkčný blok kompenzačná skupina obsahuje priradené kompenzačné stupne. Na základe vstupnej hodnoty požadovaného jalového výkonu zostaví z dostupných kompenzačných stupňov spínaciu sekvenciu, a potom pošle príkazy funkčným blokom, ktoré ovládajú jednotlivé kompenzačné stupne. Spôsob výberu optimálnych kompenzačných stupňov je vysvetlený v kapitole venovanej kompenzačnej skupine.

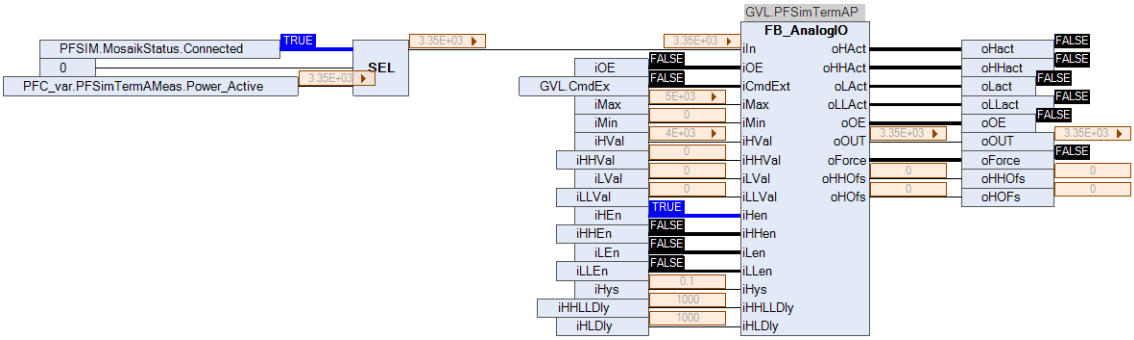
Pri tvorbe práce bolo navrhnutých niekoľko funkčných blokov a ich grafická reprezentácia. Navrhnuté funkčné bloky sú tieto:

3.3.1 Analógový vstup a výstup

Funkčný blok je navrhnutý na spracovanie analógových signálov z meracích prístrojov. Na vstupe je očakávaný reálny fyzikálny rozmer veličín a nie nespracovaná hodnota z analógovej karty. Blok obsahuje definovateľný rozsah merania, prepisovanie výstupnej hodnoty a signalizácia rôznych stavov.

Prepisovanie výstupnej hodnoty znamená, že v prípade nedostupného meracieho zariadenia vieme vynútiť výstupnú hodnotu merania, a potom ju používať v riadiacom programe . Po dosiahnutí definovateľných limitov merania, funkčný blok signalizuje štyri úrovne alarmov (dve nízke úrovne a dve vysoké úrovne). Súčasne sú pripravené vstupy na povolenie týchto alarmov, oneskorenie vyvolania alarmov, chyby meracieho zariadenia a príkaz na resetovanie chybových hlášok.

Všetky spomenuté vlastnosti funkčného bloku sú taktiež dostupné v navrhnutom HMI a grafickom okne, ktoré slúži na ovládanie analógového vstupu a výstupu .



Obr. 22: Analógový vstup pre činný výkon na termináli A.

3.3.2 Analógová požadovaná hodnota

Navrhnutý funkčný blok slúži na nastavovanie analógových požadovaných hodnôt, akými je napríklad požadovaná hodnota účinníku. Obsahuje definovateľný povolený rozsah, automatický a manuálny režim. Definovateľný rozsah je nastavený v programe a nie je možné ho meniť prostredníctvom HMI.

V automatickom režime je požadovaná hodnota vzatá z programu a operátor HMI ju nemôže meniť. Operátor môže požadovanú hodnotu meniť iba v manuálom režime. Aby nedošlo k neúmyselnej zmene požadovanej hodnoty, operátor musí zapisovať novú hodnotu príkazom na to určeným. Zároveň bola vytvorená grafická reprezentácia analógovej hodnoty vo forme dialógového okna, kde môže operátor sledovať a meniť spomenuté vlastnosti funkčného bloku.

3.3.3 Digitálny vstup a výstup

Funkčný blok slúži na spracovanie digitálnych vstupných signálov, alebo sa dá použiť na binárne príkazy ostatným častiam programu. Funkčný blok je tiež súčasťou ďalších funkčných blokov, akým je napríklad stýkač, kde je použitý ako vstup aktuálnej polohy stýkača. Obsahuje taktiež funkciu prepisovania výstupnej hodnoty a signalizáciu rôznych stavov.

Digitálny vstup a výstup možno nakonfigurovať spôsobom aby pri logickej hodnote *TRUE* bol vyvolaný alarm, ktorý môže byť vo forme chyby, varovania alebo udalosti.

3.3.4 Stýkač

Funkčný blok je navrhnutý primárne na ovládanie elektro-mechanického prvku stýkača, ale dá sa použiť na ovládanie hocijakého iného prvku, ktorý bude mať podobné správanie. Obsahuje vstupy pre spätnú väzbu stýkačov, čo je vlastne aktuálna poloha stýkača.

Okrem toho má aj viacero konfiguračných vstupov, ako povolenie chybovej hlášky pre chybu spätnej väzby. Napríklad, ak po príkaze *zatvoriť kontakt* nepríde signál spätnej väzby do piatich sekúnd, vyvolá spomenutú chybu. Dostupné sú aj vstupy pre chybové signály zariadenia.

Funkčný blok umožňuje ovládať stýkače v dvoch režimoch. V automatickému režime funkčný blok používa vstupy bloku pre príkaz *otvoriť kontakt* alebo *zatvoriť*. Tieto príkazy sú prevzaté z nadradenj logiky programu a príkazy z HMI sú ignorované. Pri zmene režimu do manuálu sa stane stýkač nedostupným pre automatický režim a operátor ho môže ľubovoľne spínať príkazmi cez grafický objekt v HMI. Obsahuje taktiež funkciu počítadla zopnutí a počítadlo prevádzkových hodín.

V aplikácii bol navrhnutý funkčný blok použitý na ovládanie hlavných odpojovacích prvkov každého terminálu, a v kombinácii s funkčným blokom *kompenzačný stupeň* ako programová jednotka na ovládanie simulovaného kompenzačného stupňa alebo jeho reálne podoby.

3.3.5 Kompenzačný stupeň

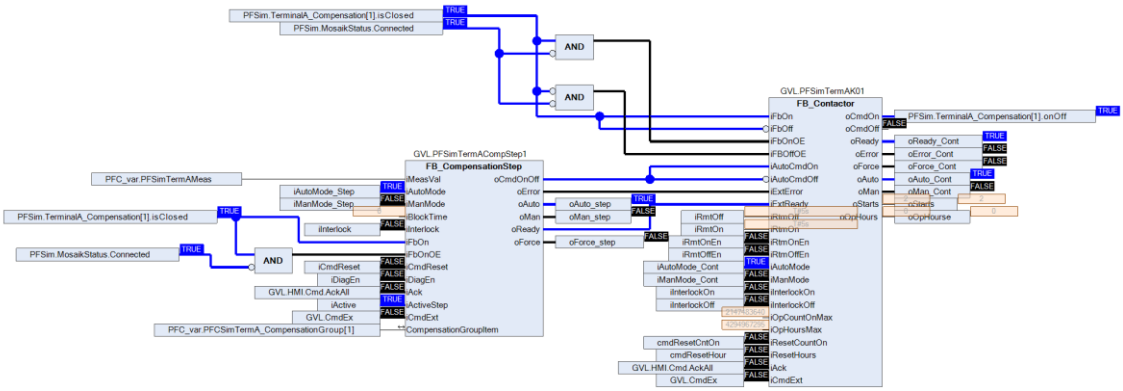
V kombinácii s funkčným blokom *stýkač* slúži tento funkčný blok na ovládanie jednotlivých kompenzačných stupňov. Každému funkčnému bloku je pridelený prvok poľa, ktorý predstavuje jeden kompenzačný stupeň. Celé pole je jedna kompenzačná skupina. Prvky poľa obsahujú informácie ako sú veľkosť kompenzačného stupňa, poradie v kompenzačnej skupine alebo či je stupeň dostupný pre automatický režim.

Rovnako ako stýkač obsahuje vstupy pre spätné väzby stupňov a chyby zariadenia. Obsahuje aj vstup, ktorý slúži na zablokovanie kompenzačného stupňa. Po uplynutí vopred definovaného času sa stýkač odblokuje. To sa využíva v prípade, ak kompenzačný stupeň prešiel zo stavu zatvorený na otvorený, inými slovami v regulačnom cykle už nebol vyžadovaný, tak je zablokovaný na päť minút. Po uplynutí tejto doby bol znova

dostupný. To je nutné z dôvodu, aby nedochádzalo k rýchlemu spínaniu kompenzačných stupňov.

V rámci diagnostiky bola implementovaná funkcia, ktorá kontroluje nominálnu hodnotu jalového výkonu daného kompenzačného stupňa. V prípade, ak nie je jalový výkon dostatočný, tak je vyvolaná chyba a reálny kompenzačný stupeň obsahuje nejaké poškodenie. To je realizované pomocou absolútnej hodnoty rozdielu meraného jalového výkonu a veľkosti kompenzačného stupňa. Tento rozdiel v podstate určuje jalový výkon pred a po kompenzácii. Ak je tento 80% dodaného jalového výkonu menší ako nominálna hodnota jalového výkonu kompenzačného stupňa, tak došlo k chybe a kompenzačný stupeň sa stáva nedostupným.

Kompenzačný stupeň je tiež možné ovládať z dvoch režimov, ktorých logika je rovnaká ako pri funkčnom bloku stýkač. Taktiež obsahuje rovnaké počítadlo prevádzkových hodín a počtu zopnutí. Počet zopnutí kompenzačného stupňa má úlohu pri jeho výbere v automatickom režime regulátora. O výber sa stará nadradený funkčný blok kompenzačná skupina. Preferujú sa kompenzačné stupne s nižším počtom zopnutia vzhľadom na rovnomerne zaťažené.



Obr. 23: Programová jednotka na ovládanie 300kVar kompenzačného stupňa na termináli A.

3.3.6 Regulátor účinníku

Funkčný blok regulátora z meraných elektrických veličín vyráta, koľko je potrebné dodať alebo odobrať jalového výkonu, tak aby systém dosiahol požadovanú hodnotu účinníku. Z popisu je zrejmé, že medzi vstupmi bude požadovaná hodnota účinníku, charakter účinníku a meranie aktuálnej hodnoty účinníku, čo v našom systéme je na sekundárnom vedení transformátora 22/0,6kV. Veľkosť odchýlky jalového výkonu, aby systém dosiahol požadovanú hodnotu účinníka, sa vypočíta podľa vzťahu 16.

$$Q_{Dev} = Q_{MV} - \frac{|P_{MV}|}{\tan(\pi \cdot 0{,}5) - \cos^{-1}(|\cos(\varphi)_{SP}|)} \tag{16}$$

Výsledná odchýlka je následne skorigovaná, aby bola deliteľná hodnotou najmenšieho kompenzačného stupňa v celom elektrickom systéme bez zvyšku.

V prípade, ak je odchýlka kladná, bude systém v podkompenzovanom stave a je nutné zopnúť taký počet stupňov, kde súčet ich jalových výkonov je rovný odchýlke. V opačnom prípade je systém v nadmerne kompenzovanom stave a je nutné odopnúť určité kompenzačné stupne.

Regulátor dáva príkazy, kedy má byť vykonaný akčný zásah do systému. Príkazy sú posielané cyklicky a prebiehajú v nejakom časovom intervale. Časový interval vychádza zo základných hodnôt nastavení regulátora. Rozlišujeme čas regulácie v pod kompenzovanom stave alebo pre kompenzovanom stave. Inkrementácia časovača je závislá na cyklickej triede programu a pomeru medzi odchýlkou jalového výkonu a najmenším kompenzačným stupňom. Týmto je zaručené, že v prípade väčšej odchýlky jalového výkonu bude príkaz na akčný zásah vyvolaný s vyššou frekvenciou a naopak.

Funkčný blok taktiež obsahuje definovateľné regulačné pásmo. V prípade, ak sa odchýlka jalového výkonu nachádza mimo toto pásmo, akčný zásah je vykonaný.

Všetky informácie o zásahu do systému sú následne predané funkčnému bloku *vyváženie záťaže*, ktorý ich ďalej spracováva.

3.3.7 Vyváženie záťaže

Navrhnutý funkčný blok slúži na optimálne rozdelenie odchýlky medzi tri kompenzačné skupiny na troch termináloch. Funkčnému bloku sú taktiež predané informácie o všetkých troch kompenzačných skupinách. Tieto informácie zahrňujú napríklad počet dostupných stupňov, dostupný jalový výkon kompenzačnej skupiny alebo už dodaný jalový výkon. Kroky funkčného bloku sú nasledovné:

1. Čakanie na príkaz z funkčného bloku regulátor účinníku
2. Kontrola či je nutný akčný zásah do 300 kVar kompenzačného stupňa na termináli A:
    a. Zopnúť: súčet aktuálnej odchýlky a sumy jalového výkonu zopnutých kompenzačných skupín B a C je väčšia ako 300kVar. Potom prejsť na krok 3.
    b. Odpojiť: 300kVar kompenzačný stupeň je už zopnutý a súčet aktuálnej odchýlky a sumy jalových výkonov kompenzačných skupín je menší ako 300kVar. Potom prejsť na krok 3.
    c. Nič z toho: pokračovať na krok 3.
3. Výpočet optimálneho rozdelenia odchýlky pre termináli B a C.
4. Odovzdanie optimálneho rozdelenia kompenzačným skupinám.

Optimálne rozdelenie odchýlky jalového výkonu

Cieľom optimálneho rozdelenia odchýlky je zabezpečiť, aby absolútna hodnota rozdielu účinníku na termináli B a termináli C bola čo najmenšia. Inými slovami je požiadavka na to aby hodnoty účinníkov boli čo najpodobnejšie. Postup je následovný: V prípade podkompenzovaného stavu systému sa zoberú hodnoty jalových výkonov dostupných stupňov z kompenzačných skupín B a C. V prípade nadmerne kompenzovaného stavu sa zoberú hodnoty z aktuálneho dodaného jalového výkonu.

Hodnoty sa prevedú na vektory (pole) s hodnotami od nuly až po maximálnu hodnotu jalového výkonu. Diskrétnym krokom je hodnota jalového výkonu najmenšieho stupňa danej skupiny.

$$Q_{Avb_A} = 300kVar = [0;\ 25;\ 50;\ 75;\ 100;\ 125;\ \ldots; 300]$$
$$Q_{Avb\_B} = 200kVar = [0;\ 12{,}5;\ 25; 37{,}5;\ 50;\ \ldots; 200] \quad (17)$$

Pomocou karteziánskeho súčinu týchto vektorov sú vytvorené všetky možné kombinácie rozdelenia odchýlky jalového výkonu medzi terminály B a C.

$$K = Q_{Avb_A} \times Q_{Avb\_B} = [[0; 0]; [0; 12{,}5]; [0; 25]; [0; 37{,}5]; \ldots; [300; 200]] \quad (18)$$

Následne prebehla filtrácia týchto kombinácií. Filtrovanie spočívalo v selekcii prvkov vektoru K, kde ich súčet hodnôt sa rovnal aktuálnej odchýlke získanej z regulátora.

Pre všetky prípustné kombinácie sa vypočítali nové hodnoty účinníkov, ktoré zahŕňali prírastky prípustných kombinácií rozdelenia jalových výkonov.

$$\cos\varphi_{new} = \frac{P_{MV}}{\sqrt{P_{MV}^2 + (Q_{MV} \pm Q_{comb})^{\square}}} \quad (19)$$

Takisto sa vypočítal absolútny rozdiel hodnôt týchto nových účinníkov a vybral sa prvok s najmenšou hodnotou.

Výber spočíval v zoradení poľa dátových štruktúr na základe absolútneho rozdielu účinníkov. Prvý prvok zoradeného poľa obsahoval hľadané optimálne rozdelenie odchýlky jalového výkonu. V aplikácii bolo navrhnuté pole dátových štruktúr, kde jedna dátová štruktúra obsahovala rozdelenie odchýlky, nové účinníky terminálov a ich absolútnu hodnotu rozdielu.

Navrhnutý algoritmus optimálneho rozdelenia odchýlky jalového výkonu je vo svojej podstate riešený hrubou silou. V tejto aplikácii to nepredstavuje problém, pretože v najhoršom možnom prípade budú spomenuté operácie prebiehať na všetkých možných kombináciách rozdelenia odchýlky jalového výkonu a riešenie bude zaručene nájdené.

3.3.8 Kompenzačná skupina

Ako z názvu vyplýva, funkčný blok slúži na skupinové ovládanie kompenzačných stupňov v automatickom režime. V automatickom režime programu funkčný blok prevezme optimálne rozdelenie odchýlky jalového výkonu a hľadá optimálnu kombináciu kompenzačných stupňov, aby súčet jalových výkonov jednotlivých stupňov bol rovný odchýlke. Optimálny výber je popísaný v samostatnej podkapitole.

Okrem toho funkčný blok poskytuje užitočné informácie pre aplikáciu alebo operátora. Tými informáciami sú napríklad počet dostupných stupňov pre automatický režim, hodnota dostupného jalového výkonu alebo hodnota dodaného jalového výkonu.

Po optimálnom výbere kompenzačných stupňov funkčný blok zostaví spínaciu sekvenciu. Prvé prvky sekvencie sú stupne, ktoré treba odpojiť a potom pokračujú stupne, ktoré treba zopnúť. Vykonanie príkazov sekvencie je dané definovateľnou frekvenciu

spínania. Základnou hodnotou je 5 sekúnd. To znamená, že každých 5 sekúnd bude vyslaný príkaz na odpojenie alebo pripojenie jednotlivého kompenzačného stupňa. Po ukončení sekvencie je poslaná informácia o ukončení funkčnému bloku *vyvážanie záťaže*, ktorý okrem čakania na koniec spínacej sekvencie, čaká aj na príkaz regulátora.

Optimálny výber kompenzačných stupňov

Optimálny výber kompenzačných stupňov spočíva vo výbere takých kompenzačných stupňov, ktorých súčet jednotlivých jalových výkonov je rovný požadovanej odchýlke. V prípade nadmernej kompenzácie systému budú vyberané už pripojené kompenzačné stupne. Súčet ich jalových výkonov musí byť rovný absolútnej hodnote odchýlky jalového výkonu, pretože odchýlka bude v tomto prípade záporná.

Riešenie tohto problému vychádza z riešenia optimalizačného „problému batohu“ (z anglického *Knapsack problem*). Optimalizačný problém batohu môžeme opísať týmito vzťahmi

$$max \sum_{i=0}^{n} x_i \cdot c_i \tag{20}$$

za podmienok

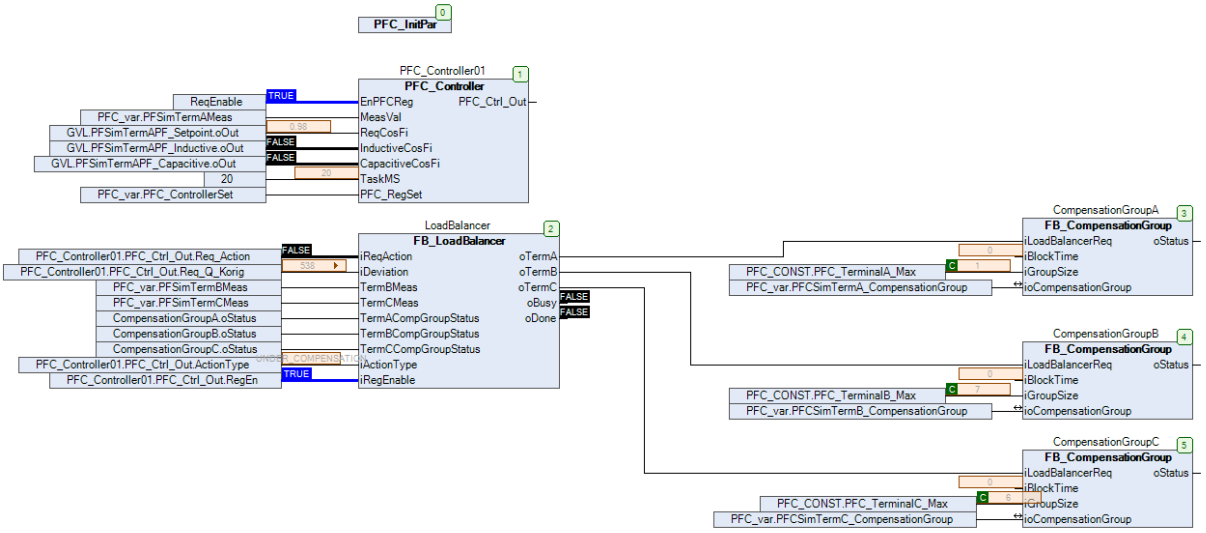
$$\sum_{i=0}^{n} x_i \cdot w_i \leq W \tag{21}$$

$$x_i \in \{0,1\} \tag{22}$$

Úlohou tohto problému je vybrať podmnožinu predmetov x_i, , ktoré majú definovanú váhu w_i a cenu c_i do batohu s obmedzenou kapacitou W, tak aby sme maximalizovali cenu predmetov v batohu. Predmety sa nedajú deliť, to znamená, že predmety budú umiestnené v batohu alebo nebudú. [15]

Formulácia problému batohu pre náš optimalizačný problém je nasledovná: Predmety sú kompenzačné stupne a maximálna kapacita batohu je priradená absolútnej hodnote odchýlky jalového výkonu. Váha predmetov je rovná jalovému výkonu kompenzačných stupňov. Nerovnosť vo vzťahu 21 je upravená na rovnosť, pretože súčet jalového výkonu kompenzačných stupňov musí byť rovný odchýlke, aby bola dosiahnutá požadovaná hodnota účinníku. Ohodnotenie kompenzačných stupňov je nasledovné: Na začiatku je cenám všetkých stupňov priradené nejaké veľké číslo. K tomu číslu je potom pripočítaný výkon daných stupňov a odčítaný počet jeho zopnutí. Vychádza to z toho, že sú preferované stupne s väčším jalovým výkonom a zároveň s nižším počtom zopnutí.

Problém bol riešený implementáciou algoritmu dynamického programovania. [15] Algoritmus funguje na báze vyplnenia tabuľky. Stĺpce tabuľky označujú cenu doposiaľ zvolených kompenzačných stupňov. Riadky značia stav po zvolení í-teho kompenzačného stupňa. Bunky tabuľky označujú hodnotu jalové výkonu kompenzovaného stupňa. Spätným sledovaním rozhodnutí algoritmu dynamického programovania bola získaná optimálna kombinácia kompenzačných stupňov.



Obr. 24: Programová jednotka hlavného regulačného cyklu.

3.3.9 OPC UA server

Ako už bolo spomenuté v kapitole 3.2, komunikácia medzi kooperatívnym simulačným prostredím Mosaik a PLC bude postavená na protokole OPC UA. OPC UA server bude bežať na PLC a k nemu búdu pripojený klienti. Tvorba OPC UA servera na PLC AC500 V3 nevyžaduje žiadne programovanie a spočíva v jednoduchej konfigurácii. Postup konfigurácie môžeme zhrnúť do uvedených krokov a je tiež popísaný v off-line dokumentácii programu Automation Builder:

1. Pridanie objektu *OPC UA server* na jedno z ethernotových rozhraní napr. *ETH1*.
2. Pridanie objektu *Symbol Configuriation* do aplikačnej štruktúry.
3. Povoliť nastavenie *Support OPC UA features* v objekte *Symbol Configuration.*
4. Aktivácia premenných, ktoré budú dostupné klientom. Možnosť nastavenia prístupových práv.
5. Skompilovať objekt *Symbol Configuration*.
6. Stiahnuť projekt na PLC.

V tomto okamihu je konfigurácia hotová. Správnosť konfigurácie sa testovala prostredníctvom voľne dostupného programu UaExpert. Program UaExpert je OPC UA klient s jednoduchým grafickým rozhraním s podporou všetkých funkcionalít protokolu OPC UA. Podrobná dokumentácia ako program používať je dostupná na odkaze [16].

Po pripojení na OPC UA server pomocou URL adresy *opc.tcp://192.168.1.10:4840* bolo vytvoren0 spojenie. Po úspešnom spojení sa vložil niektorý z dostupných OPC UA uzlov servera do okna *Data Access View*, ktoré slúži na manipuláciu s OPC UA uzlami. V tomto okne môžeme čítať alebo upravovať hodnoty uzlov. Výsledok testu je znázornený na obrázku 25 a 26.

Data Access View

#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	OPCUAServer@PM5650-2ETH	NS4|String|var|DP_PFC_AC500.Application.PFSim.TerminalA.ExtCurrentMeas	ExtCurrentMeas	0.342974	Float	1:25:31.667	1:25:31.667	Good
2	OPCUAServer@PM5650-2ETH	NS4|String|var|DP_PFC_AC500.Application.PFSim.TerminalA.ExtPMeas	ExtPMeas	3.3493	Float	1:25:42.149	1:25:42.149	Good
3	OPCUAServer@PM5650-2ETH	NS4|String|var|DP_PFC_AC500.Application.PFSim.TerminalA.ExtQMeas	ExtQMeas	1.22223	Float	1:25:23.945	1:25:23.945	Good
4	OPCUAServer@PM5650-2ETH	NS4|String|var|DP_PFC_AC500.Application.PFSim.TerminalA.ExtVoltageMeas	ExtVoltageMeas	6.00177	Float	1:25:42.361	1:25:42.361	Good
5	OPCUAServer@PM5650-2ETH	NS4|String|var|DP_PFC_AC500.Application.PFSim.TerminalA.ExtSwitch.isClosed	isClosed	true	Boolean	1:19:58.266	1:19:58.266	Good
6	OPCUAServer@PM5650-2ETH	NS4|String|var|DP_PFC_AC500.Application.PFSim.TerminalA.ExtSwitch.onOff	onOff	true	Boolean	1:20:01.115	1:20:01.115	Good

Obr. 25: Testovací OPC UA klient.

DP_PFC_AC500.Application.PFSim

Expression	Type	Value	Prepared value	Address	Comment
TerminalA	SimExtTermMeas				
ExtPMeas	REAL	3.34930134			Active Power measurement in MW
ExtQMeas	REAL	1.22223318			Reacte Power measurement in Mvar
ExtCurrentMeas	REAL	0.342974246			Current measurement in kA
ExtVoltageMeas	REAL	6.001768			Voltage measurement in kV
ExtSwitch	SimExtBreaker				Switch command and feedback
isClosed	BOOL	TRUE			
onOff	BOOL	TRUE			
TerminalB	SimExtTermMeas				
TerminalC	SimExtTermMeas				
TerminalA_Compensation	ARRAY [1..PFC_CONST.PFC_TerminalA_...				
TerminalB_Compensation	ARRAY [1..PFC_CONST.PFC_TerminalB_...				
TerminalC_Compensation	ARRAY [1..PFC_CONST.PFC_TerminalC_...				
MosaikStatus	SimStatus				
SimTime	INT	457			
Connected	BOOL	TRUE			

Obr. 26: Príklad dostupných uzlov na OPC UA servery.

3.4 Návrh HMI

Skratka HMI (Human Machine Interface) v preklade do slovenčiny znamená rozhranie medzi človekom a strojom. Ide o grafické obrazovky alebo objekty, ktoré sú zobrazované formou mobilných aplikácií, webstránok alebo sú spúšťané na operátorských paneloch. HMI slúžia na interakciu medzi operátorom a strojom. Prostredníctvom HMI operátor dokáže ovládať jednotlivé časti stroja, zobrazovať diagnostické dáta stroja, rôzne procesné merania, zobrazovanie alarmov a oveľa viac.

V produktovej rodine ABB AC500 V3 sa HMI tvoria softwarom nástroji PB610 Panel Builder 600, ktorý je integrovaný do inžinierskeho prostredia Automation Builder. Program je určený na efektívnu tvorbu flexibilných HMI aplikácií v rôznych priemyselných odvetviach. Obsahuje množstvo funkcionalít. Hlavné funkcie, ktoré sme využili pri tvorbe HMI sú tieto:

- Obsahuje bohaté knižnice widgetov (grafických elementov).
- Jednoduchá úprava a tvorba vlastných widgetov.
- Šablóny stránok pre profesionálny dizajn.
- Návrh, úprava funkcií a dynamická správa widgetov prostredníctvom jazyka Javascript.
- Jednoduché ukladanie dát a prezentácia alarmov a trendov.
- Simulácia HMI pre efektívne uvedenie do prevádzky. [14]

V rámci vývoja HMI aplikácie bolo vytvorených sedem stránok a päť grafických objektov, ktoré slúžia na ovládanie celého riadiaceho programu a jednotlivých typizovaných prvkov systému. Vývoj HMI zohľadňoval súčasné grafické trendy, objektovo orientovaný vývoj a dával veľký dôraz na opakované využitie objektov.

Aby HMI bolo schopné komunikácie s aplikáciou bolo nutné vytvoriť tento prenos dát. To prebieha v dvoch krokoch. Prvým krokom je definovanie komunikačného protokolu v okne *Protocols*. Tu bol použitý štandardný protokol TCP/IP a na fyzickej vrstve bolo použité ethernetové médium. Druhým krokom bolo importovanie dátových premenných, ktoré sa nazývajú *tagy*. Súbor s tagmi je exportovaný z dát objektu *Symbol Configuration* pri kompilácii aplikácie. Tagy sa importujú v karte *Tags* pod daným protokolom. Aby sa nezaťažovala komunikácia, každý vytvorený objekt obsahoval dátovú pod štruktúru s názvom HMI, ktorá obsahovala informácie a príkazy daného objektu. V rámci importu boli vybrané iba tieto štruktúry.

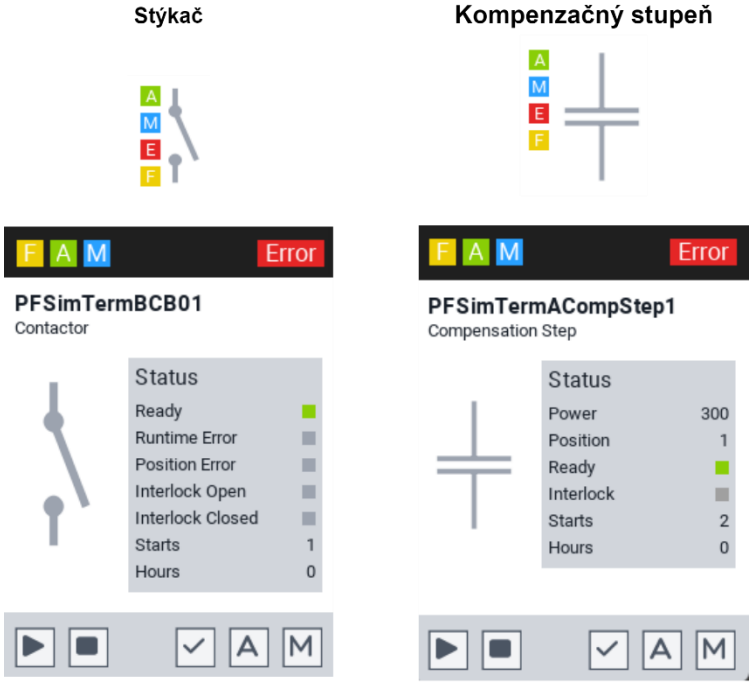
Následne bola vytvorená šablóna, ktorá obsahovala navigáciu, ktorá slúžila na prechádzanie medzi jednotlivými stránkami. Vytvorená šablóna bola použitá ako predloha pre každú vytvorenú stránku. To zjednodušilo vývoj, pretože úprava komponentov sa prejaví globálne v celej vizualizácii.

3.4.1 Navrhnuté grafické objekty

Pre každý funkčný blok z navrhnutej knižnice, ktorý popisuje nejaký komponent systému, bola vytvorená jeho grafická reprezentácia. Každému bol navrhnutý vlastný zložený widget, ktorého prvky sa dynamicky menili v závislosti na stave prvku. Po kliknutí na widget sa otvorí dialógové okno, ktoré obsahuje elementy, slúžiace na zobrazovanie dát, a ktoré ponúkajú úplnú kontrolu nad komponentami systému. Tento spôsob vývoja zjednodušil neskorší vývoj HMI, pretože umožňoval opakované používanie týchto typizovaných objektov.

Obr. 27: Navrhnuté grafické objekty 1.

Do jednotlivých objektov sa načítali dáta na základe jedinečného identifikátora, ktorý sa nachádza v ceste *tagy*. Všetky dcérske tagy boli následne napojené na atribúty primitívnych widgetov, aby dynamicky menili vzhľad objektov podľa potrieb. Na obrázkoch 27 a 28 sú zobrazené navrhnuté grafické objekty a im patriace dialógové okna.



Obr. 28: Navrhnuté grafické objekty 2.

3.4.2 Domovská stránka

Na domovskej stránke vizualizácie sa nachádza interaktívna jednopólová schéma regulovaného elektrického systému. Operátor systému môže na tejto stránke sledovať procesné merania v jednotlivých bodoch, nastavovať alarmy jednotlivých meraní alebo ovládať dodávku elektrickej energie do jednotlivých terminálov pomocou odpojovacích prvkov. Pre názornosť sa terminály prefarbia zo sivej farby na modrú v prípade prítomnosti napätia. Rovnako aj ostatné prvky, keď sú pod napätím.

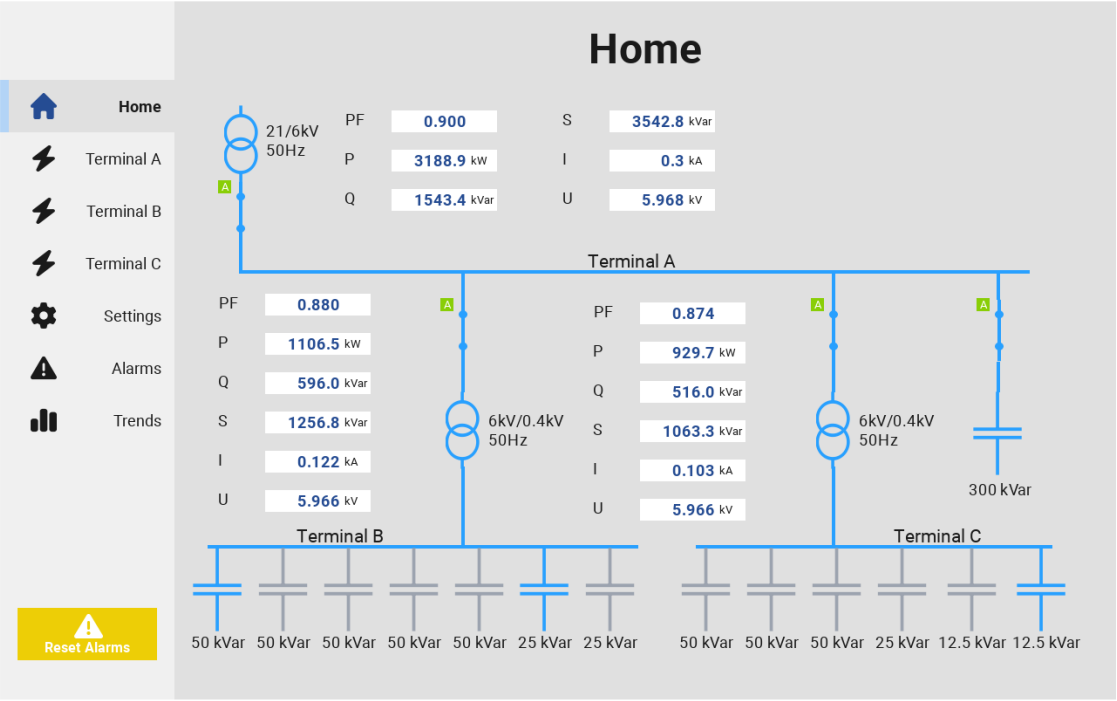
Na tejto stránke sa tiež nachádza stručný prehľad o stavoch jednotlivých kompenzačných stupňov. Ovládanie však prebieha na stránke daného terminálu.

3.4.3 Stránky terminálu

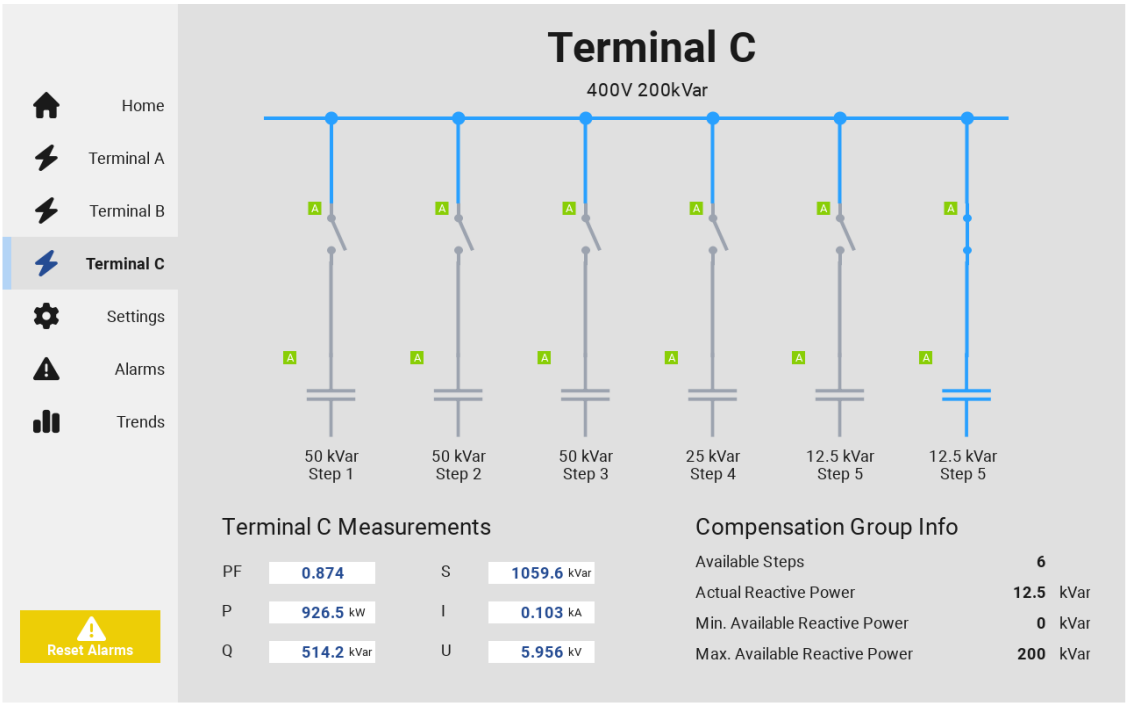
Pre každú kompenzačnú skupinu bola vytvorená samostatná stránka vizualizácie. Na stránke sa opäť nachádza interaktívne jednopólová schéma, kompenzačné stupne, stýkače a súhrn informácií o kompenzačnej skupine. Všetky stránky vyzerajú približne rovnako, líšia sa iba počtom kompenzačných stupňov.

Operátor na týchto stránkach môže po prepnutí do manuálneho režimu ľubovoľne ovládať jednotlivé kompenzačné stupne. Stýkače kompenzačných stupňov sú v programe pevne nastavené na automatický režim, pretože sú riadené príkazmi z

nadradeného funkčného bloku kompenzačný stupeň. Na obrázku 30 je znázornená stránka s kompenzačnou skupinou pripojenou na terminál C.



Obr. 29: Domovská stránka vizualizácie.

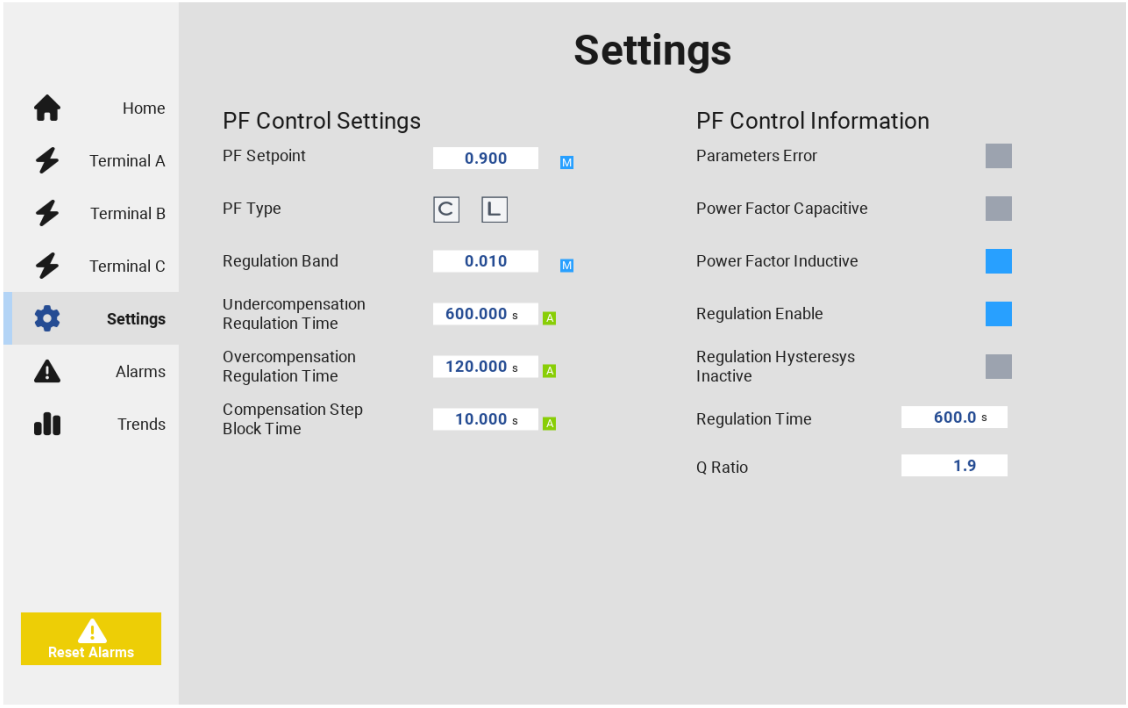


Obr. 30: Stránka s kompenzačnou skupinou.

3.4.4 Stránka s nastaveniami regulátora

Stránka slúži na nastavovanie parametrov regulátora a sledovanie priebehu alebo stavu. V prípade, ak sú vstupné parametre v manuálom režime, operátor môže nastaviť požadovanú hodnotu účinníku, jeho charakter alebo regulačné pásmo. Prebieha tu i nastavovanie časových intervalov akčného zásahu do systému v podkompenzovanom a prekompenzovanom stave.

Na druhej strane obrazovky môže operátor sledovať informácie o stave regulátora. Pod digitálnymi vstupmi regulátora rozumieme, či vstupné parametre majú akceptovateľnú hodnotu alebo či regulácia je povolená. Na obrázku je znázornený odpočet do najbližšieho akčného zásahu, aj pomer medzi odchýlkou jalového výkonu, a výkonom najmenšieho kompenzačného stupňa. Všetky nastavenia a sledované informácie sú zobrazené na obrázku 31.



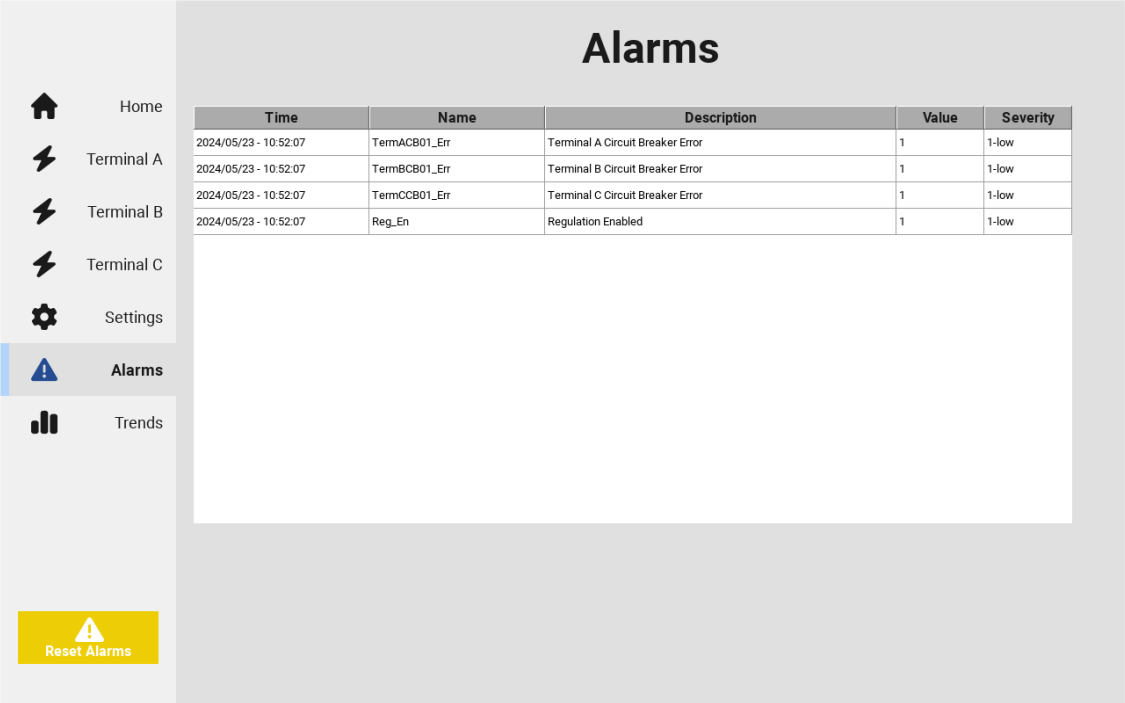
Obr. 31: Stránka s nastavenia regulátora.

3.4.5 Alarmy

Na stránke alarmov sa nachádza zoznam aktívnych alarmov nášho riadiaceho systému. Vytvorenie zoznamu aktívnych alarmov prebieha v dvoch krokoch. Prvým krokom je definovanie všetkých alarmov riadiaceho systému. To sa robí v karte *Alarms*. Konfigurácia jedného alarmu obsahuje jeho názov, alarmovú skupinu, povolenie alarmu, nutnosť potvrdenia alarmu, spúšťač alarmu, zdroj spúšťača a popis alarmu. Nie všetky tieto parametre sú povinné. Navrhnutý zoznam alarmov neobsahuje žiadnu alarmovú skupinu, všetky sú povolené a nie je nutné alarmy potvrdzovať. To znamená, že ak zdroj spúšťača zmení hodnotu z pravdivostnej hodnoty (1=True) na nepravdivú hodnotu (0=False), tak alarm prestane byť aktívny a na stránke alarmov sa už nezobrazuje. Každý

alarm je napojený na nejakú premennú v riadiacom systéme a reaguje na zmenu jej hodnoty. Resetovanie alarmov prebieha pomocou tlačidla „Reset Alarms“ pod navigáciou vizualizácie. Tlačidlo je napojené na globálnu premennú, ktorá slúži na resetovanie chýb každého funkčného bloku riadiaceho systému.

Druhým krokom je vytvorený zoznam alarmov predať widgetu *Active alarms - Lite*. Widget umožňuje zobrazovanie a manipuláciu s aktívnymi alarmami. Na zobrazenie histórie alarmov slúži iný widget (*Alarms History*). Po úprave widget tak, aby zobrazoval len nami potrebné informácie, bol celý proces tvorby alarmov u konca. Informácie, ktoré sú zobrazené predstavujú časovú značku alarmu, názov alarmu, popis alarmu, hodnota zdrojovej premennej a závažnosť alarmu.



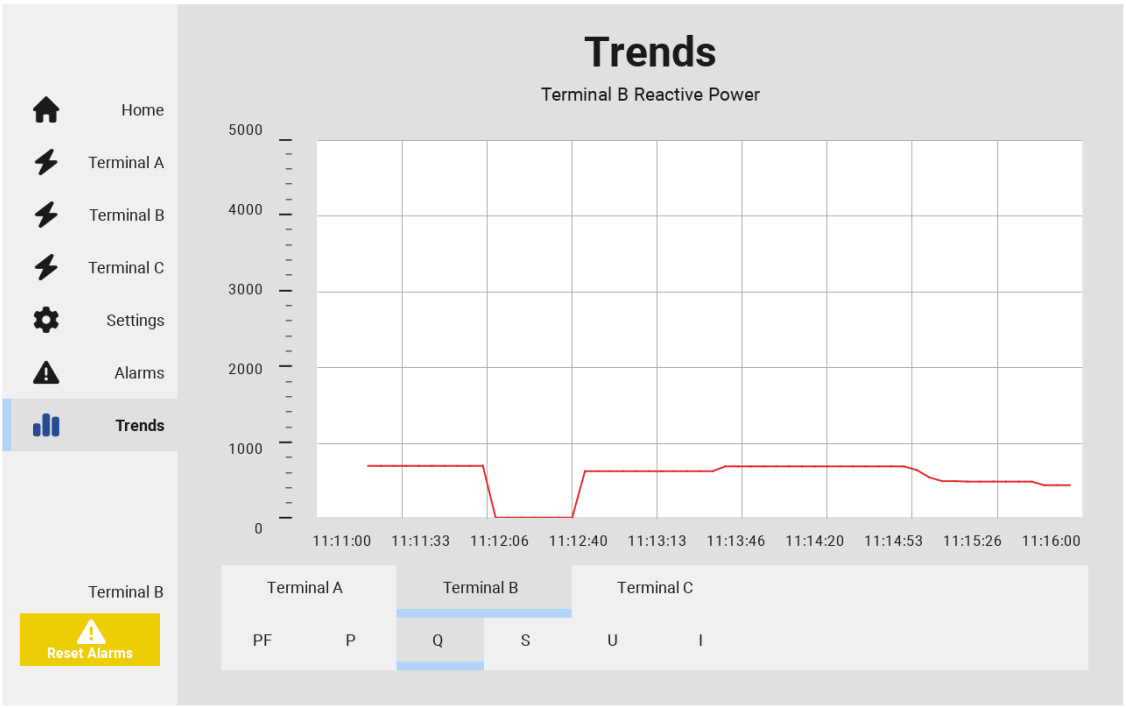
Obr. 32: Stránka aktívnych alarmov.

3.4.6 Trendy

Stránka s trendami slúži na zobrazovanie meraných fyzikálnych veličín v čase. Každému meraniu je vytvorená samostatná stránka s trendom daného merania. Operátor takýmto spôsobom môže sledovať vývoj meranej veličiny v čase a pozorovať zmeny v riadenom systéme.

Vytvorenie trendov prebieha v dvoch úkonoch. Prvým úkonom je vytvorenie dátového buffera s hodnotami merania. Tvorba trendov prebieha v karte *Trends*. Po vytvorení trendu je nutné definovať tieto povinné atribúty, názov trendu, zdrojový tag merania, vzorkovací čas a miesto uloženia. Každý trend vzorkuje meranie každých päť sekúnd a ukladá ich na lokálne úložisko PLC.

Takto vytvorený trend je potom priradený ku widgetu *History Trend*. Tento widget slúži na zobrazovanie trendov a aj ich histórie. Úprava tohto widgetu spočívala v definovaní rozmedzí na osi Y a osi X, a úprave farieb jednotlivých súčastí widgetu. Na obrázku 33 je znázornená obrazovka s trendom merania jalovej energie na termináli B.



Obr. 33: Stránka s trendom merania jalovej energie na termináli B.

4 SIMULOVANÝ REÁLNY SCENÁR A VÝSLEDKY REGULÁCIE

Na overenie správnosti navrhnutého regulátora je nutné ho otestovať. Testovanie prebiehalo pomocou virtuálneho uvedenia do prevádzky pomocou navrhnutého kooperatívneho simulačného prostredia a navrhnutého regulátora. Boli vytvorené dva simulačné scenáre, na ktorých prebehol test systému a analýza meraných elektrických veličín simulovanej prípadovej štúdie. Simulovaná štúdia a požiadavky na jej reguláciu sú vysvetlené v kapitole 3.1.

Prvý simulačný scenár spočíva v simulácii systému, kde neprebehne žiadna zmena systému, ale bude sa meniť požadovaná hodnota účinníku. Cieľom testu je overenie, či regulátor dokáže regulovať na požadovanú hodnotu pri jej zmene. V druhom scenári sa na začiatku nastaví požadovaná hodnota účinníku na nejakú hodnotu a v simulovanom elektrickom systéme sa búdu pripájať ďalšie spotrebiče s rôznym charakterom. Opäť bude cieľom overiť, či regulátor funguje správne, v tomto prípade sa bude meniť stav (poruchová veličina) simulovaného elektrického systému.

Na začiatku oboch testov prebehla príprava simulácie PowerFactory v jeho grafickom prostredí kvôli jednoduchosti ovládania. Hodnoty atribútov entít sú v každom kroku kooperatívnej simulácie zaznamenané do CSV súboru. Dáta z CSV súboru sú vizualizované vo forme grafu pomocou Python knižnice Matplotlib. Obom testom je venovaná samostatná podkapitola, kde je opísaná príprava simulačného prostredia, samotný test a výsledky regulácie.

4.1 Virtuálne uvedenie do prevádzky navrhnutého simulačného prostredia

Na overenie správneho fungovania navrhnutého regulátora je nutné ho otestovať na reálnych situáciách. Testovanie regulátora je vykonané na dodanej simulovanej prípadovej štúdii, ktorá slúži ako softwarová náhrada reálneho hardwaru. Ako už bolo spomenuté, takéto testovanie softwaru sa nazýva virtuálne uvedenie do prevádzky. Postup virtuálneho uvedenia do prevádzky je nasledovný.

Na stiahnutie aplikácie do PLC je nevyhnutné sa k nemu pripojiť pomocou PC. Pripojenie prebieha v prostredí programu Automation Builder. Najskôr je ale potrebné nastaviť vhodné lokálne IP adresy na PLC a na sieťovom rozhraní počítača. IP adresy môžu byť napríklad 192.168.1.10/24 a 192.168.1.20/24. Overenie pripojenia môžeme overiť príkazom ping alebo skenovaním pripojených zariadení funkcie *IP-configuration* v karte *Tools* v programe Automation Builder. Ak je pripojenie úspešné, je možné navrhnutú aplikáciu skompilovať a stiahnuť na PLC. V tomto okamihu môžeme taktiež sledovať zmeny programu alebo premenných v takzvanom online pripojení. V prípade, ak sa PLC nachádza v režime *STOP* je dôležite, ho prepnúť do režimu *RUN*, pomocou príkazu Start v nástrojovej lište programu Automation Builder alebo pomocou tlačidla *RUN* pod displejom PLC. V prípade úspešného stiahnutia aplikácie a pripojenia na PLC

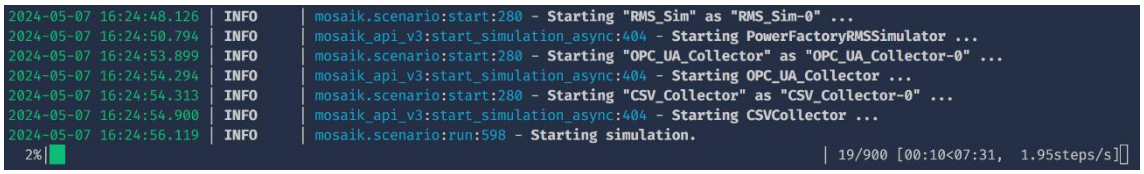
sa na displeji PLC objaví nápis *RUN*, *ako aj* na spodnej lište vývojového prostredia v zelenom rámčeku.

Overenie pripojenia ku OPC UA serveru môže prebiehať pomocou programu UaExpert, ako je vysvetlené v kapitole 3.3.9. Typická URL adresa OPC UA servera je potom *opc.tcp://192.168.1.10:4840*. V tomto momente je riadiaci systém v plnej prevádzke a ostáva spustiť už len navrhnuté kooperatívne simulačné prostredie.

Pred spustením simulačného prostredia sme definovali určité nastavenia. Týmito nastaveniami sú:

- Dĺžka simulácie je 900 simulačných krokov (čo je približne 15 minút a dostatočný čas pre test aplikácie).
- Všetky simulátory majú veľkosť simulačného kroku 1 (v prípade, ak by sme chceli beh v reálnom čase, musíme metóde *run()*, objektu *World* predať parameter *rt_factor=1*. Tým docielime, že jeden simulačný krok bude rovný jednej reálnej sekunde).
- PowerFactory RMS simulátoru sa určilo, aký projek,t a akú prípadovú štúdiu má spustiť.
- Simulátoru OPC UA kolektor sa určila URL adresa OPC UA servera (*opc.tcp://192.168.1.10 :4840*).
- Simulátoru, ktorý zapisoval simulačné dáta, sa určil názov súboru, do ktorého má prebiehať zápis simulačných dát.

V tomto bode bolo všetko priprávané na spustenie kooperatívneho simulačného prostredia pomocou konzolového príkazu *python main.py 127.0.0.1:8888*. Súbor *main.py* obsahuje navrhnutý simulačný scenár a IP adresa, je lokálna adresa TCP/IP socketu, ku ktorému sa majú simulátory pripojiť. Po úspešnom spustení sa zobrazil očakávaný výsledok na štandardom konzolovom výstupe.

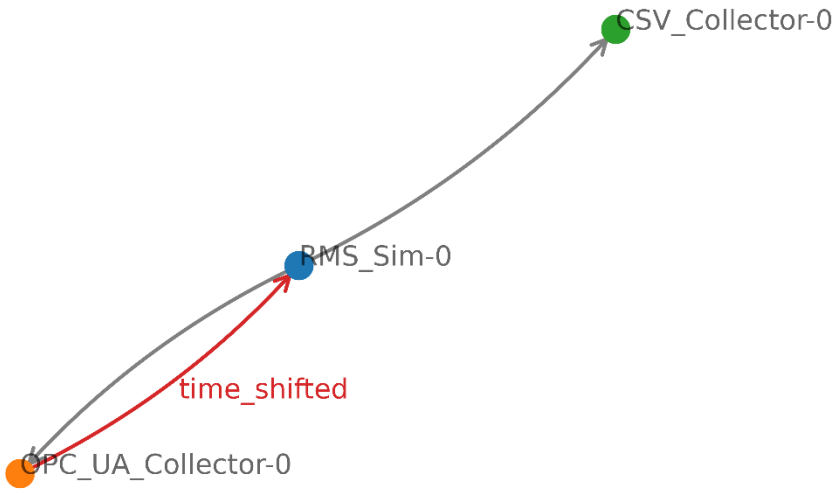


Obr. 34: Konzolový výstup kooperatívnej simulácie.

V tomto okamihu kooperatívne simulačné prostredie funguje a celý systém je vo virtuálnej prevádzke a je možné ho plne otestovať.

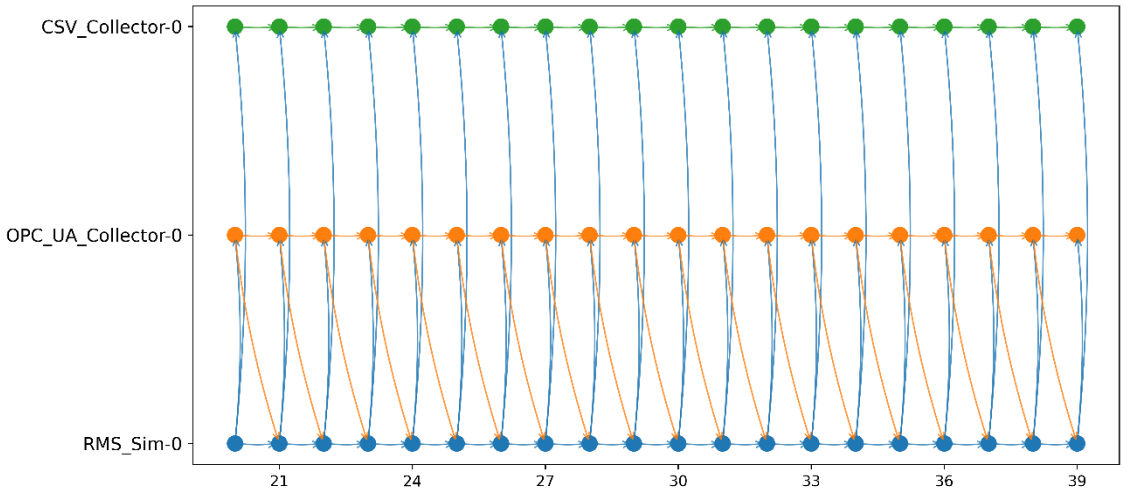
Ekosystém Mosaik obsahuje pomocné funkcie, ktoré slúžia na vykreslenie grafov, ktoré nám môžu priblížiť, akým spôsobom je kooperatívna simulácia vykonávaná, a ako prebieha výmena dát medzi jednotlivými simulátormi. Tok dát medzi simulátormi je znázornený na obrázku 35. Správu komunikácie ma na starosti samotný Mosaik. Graf toku dát zobrazuje smer toku dát medzi simulátormi. V našom prípade simulátor RMS_Sim-0 posiela dáta do simulátoru CSV_Collector-0 a simulátoru OPC_UA_Collector-0. Simulátor OPC_UA_Collector-0 následne posiela dáta naspäť

simulátoru RMS_Sim-0. Týmto je vytvorená cyklická výmena dát medzi simulátormi RMS_Sim-0 a OPC_UA_Collector-0.



Obr. 35: Graf výmeny dat medzi simulátormi.

Druhým grafom, ktorým sa môže overiť požadované chovanie simulácie, je graf vykonávania simulácie. Na grafe je znázornená výmena dát medzi simulátormi v každom kroku simulácie. V dolnej polovici grafu je znázornená cyklická výmena dát medzi simulátormi *RMS_Sim-0* a *OPC_UA_Collector-0*. Je očividné, že tieto dva simulátory sú voči sebe časovo posunuté. To znamená, že dáta zo simulátoru *OPC_UA_Collector-0*, z daného simulačného kroku, sú pre simulátor *RMS_Sim-0* dostupné až v nasledujúcom simulačnom kroku. Takéto správanie zabezpečuje, že simulácia je vykonávaná sekvenčne a nedôjde ku desynchronizácii jednotlivých simulátorov.



Obr. 36: Graf vykonávania simulačných krokov.

4.2 Test regulátora na zmenu požadovanej hodnoty účinníku

Test sa zameriava na testovanie regulátora a jeho schopnosť reagovať na zmeny požadovanej hodnoty účinníku. Stav systému sa v rámci testu nemenil. Regulátor by mal byť schopný rýchlo a presne prispôsobiť výstupný signál tak, aby čo najviac odpovedal novej požiadavke. V teste regulátora je hlavne sledovaná schopnosť dosiahnuť novú požadovanú hodnotu účinníku, rozdelenie odchýlky jalového výkonu medzi dva podružné terminály s dôrazom na podobnosť hodnôt ich účinníkov a optimálny výber kompenzačných stupňov. Optimálne rozdelenie odchýlky a optimálny výber kompenzačných stupňov bolo analyzované v prostredí programu Automation Builder po pripojení na PLC. Vstupné parametre regulátora sú:

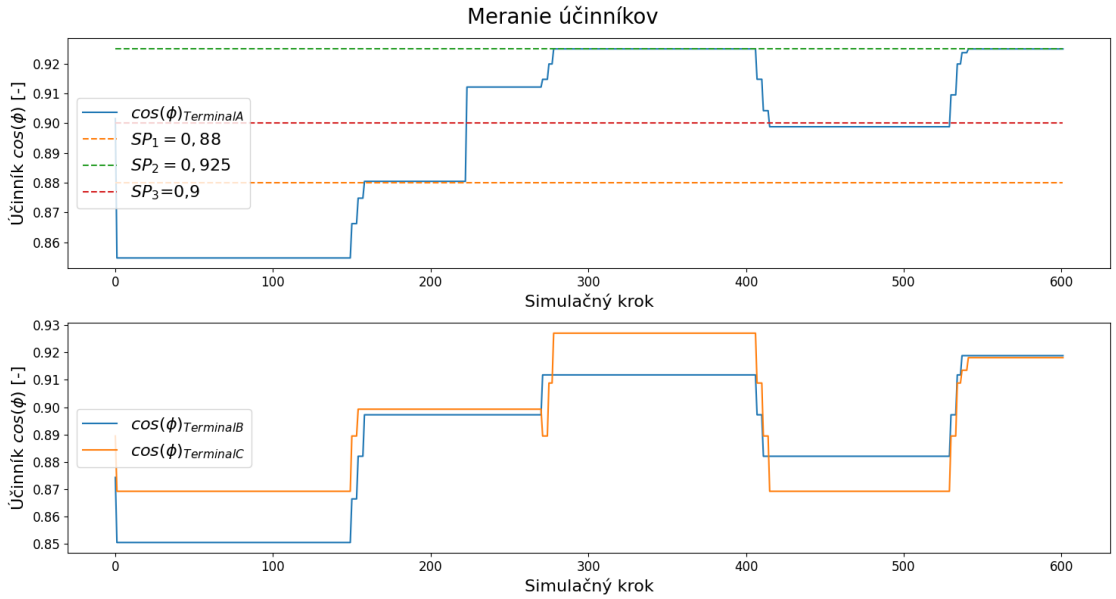
- **Požadovaná hodnota účinníku**:0,88; 0,925; 0,9; 0,925
- **Hysterézia regulačného pásma**: 0,1.
- **Doba regulácie v pod kompenzovanom stave**: 600s.
- **Doba regulácie v pre kompenzovanom stave**: 120s.
- **Charakter účinníku**: Indukčný.

Postup testu bol následovný: Najprv sme uviedli simulovaný systém do virtuálnej prevádzky. Pokračovalo sa tým, že sa odpojil každý kompenzačný stupeň a nastavila sa prvá požadovaná hodnota. Po tomto kroku ostávalo úž len regulátor zapnúť. Počas priebehu testu boli simulačné dáta zapisované v každom kroku simulácie do CSV súboru a následne spracované pomocou knižnice pandas a matplotlib. Výsledok testu pozostáva zo šiestich grafov, na ktorých sa nachádzajú merané hodnoty účinníkov, merania všetkých typov výkonov, merania elektrického prúdu a elektrického napätia a stavy všetkých kompenzačných stupňov.

Na začiatku simulácie mal účinník na termináli A hodnotu 0,85. Požadované hodnoty účinníku v teste sa menili týmto spôsobom. Približne v stom simulačnom kroku sa zmenila požadovaná hodnota účinníku na hodnotu 0,88, potom v približne dve stom kroku na hodnotu 0,925, v približne štyri stom kroku na hodnotu 0,9 a nakoniec v približne päťstom kroku naspäť na hodnotu 0,925.

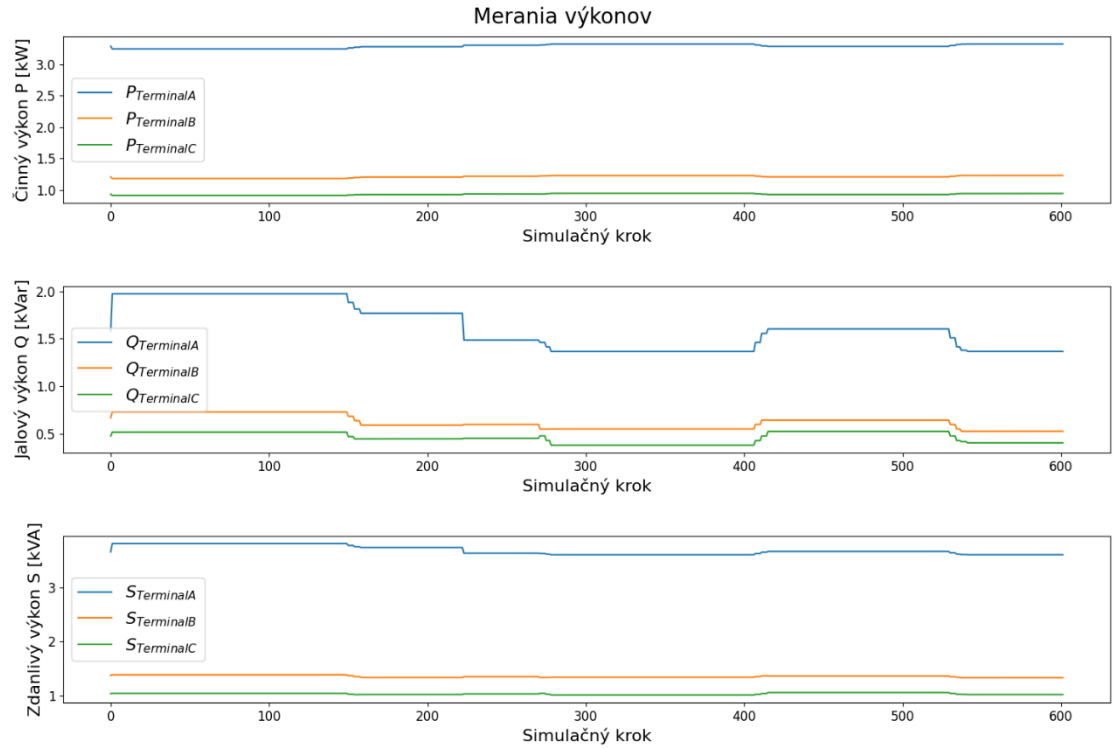
Akčný zásah do systému prišiel až o pár desiatok simulačných krokov neskôr, kvôli navrhnutému správaniu regulátora. Je to spôsobené odpočtom doby regulácie v podkompenzovanom stave. V prípade ak odpočet skončil, prebiehal akčný zásah. Rýchlosť odpočtu je závislá na pomere odchýlky jalovej energie a najmenšieho kompenzačného stupňa. Zuby na krivkách účinníku na obrázku 38 sú spôsobené postupným spínaním kompenzačných stupňov. Na obrázku 38 je taktiež vidno, že regulátor pri každej zmene požadovanej hodnoty účinníku dosiahol požadovanú hodnotu. Pri zmene požadovanej hodnoty z 0,925 na 0,9 je viditeľná malá odchýlka, ale tá je spôsobená hysteréziou regulačného pásma.

Na druhom grafe na obrázku 37 je znázornený priebeh účinníkov na termináli B a C. Je viditeľné, že najväčší rozdiel hodnôt je približne 0,02. Po bližšej analýze hodnôt programu sa došlo k záveru, že neexistoval menší rozdiel, a tým bola odchýlka optimálne rozdelená.



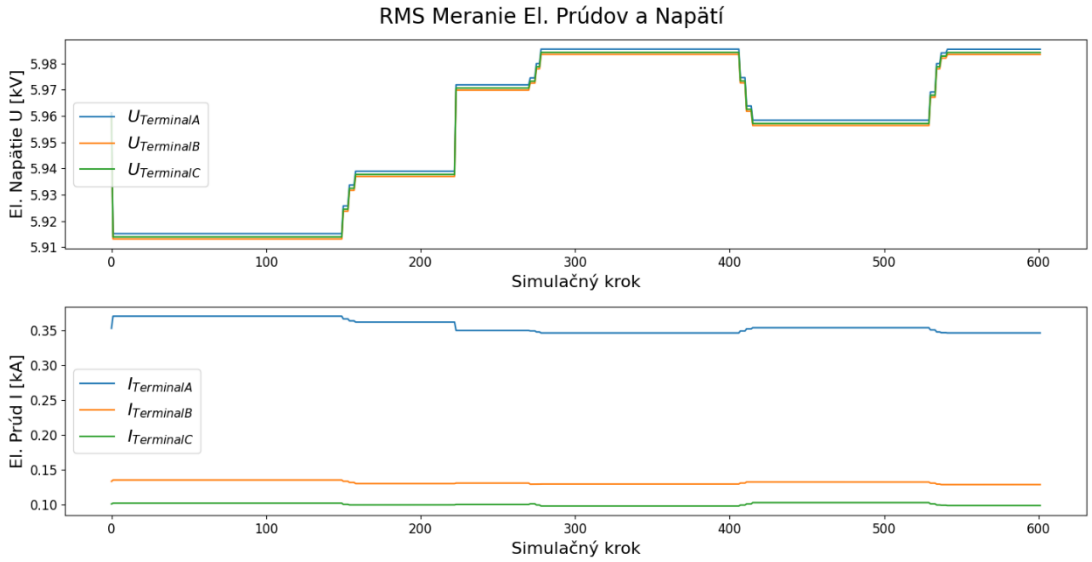
Obr. 37: Meranie účinníkov na každom terminály.

Pri úspešnom prevedení kompenzácie účinníku na termináli A môžeme pozorovať zmenu jednotlivých výkonov na obrázku 38. Rovnako ako pri krivkách účinníkov aj tu sú zuby na krivkách spôsobené postupným spínaním kompenzačných stupňov. V momente, ak je zopnutý nejaký kompenzačný stupeň dôjde k poklesu jalovej energie o približnú hodnotu jeho jalového výkonu. Zároveň dôjde aj k poklesu prenášaného zdanlivého výkonu. Vzťahy medzi jednotlivými výkonmi sú vysvetlené v kapitole 2.1.



Obr. 38: Meranie výkonov na každom terminály.

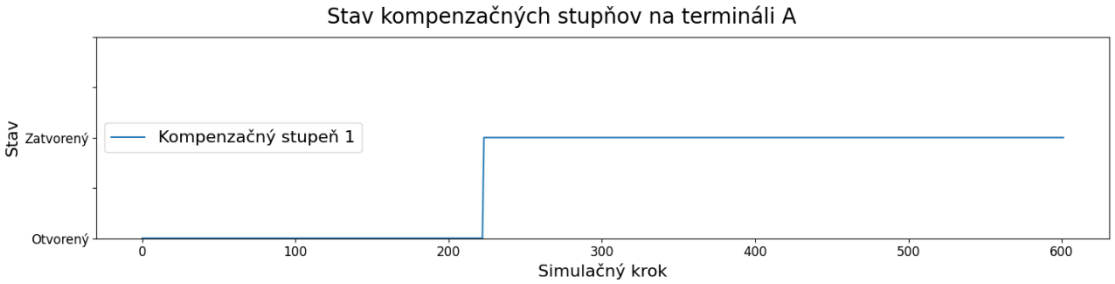
Ďalším pozitívnym efektom kompenzácie účinníka je menšia hodnota úbytku napätia na elektrickom vedení. Zmenu úbytku napätia môžeme pozorovať na priloženom obrázku 39. Pri vyššej hodnote účinníka sa znížil úbytok napätia, čo znamená vyššiu hodnotu napätia na zdroji (terminál A). Aj hodnota odoberaného prúdu sa znížila, ale činný výkon ostal takmer nezmenený



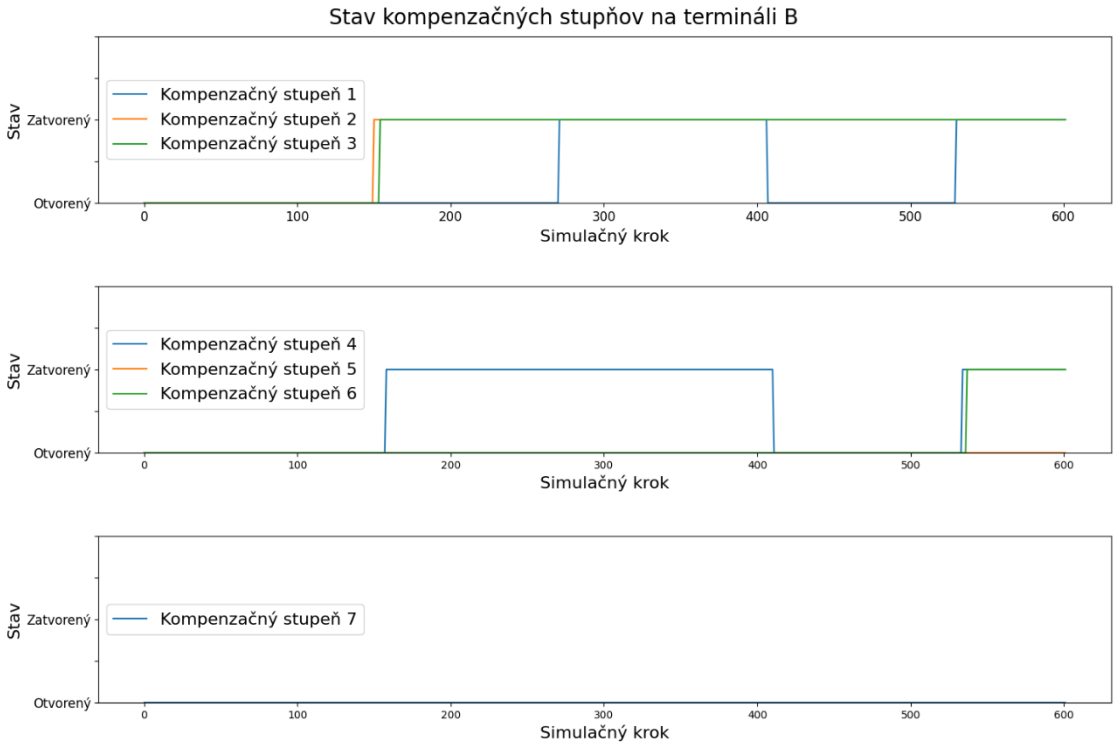
Obr. 39: RMS hodnoty elektrického napätia a prúdu na všetkých termináloch.

Optimálne rozdelenie odchýlky jalového výkonu a optimálny výber kompenzačných stupňov bolo úlohou funkčného bloku vyváženia záťaže a kompenzačná skupina. Monitorovanie týchto informácií prebiehalo v prostredí programu Automation Builder.

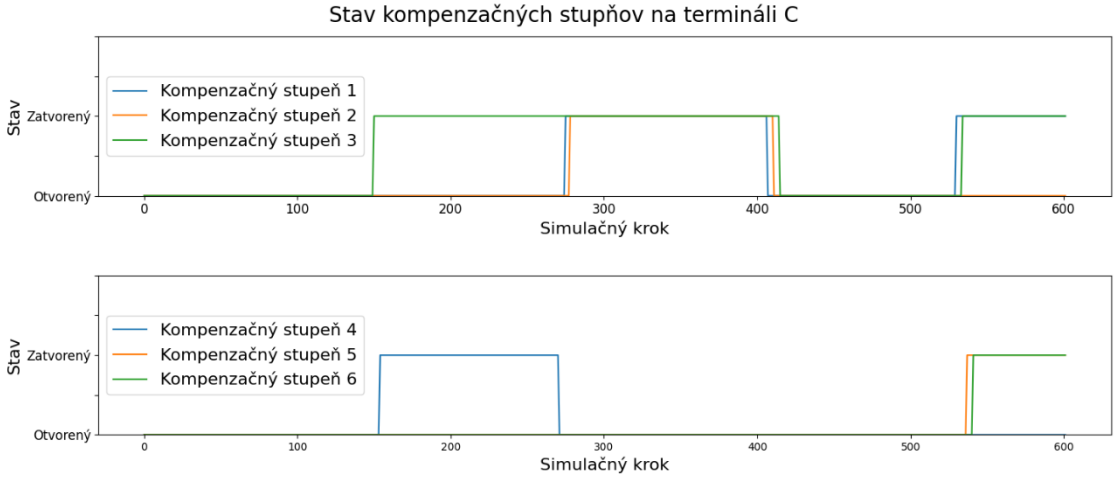
Na obrázku 40 je znázornený stav 300kVar kompenzačného stupňa. Hlavnou podmienkou zopnutia tohto stupňa je, aby súčet odchýlky a dodaného jalového výkonu bol väčší ako 300kVar. To sa udialo pri zmene požadovanej hodnoty na 0,925 približne v kroku 200. Ako môžeme vidieť, zopnutie prebehlo podľa požiadavky. Zvyšná odchýlka jalového výkonu bola v nasledujúcom cykle akčného zásahu rozdelená medzi terminály B a C. Optimálny výber kompenzačných stupňov z kompenzačných skupín B a C je znázornený na obrázkoch 40 až 42.



Obr. 40: Zmena stavu kompenzačného stupňa na termináli A.



Obr. 41: Zmena stavu kompenzačných stupňov na termináli B.



Obr. 42: Zmena stavu kompenzačných stupňov na termináli C.

Zo získaných výsledkov testu na zmenu požadovanej hodnoty účinníku môžeme povedať, že regulátor reguluje s dostatočnou presnosťou a jeho správanie odpovedaná požadovanému správaniu.

4.3 Test regulátora na zmenu stavu systému

V tomto teste sa pristupovalo k zmenám systémových podmienok na overenie regulátoru presne naopak. Na začiatku testu sa nastavila jedna požadovaná hodnota účinníku a tá sa nemenila počas celého testu. Čo sa však menilo, bol stav simulovaného elektrického systému.

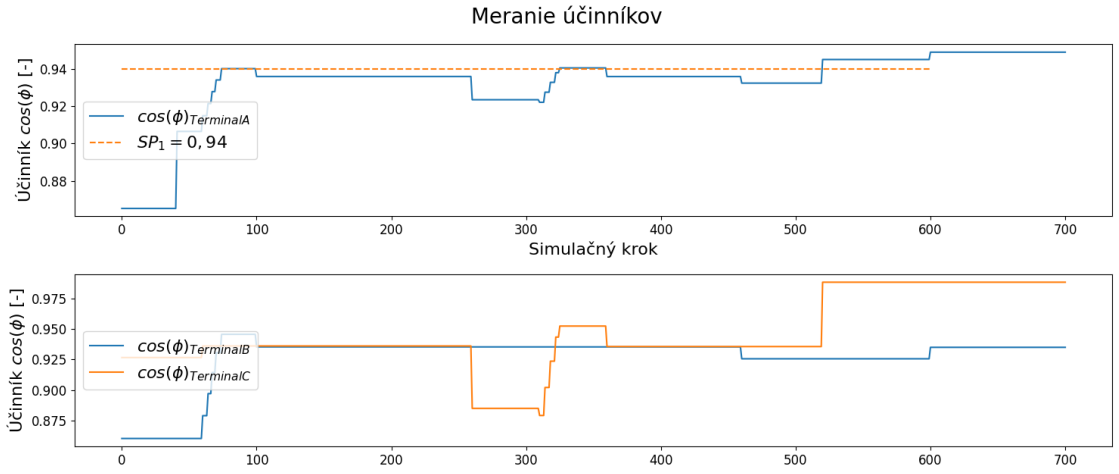
V grafickom rozhraní programu PowerFactory boli vytvorené udalosti, ktorých úlohou bolo pripojiť ďalšie spotrebiče s rôznym charakterom účinníku do simulovanej elektrickej siete. Rovnako ako v predchádzajúcom teste aj tu sa hlavne sleduje schopnosť regulátora udržať požadovanú hodnotu účinníku, rozdelenie odchýlky jalovej energie medzi dva terminály a optimálny výber kompenzačných stupňov. Vstupné parametre regulátora sú:

- **Požadovaná hodnota účinníku**:0,94
- **Hysterézia regulačného pásma**: 0,1.
- **Doba regulácie v pod kompenzovanom stave**: 600s.
- **Doba regulácie v pre kompenzovanom stave**: 120s.
- **Charakter účinníku**: Indukčný.

Začiatok testu bol totožný s predchádzajúcim testom. Po virtuálnom uvedení do prevádzky mal účinník v mieste merania na termináli A hodnotu približne 0,86. Po povolení regulácie bol elektrický systém regulovaný na požadovanú hodnotu 0,94. Požadovaná hodnota účinníku a zmena hodnoty účinníku je znázornená na obrázku 43.

V priebehu testu boli do simulovanej elektrickej siete pripájané alebo odpájané spotrebiče s rôznym charakterom. Zoznam a charakter spotrebičov sa nachádza v tabuľke 1. Tabuľka obsahuje aj simulačný krok, v ktorom bol daný spotrebič pripojený prípadne odpojený.

Rovnako ako v predošlom teste, aj tu je schodovitá zmena účinníku spôsobená spínaním jednotlivých stupňov. Posun v osi Y medzi krivkami požadovanej hodnoty účinníku a meranej hodnoty je spôsobený hysteréziou regulačného pásma. Preto je aj na konci simulácie výrazný rozdiel hodnôt účinníkov na termináli B a C.



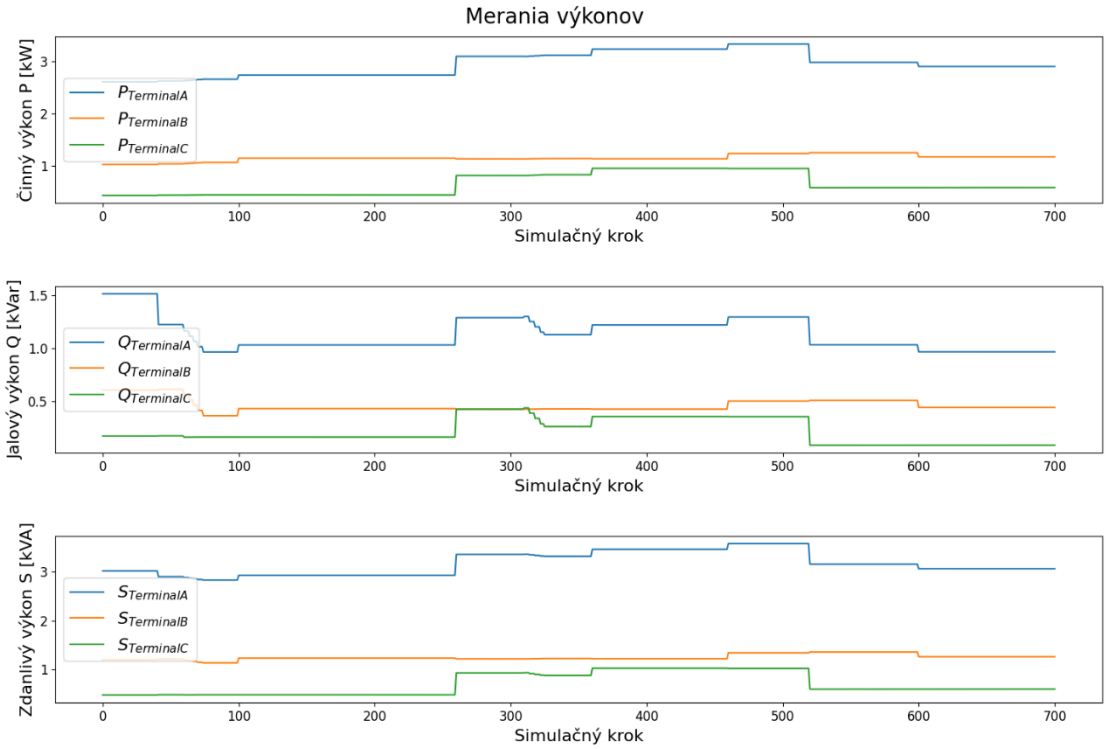
Obr. 43: Meranie účinníkov.

Tento problém by sa dal vyriešiť zmenšením hysterézie alebo jej úplným odstránením tým, že by sme jej nastavali nulovú hodnotu. V prípade nižšej hodnoty by riadiaci program vyhodnotil, že je potrebný dodatočný akčný zásah a prepočítal by optimálne rozdelenie odchýlky jalového výkonu a tiež vytvoril optimálnu sekvenciu spínania kompenzačných stupňov.

Miesto pripojenia	Simulačný krok	Názov spotrebiča	Parametre spotrebiča	Zmena systému
Terminál B	100	Motor 1	86 kW, 60 kVar	Pripojenie
Terminál C	260	VFD 1	382,6kW, 242,2 kVar	Pripojenie
Terminál C	360	Motor 2	130 kW, 85 kVar	Pripojenie
Terminál B	460	VFD 2	237kW, 151 kVar	Pripojenie
Terminál C	520	VFD 1	382,6kW, 242,2 kVar	Odpojenie
Terminál B	600	Motor 1	86 kW, 60 kVar	Odpojenie

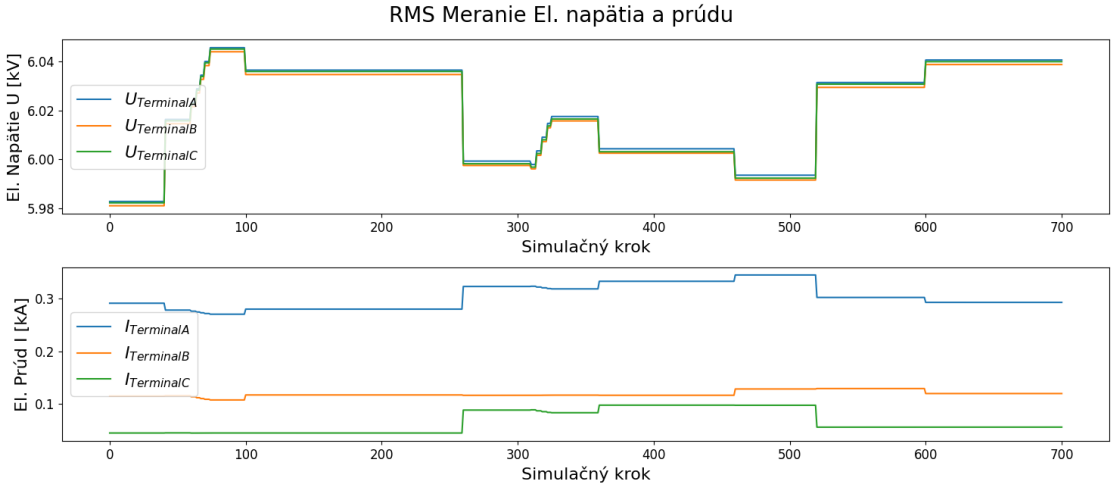
Tab. 1: Zoznam spotrebičov.

Zmenu jednotlivých výkonov, ktorý popisujú stav systému môžeme sledovať na obrázku 44. V momente, ak je do elektrickej siete pripojený nový spotrebič, tak sú zvýšené hodnoty výkonov o parametre spotrebiča. Na druhej strane, ak riadiaci systém vyhodnotí, že je nutný akčný zásah do systému pre korekciu účinníka, môžeme sledovať pokles jalového a zdanlivého výkonu, čo je spôsobené pripojením kompenzačných stupňov. Tým, že kompenzačné stupne nemajú činný výkon, nedôjde k zmene hodnôt činného



Obr. 44: Zmeny výkonov.

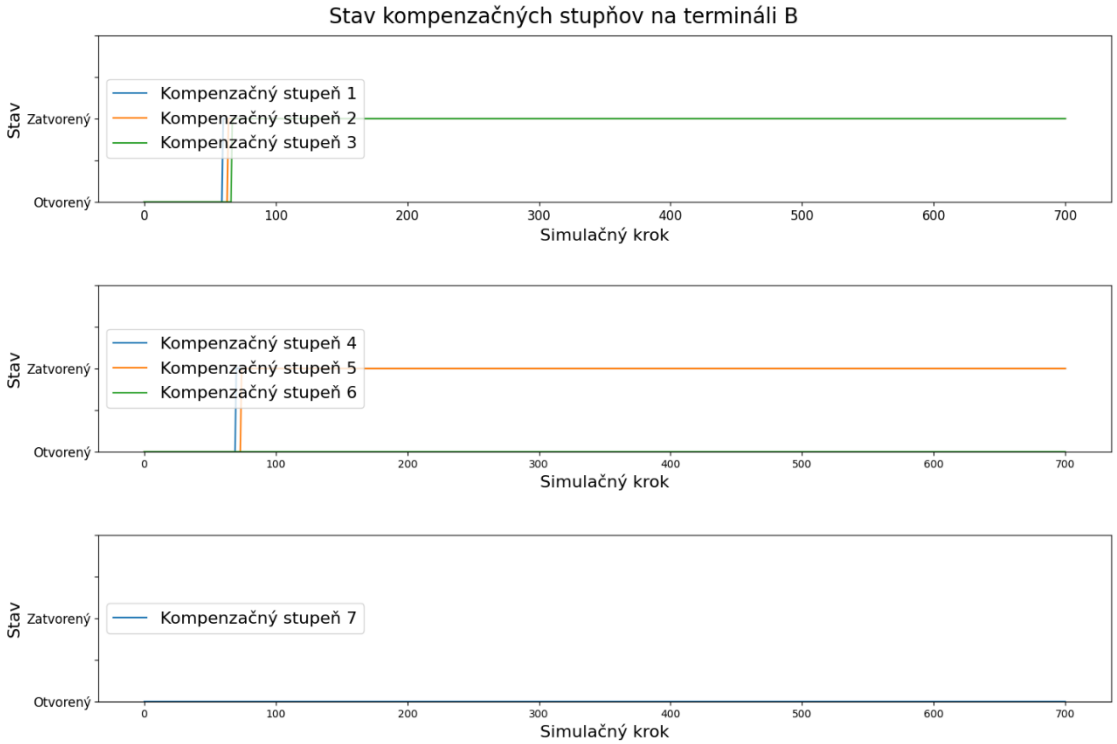
výkonu. Zmena systému je pozorovateľná aj v zmene trendov elektrického napätia a prúdu. Po pripojení spotrebiča klesne hodnota napätia o hodnotu úbytku napätia na spotrebiči. Naopak, v prípade dodanej jalovej energie kompenzačnými stupňami dôjde k zníženiu úbytku napätia, a tým sa zvýši napätie na zdroji.



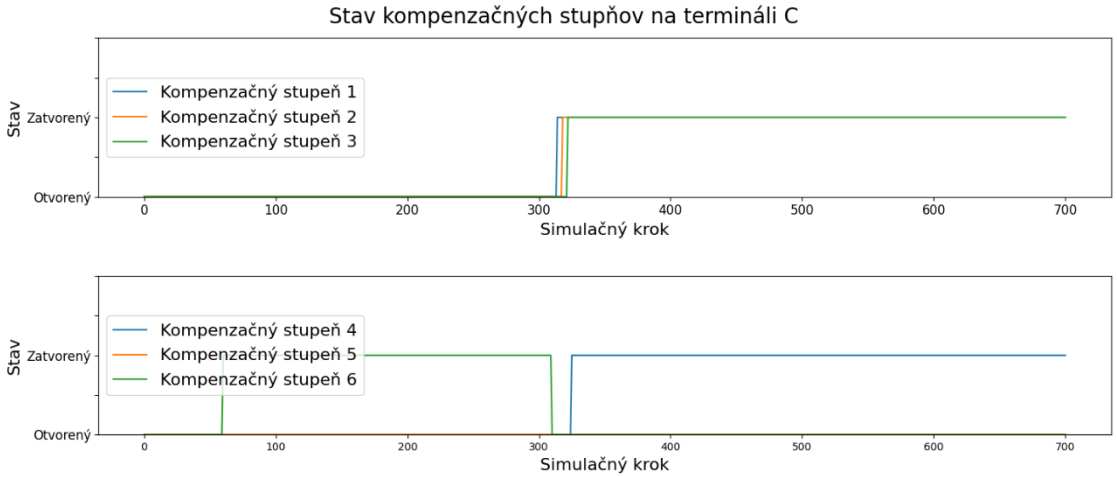
Obr. 45: Meranie el. napätia a prúdu.

Už na začiatku regulácie bolo potrebné zopnúť 300kVar kompenzačný stupeň na termináli A, aby účinník dosiahol požadovanú hodnotu. To však nestačilo a v ďalšom cykle akčného zásahu boli ďalšie kompenzačné stupne na termináli B a C. Výber stupňov je znázornený na obrázku 46 až 47. V simulačnom kroku približne 250 došlo k pripojeniu spotrebiču, ktorý spôsobil výrazný skok v jalovom výkone systému. Riadiaci systém preto vyhodnotil, že je nutné zopnúť dodatočné kompenzačné stupne na termináli C, aby účinník dosiahol požadovanú hodnotu.

Obr. 46: Zmena stavu kompenzačného stupňa na termináli A.



Obr. 47: Zmena stavu kompenzačných stupňov na termináli B.



Obr. 48: Zmena stavu kompenzačných stupňov na termináli C.

Aj z tohto testu vyplýva záver, že navrhnutý riadiaci systém funguje správne a dokáže udržiavať hodnotu účinníku na predom zvolenú požadovanú hodnotu. Je schopný optimálne rozdeliť odchýlku jalového výkonu medzi dva termináli a zvoliť optimálnu kombináciu, aby pokryl túto odchýlku.

5 ZÁVER

Hlavným cieľom diplomovej práce bolo navrhnúť automatizovaný regulátor účinníku elektrického systému. Funkčnosť návrhu bola úspešne overená testovaním vyvinutého regulátora pomocou metodiky virtuálneho uvedenia do prevádzky, v kombinovanej forme reálneho PLC a simulovaného elektrického systému.

Začiatok diplomovej práce je venovaný úvodu do teórie lineárnych výkonov bez uvažovania harmonického skreslenia. Okrem teórie lineárnych výkonov bol popísaný stručný prehľad problematiky korekcie účinníka a rôznych regulačných techník. Toto zahŕňalo aj diskusiu technických a ekonomických dopadov prevádzkovania elektrickej siete s nízkou hodnotou účinníka.

Vývoj riadiaceho systému začal virtuálnym uvedením do prevádzky regulátora a regulovaného elektrického systému. Virtuálne uvedenie do prevádzky je postavené na kooperovanej simulácii vytvorenej frameworkom Mosaik. Mosaik slúži na vytváranie kooperatívnych simulačných prostredí, čo umožňuje jednoduchú integráciu rôznych simulačných prostredí a modelov. Regulovaný elektrický systém je simulovaný v programe DigSilent PowerFactory, ktorý je špecializovaným softwarom určeným pre analýzu a simulácie elektrických sieti a systémov. Na druhej strane regulovaného systému sa nachádza navrhnutý regulátor, ktorý komunikuje s Mosaikom prostredníctvom protokolu OPC UA. Týmto spôsobom bolo úspešne vytvorené kooperatívne simulačné prostredie s charakterom regulačného cyklu so spätnou väzbou, v ktorom prebiehal vývoj a test regulátora.

Návrh regulátora vychádzal z techniky stupňovitej regulácie účinníka. Regulátor z nameraných elektrických veličín simulovaného systému vyhodnotí koľko jalového výkonu je nutné kompenzovať, tak aby bola dosiahnutá požadovaná hodnota účinníku. Kompenzácia jalového výkonu prebieha pomocou spínania kompenzačných stupňov, ktoré sú zoskupené do troch skupín. Regulátor dokáže optimálne rozdeliť odchýlku jalového výkonu medzi dva podružné termináli elektrického systému tak, aby hodnoty ich účinníkov boli čo najpodobnejšie. Regulátor zároveň obsahuje funkciu, ktorá zabezpečí optimálny výber kompenzačných stupňov s dôrazom na rovnomerné opotrebovanie. Do vývoja riadiaceho systému bol tiež zahrnutý aj návrh grafického rozhrania medzi regulátorom, regulovaným systémom a operátorom prevádzky.

V závere diplomového projektu prebehlo testovanie navrhnutého regulátora. Prvý testom bol test na reguláciu pri zmene požadovanej hodnoty a druhým bol test, kde sa menil stav systému, ale požadovaná hodnota bola konštantná. Výsledky oboch testov boli uspokojivé, riadiaci systém reguloval s dostatočnou presnosťou a za všetkých okolností sa správal podľa požadovaných kritérií.

V budúcnosti je plánovaný ďalší vývoj konceptu a rozšírenie kooperatívneho simulačného prostredia o ďalšie simulátory. Navrhnutý regulátor bude zahrnutý do portfólia produktov spoločnosti ABB a v prípade záujmu uvedený do reálnej prevádzky. Diplomová práca bude v rámci vývojového tímu súčasne slúžiť ako stručný návod k tvorbe kooperatívnych simulačných prostredí, postavených na ekosystéme Mosaik.

ZOZNAM POUŽITEJ LITERATÚRY

[1] HAMMER, Miloš. *Elektrotechnika a elektronika: přednášky*. Brno: Akademické nakladatelství CERM, 2006. ISBN isbn80-214-3334-5.

[2] WHITAKER, Jerry C. *AC Power Systems Handbook*. 3rd Edition. Morgan Hill, California: Technical Press, 2007. ISBN 0-8493-4034-9.

[3] KORENC, Vladimír a HOLOUBEK, Jiří. *Kompenzace jalového výkonu v praxi*. Knižnice Elektro. Praha: IN-EL, 1999. ISBN 80-86230-07-4.

[4] GONZALEZ-LONGATT, Francisco a RUEDA TORRES, José Luis. Introduction to Smart Grid Functionalities. In: GONZALEZ-LONGATT, Francisco a RUEDA TORRES, José Luis. *Advanced Smart Grid Functionalities Based on PowerFactory*. 30.12.2017. Springer, Cham, 2017, s. 1-18. ISBN 978-3-319-50531-2. Dostupné také z: https://link.springer.com/book/10.1007/978-3-319-50532-9.

[5] LEE, Chi G. a PARK, Sang C. Survey on the virtual commissioning of manufacturing systems. Online. *Journal of Computational Design and Engineering*. 2014, roč. 1, č. 3, s. 213-222. ISSN 2288-5048. Dostupné z: https://doi.org/10.7315/JCDE.2014.021. [cit. 2024-02-28].

[6] *DIgSILENT PowerFactory Version 2023 User Manual*. Online. September 2023. 72810 Gomaringen / Germany: DIgSILENT, 2023. Dostupné z: https://www.digsilent.de. [cit. 2024-03-03].

[7] STIFTER, Matthias; ANDRÉN, Filip; SCHWALBE, Roman a TREMMEL, Werner. Interfacing PowerFactory: Co-simulation, Real-Time Simulation and Controller Hardware-in-the-Loop Applications. Online. In: GONZALEZ-LONGATT, Francisco M. a LUIS RUEDA, José (ed.). *PowerFactory Applications for Power System Analysis*. Power Systems. Cham: Springer International Publishing, 2014, s. 343-366. ISBN 978-3-319-12957-0. Dostupné z: https://doi.org/10.1007/978-3-319-12958-7_15. [cit. 2024-04-24].

[8] LÓPEZ, Claudio David a RUEDA TORRES, José Luis. Python Scripting for DIgSILENT PowerFactory: Leveraging the Python API for Scenario Manipulation and Analysis of Large Datasets. Online. In: GONZALEZ-LONGATT, Francisco a RUEDA TORRES, José Luis (ed.). *Advanced Smart Grid Functionalities Based on PowerFactory*. Green Energy and Technology. Cham: Springer International Publishing, 2018, s. 19-48. ISBN 978-3-319-50531-2. Dostupné z: https://doi.org/10.1007/978-3-319-50532-9_2. [cit. 2024-04-24].

[9] SCHUTTE, Steffen; SCHERFKE, Stefan a TROSCHEL, Martin. Mosaik: A framework for modular simulation of active components in Smart Grids. Online. In: *2011 IEEE First International Workshop on Smart Grid Modeling and Simulation (SGMS)*. IEEE, 2011, s. 55-60. ISBN 978-1-4673-0195-4. Dostupné z: https://doi.org/10.1109/SGMS.2011.6089027. [cit. 2024-02-14].

[10] GRYGA, Lukáš. *Mosaik Framework for Co-Simulations of Smart Grids Reliability*. Bachelor's Thesis. Brno: Masaryk University Faculty of Informatics, 2019.

[11] HESS, Tobias; DICKERT, Joerg a SCHEGNER, Peter. Multivariate power flow analyses for smart grid applications utilizing Mosaik. Online. In: *2016 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. IEEE, 2016, s. 1-6. ISBN 978-1-5090-3358-4. Dostupné z: https://doi.org/10.1109/ISGTEurope.2016.7856327. [cit. 2024-02-14].

[12] OPC FOUNDATION. *The OPC Unified Architecture (UA)*. Online. 2024. Dostupné z: https://reference.opcfoundation.org/. [cit. 2024-05-13].

[13] *Python opcua-asyncio Documentation*. Online. 2022. Dostupné z: https://opcua-asyncio.readthedocs.io/en/latest/. [cit. 2024-05-13].

[14] *PLC Automation: PLCs, Control Panels, Engineering Suite AC500, CP600, ABB AbilityTM Automation Builder*. PDF. REV F. ABB, 2024.

[15] BERTSEKAS, Dimitri P. *Dynamic Programming and Optimal Control Volume 1*. Third Edition. MIT: Athena Scientific, 2005. ISBN 1-886529-26-4.

[16] UNIFIED AUTOMATION. *UaExpert*. Online. 2024. Dostupné z: https://documentation.unified-automation.com/uaexpert/1.4.2/html/index.html. [cit. 2024-05-13].

[17] POWERWORLD CORPORATION. *Power Flow: Bus Equation Basics*. Online. POWERWORLD CORPORATION. PowerWorld Corporation. 2024. Dostupné z: https://www.powerworld.com/WebHelp/Default.htm#MainDocumentation_HTML/PowerFlow%20Equation%20Basics.htm. [cit. 2024-03-06].

[18] PANDAS. *Pandas documentation*. Online. 2024. Dostupné z: https://pandas.pydata.org/docs/. [cit. 2024-05-13].

[19] MATPLOTLIB. *Matplotlib 3.8.4 documentation*. Online. 2024. Dostupné z: https://matplotlib.org/stable/users/index.html. [cit. 2024-05-13].

[20] ZELLWEGER, Urs. *Phasor diagram*. Online. In: Texample.net. 2010. Dostupné z: https://texample.net/tikz/examples/phasor-diagram/. [cit. 2024-04-24].