VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2018 Marek Magáth

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

OPC UA KLIENT PRO LABORATORNÍ MODEL "TŘÍDIČKA BEDEN"
OPC UA CLIENT FOR "SORTER OF BOXES" LABORATORY MODEL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Marek Magáth**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Miroslav Jirgl, Ph.D.**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: Marek Magáth ***ID:*** 186135
Ročník: 3 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

OPC UA klient pro laboratorní model "Třídička beden"

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit OPC UA klienta pro sběr a vyhodnocování vybraných veličin (dat) z laboratorního modelu „Třídička beden“.

1. Realizujte připojení laboratorního modelu „Třídička beden“ na PLC S7-1500 a ve vybraném jazyce v prostředí TIA Portal vytvořte vzorovou úlohu pro řízení modelu.
2. Vytvořte vizualizaci úlohy ve WinCC.
3. Seznamte se s komunikačním standardem OPC UA a stručně jej popište.
4. Nakonfigurujte PLC jako OPC UA server.
5. Ve vhodném prostředí realizujte OPC UA klienta pro sběr a vyhodnocování provozních dat z modelu s využitím vybraného databázového systému.
6. Demonstrujte funkčnost Vašeho řešení.

DOPORUČENÁ LITERATURA:

OPC Foundation. Unified Architecture. opcfoundation.org [online]. ©2017 [cit. 2018-01-11]. Dostupné z: https://opcfoundation.org/about/opc-technologies/opc-ua/

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 21.5.2018

Vedoucí práce: Ing. Miroslav Jirgl, Ph.D.
Konzultant:

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

ABSTRAKT

Táto bakalárska práca sa zaoberá komunikačným protokolom OPC, konkrétne najnovšou špecifikáciou OPC UA (*Unified Architecture*), vytvorením OPC UA klient aplikácie pomocou OPC UA .NET knižníc, ktoré ponúka zadarmo OPC Foundation a nastavením OPC servera na PLC S7-1500 pomocou *TIA Portálu V14*. Časť práce sa venuje úpravou modelu *Třídička beden*, konkrétne úpravou riadenia motorov a vymyslením komplexnejšej úlohy pre tento model. Ďalej vytvorením riadiaceho programu v grafickom jazyku *Ladder Diagram*, kde popisujem konkrétne niektoré riešenia problémov. A tak isto aj vytvorením vizualizácie pre dotykový panel TP700 Comfort. Vizualizácia je vytvorená vo *WinCC*, ktoré je súčasťou *TIA Portálu*. Na záver popisujem ako prepojiť vytvoreného OPC UA klienta spolu s dotykovým panelom a PLC, ktoré riadi tento model. A ako zbierať či pozorovať procesné dáta.

KĽÚČOVÉ SLOVÁ

OPC, OPC DA, OPC UA, SIEMENS, PLC, SIMATIC S7-1500, TP700 Comfort, TIA PORTAL V14, Ladder diagram, WinCC, PLCSIM Advanced V2.0, MS SQL, C#, WPF, XAML, Visual Studio 2017, MVVM, Repository, Unit Of Work, Messenger, Entity Framework

ABSTRACT

This bachelor's thesis is about OPC comunication protocol, specifically the newest specification OPC UA (Unified Architecture). By creating OPC UA client application using the OPC UA .NET libraries offered by the OPC Foundation for free. Setting up the OPC server on the S7-1500 PLC using the TIA Portal V14. Part of the work is done by modifying model *Třídička beden*, specifically with the modification motor control and creating more complex control task for this model. Next, by creating a control program in the Ladder Diagram, where I specifically describe some solutions of problems. And also by creating visualization on the touch panel TP700 Comfort. The visualization is created in WinCC, which is part of the TIA Portal. Finally, I describe how to link the OPC UA client created together with the touch panel and the PLC that control this model. And how to collect or observe the process data.

KEYWORDS

OPC, OPC DA, OPC UA, SIEMENS, PLC, SIMATIC S7-1500, TP700 Comfort, TIA PORTAL V14, Ladder diagram, WinCC, PLCSIM Advanced V2.0, MS SQL, C#, WPF, XAML, Visual Studio 2017, MVVM, Repository, Unit Of Work, Messenger, Entity Framework

MAGÁTH, Marek. *OPC UA klient pro laboratorní model "Třídička beden"*. Brno, 2018, 70 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedúci práce: Ing. Miroslav Jirgl, Ph.D.

VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na tému „OPC UA klient pro laboratorní model "Třídička beden"“ vypracoval samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Miroslavu Jirglovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov

Úvod

V automatizácii je veľmi dôležité zbierať z riadiacich zariadení aktuálne procesné dáta či už pre vizualizáciu, archivovanie alebo komunikáciu s inými zariadeniami či systémami. Dôležitým faktorom je taktiež vytvorenie komunikačného protokolu, ktorý bude podporovať čo najviac zariadení, a tým by sa predišlo inštalovaniu rôznych doplňujúcich programov pre rôzne protokoly. Na základe týchto požiadaviek vznikol komunikačný protokol OPC Classic, ktorý mal za účelom zjednodušiť komunikáciu medzi zariadeniami vo fabrikách. Postupom času sa zvyšovali požiadavky a bolo potrebné vylepšiť starší OPC protokol novším. Tak vzniklo OPC UA, ktoré ponúka oveľa viac možností ako jeho predchodca.

Cieľom tejto práce je zoznámiť sa s týmito komunikačnými protokolmi a opísať ich vlastnosti. Po naštudovaní týchto protokolov si budem musieť vybrať nejaké knižnice, kde bude naprogramovaná základná funkcionalita OPC UA protokolu a naštudovať ich. Aplikáciu budem písať v jazyku C# a užívatelské rozhranie pomocou WPF (*Windows Presentation Foundation*). Užívateľ by mal byť schopný pomocou tejto aplikácie pripojiť na OPC Server, ktorý sa nachádza na PLC S7-1500, prehľadávať jeho adresový priestor a zhromažďovať, sledovať a upravovať procesné dáta. Po vytvorení aplikácie sa budem zaoberať úpravou školského modelu *Třídička beden* a navrhnutím riadiacej úlohy pre tento model tak, aby sa dal použiť vo výuke. Týmto modelom sa zaoberám hlavne kvôli demonštrácii funkčnosti výslednej aplikácie, OPC klienta a prispôsobeniu modelu novej modernizovanej učebne. Pre riadenie tohto modelu budem používať PLC IMATIC S7-1500 od firmy Siemens a pre vizualizáciu dotykový panel TP700 Comfort, taktiež od firmy Siemens. Budem musieť navrhnúť riadiaci program pre celý proces triedenia, ktorý bude obsahovať zisťovanie dĺžky debničky pomocou optickej závory a rozlišovanie stlačenia červeného tlačidla, či bolo stlačené raz alebo dvakrát počas dvoch sekúnd a podľa toho rozhodnúť či ide o pauzu alebo stop. Taktiež si v tomto programe musím vytvoriť pomocné premenné pre vizualizáciu, ktorými budem kontrolovať animácie. Pre písanie riadiaceho algoritmu budem využívať grafický jazyk *Ladder diagram*. Taktiež zhotovím vizualizáciu, na ktorej bude možné sledovať aktuálny stav triedenia a taktiež tento model ovládať. Program aj vizualizácia budú zhotovené v TIA Portále V14.

1 OPC

OPC (*OLE for process control*) je priemyselný komunikačný protokol, vytvorený v spolupráci viacerých svetových dodávateľov automatizačných prostriedkov, hardwaru a softwaru v oblasti automatizácie. Je spoločným rozhraním pre vzájomnú komunikáciu medzi rôznymi zariadeniami určenými pre monitorovanie a riadenie technologických procesov. Na výmenu informácií používa architektúru klient-server. Jeho úlohou je zabrániť závislosti daného monitorovacieho alebo riadiaceho softwaru na výrobcovi hardwaru. Metódy získavania dát sú nezávislé na type pripojeného zariadenia, vďaka čomu si koncový užívateľ môže vybrať ľubovoľný software a hardware, podporujúci štandard OPC a nemusí si robiť starosti s dostupnými komunikačnými ovládačmi pre jednotlivé zariadenia. Túto špecifikáciu ďalej upresňuje nezisková organizácia OPC Foundation, ktorá doposiaľ vydala dve špecifikácie, staršiu OPC Classic a novšiu OPC UA (*Unified Architecture*).[1][8]

1.1 OPC Classic

Hlavným dôvodom vzniku OPC Classic bolo vytvorenie štandardizovaného automatizačného komunikačného protokolu, ktorý umožňoval prístup k dátam zo systémov, v ktorých bolo použité veľké množstvo rôznych zbernicových systémov, protokolov a rozhraní. V roku 1995 sa veľké spoločnosti dali dokopy v snahe definovať *Plug&Play* štandard pre zariadenia poskytnutím štandardizovaného prístupu k dátam v systémoch založených na systéme Windows. V roku 1996 vydali prvú špecifikáciu OPC Data Access, ktorá je založená na OLE/COM/DCOM technológiách vyvinuté Microsoftom. V nasledujúcich rokoch definovala OPC Foundation niekoľko softwarových rozhraní na štandardizovanie toku dát z výrobnej úrovne do úrovne riadenia. Tieto rozhrania boli vytvorené hlavne pre aplikácie priemyselnej automatizácie ako HMI alebo SCADA, ktoré spotrebujú aktuálne dáta zo zariadení a poskytujú aktuálne či historické údaje a udalosti pre aplikácie riadenia.

Na základe týchto potrieb v priemyselných aplikáciach boli definované tri hlavné OPC špecifikácie:

- *Data Access* (DA) definuje získavanie aktuálnych dát.
- *Alarm & Events* (A&E) definuje rozhranie pre udalosti ako alarmy z procesu.
- *Historical Data Accsess* (HDA) popisuje prístup k archivovaným údajom.

Všetky tri špecifikácie umožňujú navigáciu adresovým priestorom a poskytujú informácie o dostupných dátach. OPC používa klient-server architektúru pre výmenu dát. OPC server získava dáta zo zariadenia a poskytuje ich prostredníctvom svojho adresového priestoru. OPC klient môže po pripojení na server pristúpiť k dátam a pracovať s nimi.[2][8]

Dve hlavné nevýhody OPC Classic sú:[2]

- Závislosť na platforme Windows.
- Problémy s COM/DCOM pri používaní vzdialenej komunikácii, pretože je ťažko konfigurovateľné a nemôže byť používané na internetovú komunikáciu.

Ďalšie OPC špecifikácie

OPC špecifikovalo niekoľko ďalších špecifikácií pre špeciálne potreby. *OPC Security* špecifikuje, ako kontrolovať pripájanie klienta k serverom na ochranu citlivých informácií a ochranu pred neoprávnenou zmenou parametrov procesu.

Ďalej sa nemôže vynechať *OPC XML-DA*, ktorá bola ako prvá platformovo nezávislá špecifikácia a nahradila COM/DCOM s HTTP/SOAP technológiami. Kvôli tejto zmene bola akceptovaná a hneď využívaná na iných platformách ako Windows. Keďže nepotrebovala metódy na vytvorenie a úpravu komunikácie, počet metód sa skrátil na osem. Jedinou nevýhodou bol obmedzený výkon oproti klasickému OPC DA.[2][1]

1.2 OPC UA

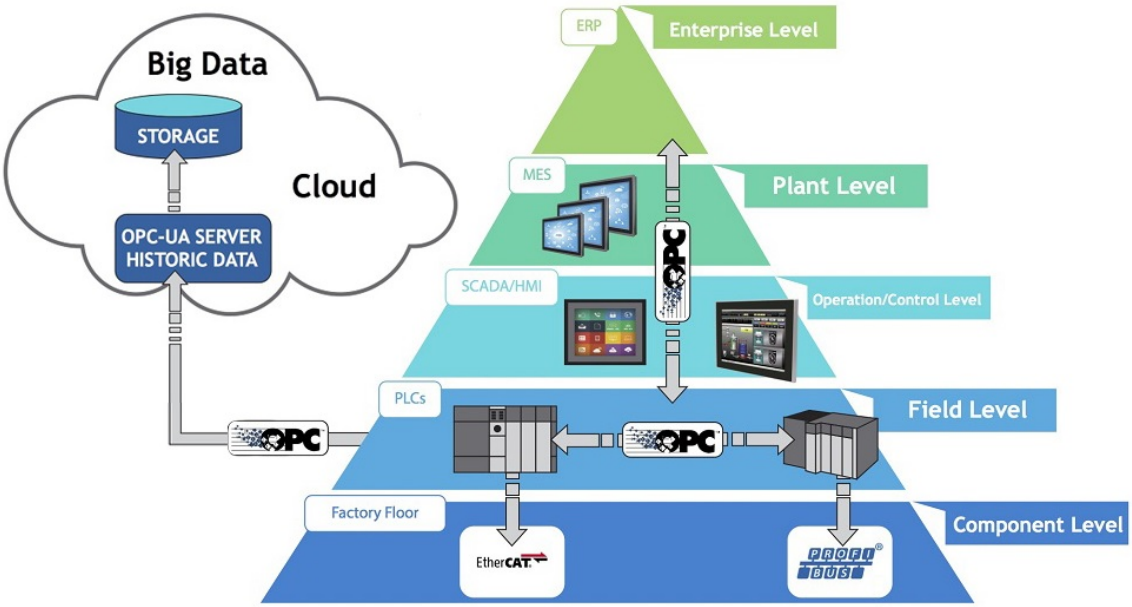
OPC UA bolo vytvorené za potreby nahradiť všetky existujúce špecifikácie založené na COM/DCOM bez straty funkcií alebo výkonu. Navyše spĺňa všetky požiadavky pre platformovo nezávislý systém s možnosťou ponúknuť viac informácií o premenných či popísať aj zložitejšie (komplexnejšie) systémy ako doteraz. Je oveľa flexibilnejšie a má oveľa viac funkcií ako všetky klasické OPC špecifikácie spolu, ale zahŕňa všetky úspešné koncepcie existujúcich OPC špecifikácií. Môže byť zabudované v senzoroch, embedded systémoch, kontroléroch, smartfónoch a taktiež v MES a ERP systémoch. Rieši známe problémy v existujúcich štandardoch a pridáva štandardizáciu pre mnoho ďalších prípadov použitia. Nie je možné porovnať všetky funkcie UA s klasickými OPC rozhraniami, ale nie je problém namapovať klasické funkcie OPC pre OPC UA.[2][4][5]

OPC UA špecifikuje dva typy serializácie/deserializácie dát.[2]

- *UA Binary* - Poskytuje rýchlu serializáciu/deserializáciu dát, ktoré majú malé rozmery a efektívny formát na fyzickej vrstve.
- *UA XML* - Je oveľa pomalšia serializácia/deserializácia dát než *UA Binary*, ale bola pridaná, aby OPC dokázalo komunikovať s inými aplikáciami na rôznych leveloch výrobného podniku.

Ďalej špecifikuje dva typy transportných protokolov, ktoré slúžia na vytvorenie spojenia medzi serverom a klientom na transportnej vrstve.[2]

- *UA TCP*
- *SOAP/HTTP(S)*



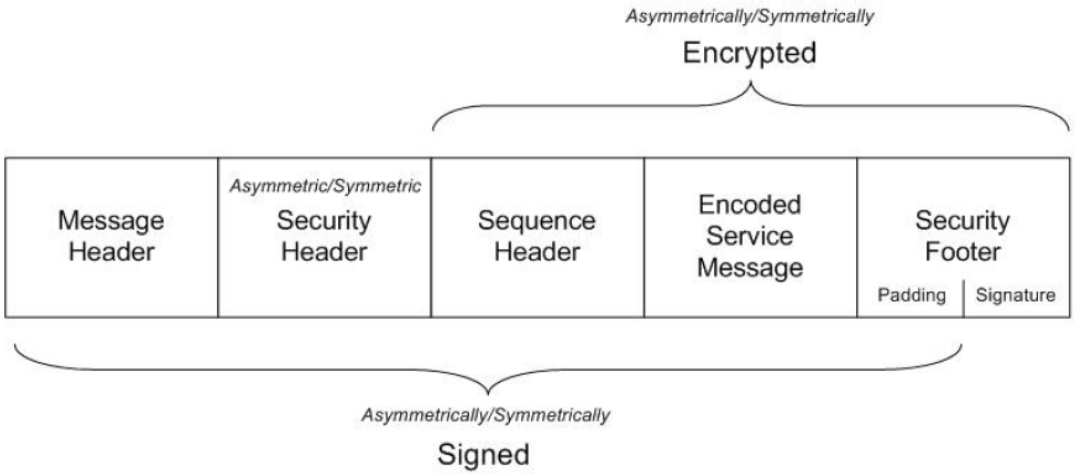
Obr. 1.1: Využitie OPC UA v podniku.[11]

1.2.1 Zabezpečenie

Keďže táto špecifikácia používa komunikáciu cez internet, tak bolo vymyslených viac vrstiev ochrany pri prenose dát.[2][4]

- *Security Policy* - určuje mieru zabezpečenia, pričom môžu byť použité zabezpečenia šifrou:
  - *None*
  - *Basic128Rsa15* : už sa nepovažuje za bezpečne.
  - *Basic256*
  - *Basic256Sha256*
- *Message Security Mode* - určuje ako budú správy zabezpečené:
  - *None*
  - *Signed* : Odosielateľ zašifruje podpis svojím súkromným kľúčom a prijímateľ si verejným kľúčom odosielateľa skontroluje či je tá správa skutočne od neho.
  - *Signed & Encrypted* : V tomto móde sa k podpisu pridá aj zašifrovanie správy verejným kľúčom prijímateľa a prijímateľ si vie svojím súkromným kľúčom správu rozšifrovať.

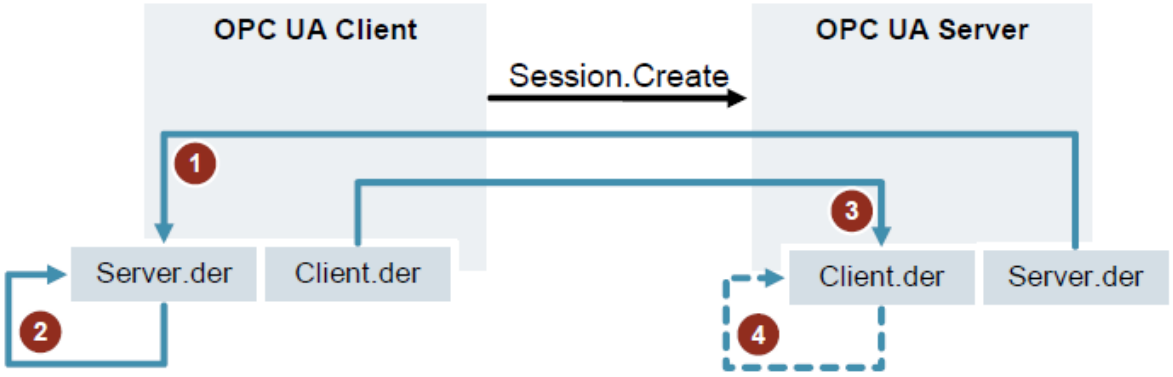
- *User Authentification* - určuje ako sa môže užívateľ prihlasovať na server.
  - *Anonymous* : anonymné prihlásenie
  - *Username* : prihlasovanie pomocou prihlasovacieho mena a hesla
  - *X.509* : prihlásenie pomocou certifikátu
  - *Issued* : prihlasovanie pomocou tokenu vygenerovaným serverom



Obr. 1.2: Zloženie podpísanej a zašifrovanej správy.[2]

Na serveri si užívateľ môže nastaviť, aké druhy zabezpečenia bude podporovať server. Na výber má aj možnosť komunikovať bez zabezpečenia, keď nastaví na serveri, že sa môže pripojiť anonymný užívateľ a povolí komunikáciu bez šifrovania. Základy mechanizmu šifrovania sú certifikáty X509, ktoré identifikujú aplikáciu na základe PKI(*Public Key Infrastructure*) klúča.

Na obrázku 1.3 je zobrazené, ako prebieha výmena certifikátu medzi klientom a serverom. Ak sa pripájame na koncový bod bez zabezpečenia, nasledujúca výmena certifikátov neprebieha.[3][5]



Obr. 1.3: Výmena certifikátov medzi klientom a serverom.[3]

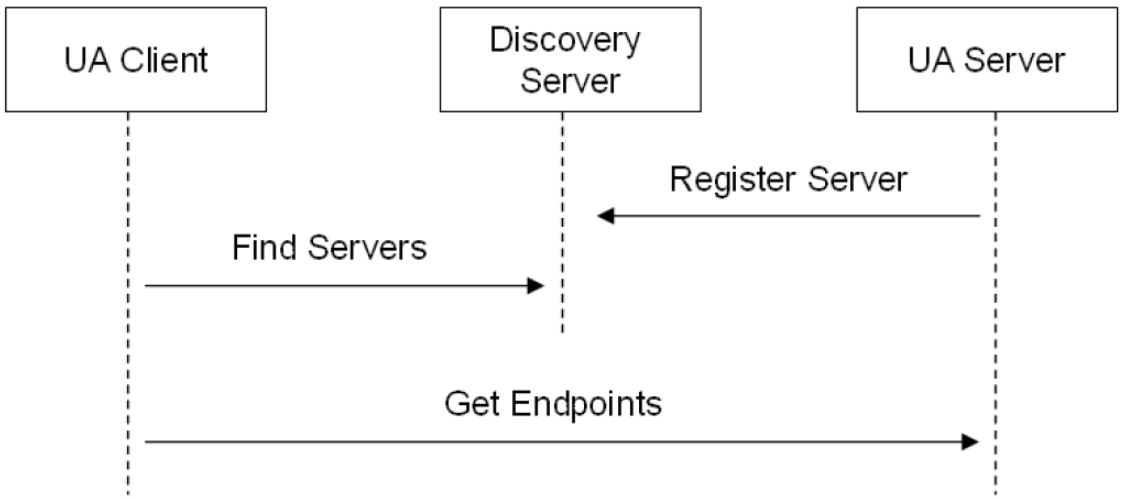
1. Keď sa klient snaží pripojiť na server, dostane od servera z daného koncového bodu verejný kľúč, ktorým bude šifrovať správy, ktoré odosiela na server.
2. Klientský program dá na výber užívateľovi či chce tento certifikát prijať alebo odmietnuť.
3. V takom istom poradí pošle klient svoj verejný kľúč na server, ale server tento certifikát zamietne a uloží ho do priečinku medzi zamietnuté certifikáty.
4. Klientský verejný kľúč musí byť potvrdený manuálne administrátorom servera. Administrátor môže tento verejný kľúč potvrdiť tak, že ho presunie z priečinku zamietnutých do priečinku vierohodných kľúčov.[3][5]

1.2.2 Vyhľadávanie OPC serverov

Na vyhľadávanie servera slúži takzvaný *Discovery server* (DS). Existujú dva typy DS:

- *Local Discovery Server* (LDS) - je potrebný, len ak je viac než jeden server dostupný na PC v lokálnej sieti. LDS beží na lokálnej sieti a vyhradenom porte 4840.
- *Global Discovery Server* (GDS)

Server sa zaregistruje na DS a klient po zaslaní požiadavky na DS dostane všetky adresy serverov, ktoré sú na ňom zaregistrované. V prípade, ak máme priamo adresu na server, môžeme tento krok preskočiť a rovno poslať požiadavku na server o zaslanie koncových bodov(*endpointov*).[7][9][10]



Obr. 1.4: Proces hľadania OPC serverov.[2]

Endpoint sa skladá z nasledujúcich informácii:[7][9][10]

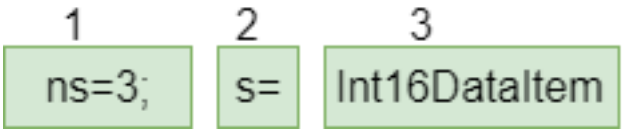
- *Endpoint URL* - adresa daného koncového bodu
- *Security Policy* - určuje algoritmus šifrovania
- *Messafe Security Mode* - určuje ochranu správ
- *User Token Type* - určuje možné druhy prihlásenia
- *Transport profile* - určuje v akom kódovaní sa odosielajú správy

1.2.3 Adresový priestor

Hlavným cieľom je sprístupniť informácie pre pripojeného klienta, aby klient vedel aké premenné, meta dáta a metódy mu server poskytuje. Celý adresový priestor sa skladá z takzvaných uzlov (*nodes*). Tieto jednotlivé uzly sú medzi sebou poprepájané referenciami a spolu tvoria hierarchickú štruktúru (môžeme si to predstaviť ako štruktúru súborov). Existuje osem typov uzlov: *Object*, *Variable*, *Method*, *View*, *ObjectType*, *VariableType*, *ReferenceType*, *DataType*.

Uzle poskytujú informácie (atribúty) o sebe (id, typ uzlu, meno, popis a údaj o tom, kto môže tento uzol upravovať). Tieto informácie sú spoločné pre všetky typy uzlov, ďalšie informácie sú už pre každý uzol špecifické. Napríklad uzol typu *Variable* má v sebe údaje ako hodnotu, dátový typ, či je to skalárna hodnota alebo pole hodnôt, kto ma oprávnenie čítať/zapisovať túto hodnotu, minimálny vzorkovací interval a či sa má táto hodnota archivovať.[7][4][2]

Každý uzol musí mať špecifické id. Tento kľúč sa skladá:[7]



Obr. 1.5: Zloženie id uzla.

1. *Namespace index* - Slúži na rozlíšenie uzlov z rôznych typov subsystémov. Index 0 je vyhradený vždy pre server.
2. *Id Type* - Určuje akého typu je identifikátor (*s=String*; *i=Numeric*; *g=GUID*; *b=OPAQUE* (*ByteString*))
3. *Id* - Jedinečný identifikátor daného uzla.

1.2.4 Služby

Definujú komunikáciu na aplikačnej úrovni. Sú to metódy slúžiace na získanie dát zo servera. Definované služby sú nezávislé na transportnom protokole a programovacom prostredí používané na vývoj OPC UA aplikácii (narozdiel od OPC Classic). Definície služieb používajú požiadavka/odpoveď vzor ako pri webových službách. OPC UA definuje 37 služieb, z toho 21 je určených na komunikáciu a 16 je určených na výmenu informácii.[2][6]

Rozdelenie služieb do skupín:[2]

- Hľadanie serverov
  - *FindServers*
  - *GetEndpoints*
  - *RegisterServer*
- Vytváranie pripojenia medzi klientom a serverom
  - *OpenSecureChannel/CloseSecureChannel*
  - *CreateSession/CloseSession*
  - *ActivateSession*
  - *CancelService*
- Hľadanie informácii v adresovom priestore
  - *Browse*
  - *BrowseNext*
  - *TranslateBrowsePathsToNodeIds*

- Čítanie/Zapisovanie dát a meta dát
  - *Read*
  - *Write*
  - *RegisterNodesService*
  - *UnregisterNodesService*
- Odoberanie zmien hodnôt a udalostí
  - *CreateSubscriptionService*
  - *DeleteSubscriptionsService*
  - *ModifySubscriptionService*
  - *SetPublishingModeService*
  - *TransferSubscriptionsService*
  - *CreateMonitoredItemsService*
  - *DeleteMonitoredItemsService*
  - *ModifyMonitoredItemsService*
  - *SetMonitoringModeService*
  - *SetTriggeringService*
  - *Publish*
  - *Republish*
- Volanie metód definované na serveri
  - *Call*
- Prístup k archívu dát a udalostí
  - *HistoryRead*
  - *HistoryUpdate*
- Hľadanie informácií v zložitom adresovom priestore
  - *QueryFirst*
  - *QueryNext*
- Upravovať adresový priestor servera
  - *AddNodes*
  - *DeleteNodes*
  - *AddReferences*
  - *DeleteReferences*

Spracovanie chýb

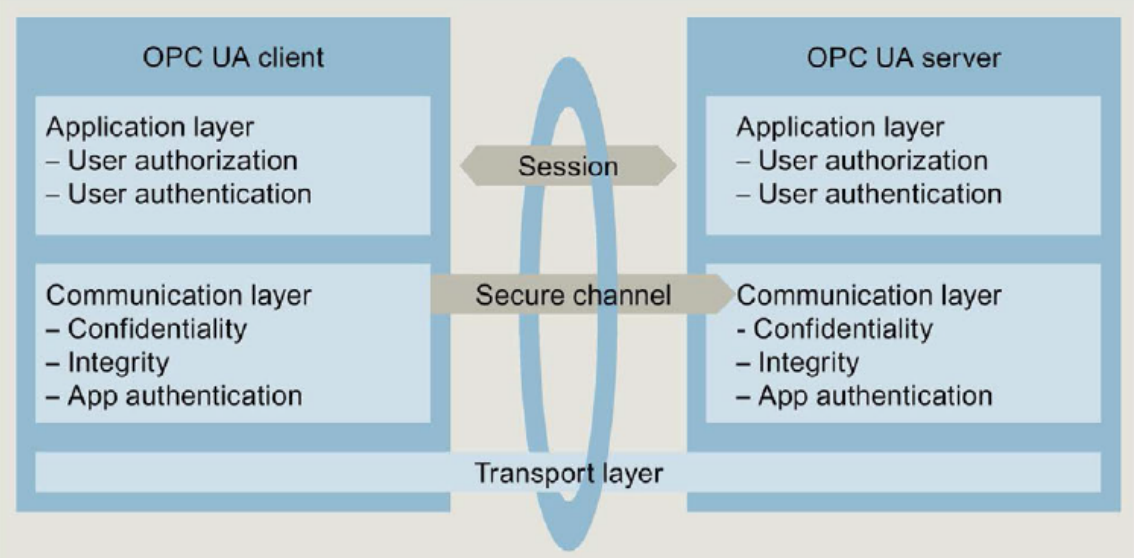
Keďže OPC UA komunikácia je navrhovaná na komunikáciu medzi rôznymi systémami, ktoré bežia v rôznych sieťach a procesoch, tak komunikácia medzi týmito systémami môže byť ľahko narušená. Preto je potrebné zvládať tieto situácie, aby nedošlo k strate dát. Preto sú definované maximálne časy na odpoveď pri službách, aby sa prípadné narušenie komunikácie zistilo.[2][6]

Spracovanie chýb pri službách je veľmi dôležité, keďže chyba môže nastať v rôznych častiach aplikácie a v rôznych prípadoch. Chyba môže nastať, ak užívateľ zadá zlé parametre alebo nastane chyba v komunikácii na niektorej vrstve. Preto každá služba vracia kód, ktorý hovorí o tom ako bola služba vykonaná. Tento kód je 32 bitový bezznamienkový integer, kde najvyšších 16 bitov predstavuje hodnotu chyby a najnižších 16 slúži na dodatočné informácie. Dva najvyššie bity hovoria o tom, či je návratový kód *good*, *uncertain* alebo *bad*.[2][6]

1.2.5 *Secure Channel* a *Session*

Pri vytváraní pripojenia medzi klientom a serverom sa najskôr vytvorí *Secure Channel*, ktorý má za úlohu zabezpečiť nízkoúrovňovú komunikáciu, na vrchu ktorej sa vytvorí *Session*. *Secure Channel* a *Session* nemajú za úlohu prenášať dáta, ale len zabezpečiť spoľahlivú a bezpečnú komunikáciu. Život *Session* je nezávislý na *Secure Channel* a pri vypadnutí spojenia sa vytvorí nový *Secure Channel*, ktorý sa pridá k existujúcej *Session*. *Session* má definovaný čas, za ktorý keď neprebehne žiadna komunikácia medzi serverom a klientom, server uvoľní všetky prostriedky súvisiace s touto *Session*. Tento čas sa resetuje s každou požiadavkou na server zo strany klienta.

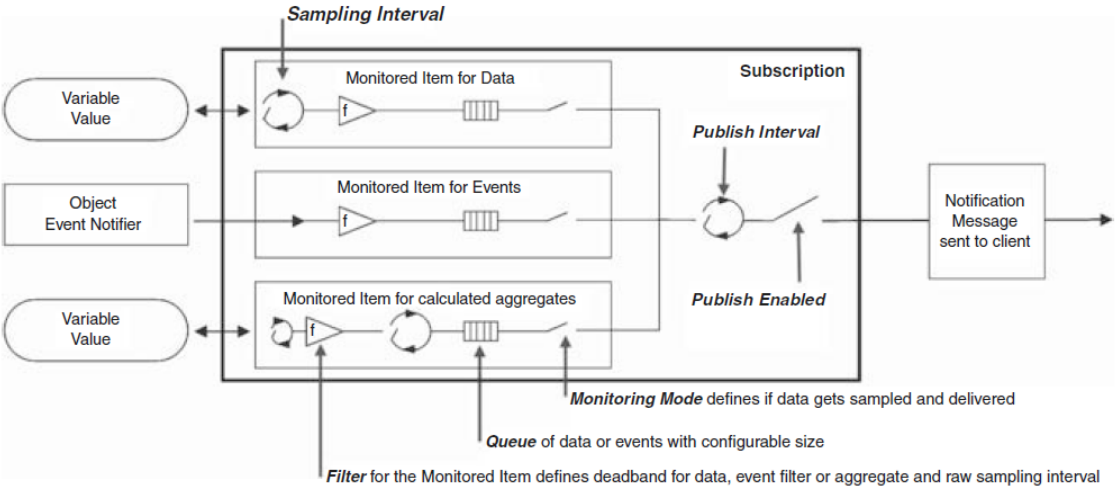
Pri požiadavke klienta o vytvorenie *Session*, server zašle identifikačný kľúč, ktorý sa používa vo všetkých ostatných službách, aby server vedel od koho prichádzajú požiadavky.[2]



Obr. 1.6: Úlohy pre *Session* a *Secure Channel*.[2]

1.2.6 *Subscription*

Klient môže odoberať(*Subscribe*) tri typy informácií, zmeny hodnoty, udalosti a agregované hodnoty. Klient pri vytváraní *Subscription* určuje, ako často mu má server posielať zmenené hodnoty odoberaných premenných, tento parameter sa nazýva *Publish Interval*. Ďalej si vytvára z premenných, ktoré chce sledovať takzvané *Monitored Items*, ktorými nastavuje pri akej zmene hodnoty(filter) má server odoslať notifikáciu, ako často má server vzorkovať(*Sampling Interval*) danú premennú, koľko zmien si má uchovať pokiaľ neprebehne odoslanie na server a typ monitorovacieho módu.[2][4][7]



Obr. 1.7: Implementácia *Subscription* na strane servera.[2]

Monitorovanie premenných:[2]

- Server vzorkuje odoberané premenné podľa definovaného času (*Sampling Interval*)
- Po navzorkovaní vo filtre skontroluje či sa zmenila hodnota o požadovanú hodnotu alebo percento.
- Ak hodnota prejde cez filter premenná sa zaradí do rady. Dĺžka tejto rady sa definuje klientom. Ak je už rad plný, tak sa prepisuje prvá alebo posledná hodnota v rade podľa nastavenia klienta.
- Po uplynutí *Publish Interval* sa hodnoty uložené v rade odošlú klientovi len ak je nastavený monitorovací mód *Reporting*. Ak je nastavený mód *Sampling*, tak sa hodnoty neodosielajú, len sa ďalej vzorkujú.

Na tomto princípe fungujú aj udalosti s tým rozdielom že udalosť nastáva sama, nemusí sa vzorkovať a klient si nastaví vo filtri aké typy udalostí chce dostávať. Agregované hodnoty sa rátajú až za filtrom.[2]

1.2.7 Čítanie/zapisovanie hodnôt

Ak poznáme id uzlov premenných, s ktorými potrebujeme komunikovať, môžeme vykonávať čítanie/zápis. Po každom čítaní/zápise dostaneme status kód, ktorý nám hovorí ako daná operácia skončila. Pri načítani dostaneme spolu s hodnotou dve časové známky, z toho jedna je čas, kedy server odoslal hodnotu a druhá patrí času, v ktorom premenná túto hodnotu nadobudla. Pri zápise môžeme spolu s hodnotou určiť aj časovú známku, ale treba dbať na to, že každý server túto možnosť nepodporuje.[2][4]

Ak má premenná id uzlu číslo a nie string, prístup k tejto hodnote na serveri je oveľa rýchlejší. Ak vieme, že budeme nejaké premenné načítavať často, v určitých intervaloch, mali by sa tieto id uzlov zaregistrovať na serveri pomocou služby *RegisterNode*. Ak daný uzol zaregistrujeme, dostaneme naspať nove id uzla, ale už v numerickej podobe. Pre ďalšie čítanie a zápis je potrebné použiť páve numerické id. Pri rýchlo opakovanom čítaní sa odporúča použiť radšej *Subscription* namiesto klasického čítania.[2][4]

1.2.8 Štruktúra OPC UA server a klient aplikácie

Typická OPC UA aplikácia sa skladá z troch vrstiev na strane klienta a z troch vrstiev na strane servera. Podrobné zobrazenie štruktúry aplikácie je v prílohe A.1.[2][5]

- *Stack* - Obsahuje funkcionalitu najnižšej úrovne ako vytvorenie spoľahlivej a bezpečnej komunikácie, kódovanie/dekódovanie správ, zabezpečenie správ a nadefinované všetky základné dátové typy. *Stack* sa používa na vývoj klient aj server aplikácií, keďže obsahuje funkcionalitu, ktorú používajú obe strany. Platformovo špecifická vrstva obsahuje platformovo špecifický kód ako knižnice na vlákna, kryptografiu, ukladanie certifikátov a tak ďalej.
- *SDK* - Obsahuje už všetku potrebnú funkcionalitu na vytvorenie plnohodnotnej OPC UA aplikácie ako napríklad všetky služby na vytvorenie komunikácie medzi klientom a serverom.
- *Application* - V tejto časti sa nachádzajú funkcie pre špecifické prípady ako aj logika celej aplikácie.

OPC Foundation ponúka zadarmo *OPC UA Stack* v jazykoch C/C++, C# (.NET Framework, .NET Standard) a v Jave. Tento *Stack* obsahuje len základnú funkcionalitu. Pre C# ponúkajú aj ďalšiu knižnicu s názvom *UA Client library*, kde sú nadefinované všetky služby ako *Session*, *Subscription* a tak ďalej. Tieto dve knižnice (*Stack* a *UA Client*) ponúkajú naimplementovanú v *.NET Framework* aj v *.NET Standard*. Rozdiel medzi *.NET Frameworkom* a *.NET Standard* je, že aplikácia založená na *.NET Framework* beží len na platforme Windows a aplikáciu založenú na *.NET Standard* môžeme spustiť na Mac/Linux/Windows/Android. OPC UA Client a Server SDK už nie sú voľne dostupné, pretože ich vyvíjajú súkromné spoločnosti. Dajú sa ale stiahnuť demo knižnice, na ktorých sa dá rozbehnúť a vyskúšať OPC aplikáciu, ale po určitom čase sa aplikácia sama vypne.

2 OPC UA server na PLC *S7-1500*

OPC UA server bol pridaný do všetkých PLC rady S7-1500 s firmwarom 2.0 a dá sa nastavovať len v TIA Portáli od verzie 14. Táto zmena pridala možnosť priamej výmeny dát tohto PLC so širokou škálou ďalších systémov, ktoré podporujú OPC UA.

2.1 Implementácia

OPC server, vytvorený spoločnosťou Siemens, ktorý sa nachádza v PLC nepodporuje zatiaľ všetky služby, ktoré definovali tvorcovia komunikačného protokolu OPC UA. S príchodom TIA Portálu V14 a firmwarom 2.0 OPC server podporoval:

- čítanie hodnôt
- zapisovanie hodnôt
- registrované čítanie / zapisovanie
- oznamovanie (*Subscription*)

Tieto služby sa týkajú, hlavne získavaniu aktuálnych procesných dát (*Data-Access*). Alarmy, udalosti a archivovanie dát nie sú zatiaľ v tomto serveri implementované.

Príchodom TIA Portálu V15 s firmwarom 2.5 je možné v PLC definovať metódy, ktoré je možné spúšťať pomocou OPC klienta. Týmto metódam sa dajú definovať vstupné parametre a taktiež návratové parametre. Ďalej pridali ďalšie nastavenia pre OPC server.

OPC server podporuje len prenos správ ako binárny tok cez TCP/IP. Prenos správ pomocou XML cez TCP/IP a HTTPS server zatiaľ neimplementuje.

Ďalej v adresovom priestore servera môžme nájsť informácie o serveri, typu PLC a CPU, systémové tagy a dáta bloky, ktoré majú povolenú prístupnosť cez OPC UA. Adresový priestor pre S7-1500 má *namespace index* 3 a id uzlov sú názvy premenných ako v PLC s tým rozdielom, že sú medzi úvodzovkami, aby sa predišlo konfliktom s pomenovaním. Id uzlu z PLC S7-1500 môže vyzerať nasledovne - *ns=3;s="DB1"."myInt16"*.

Ak je CPU v stope, OPC server ostáva pracovať naďalej. Ak si počas tohto stavu chceme načítať nejakú premennú, dostaneme naspať poslednú navzorkovanú. Tak isto môžeme v tom čase zapísať hodnotu na server, ale CPU túto hodnotu nespracuje, lebo program sa nemôže vykonávať v stop móde. Ak sa pokúsime zavolať metódu dostaneme chybný kód. Každým novým nahratím programu do CPU sa OPC server reštartuje.[3][4]

2.2 Konfigurácia

Pri vytvorení nového projektu v TIA Portáli V14 je OPC server predvolene vypnutý. Zapnúť ho môžeme nasledovne: V ľavom menu (*Project tree*) si otvoríme zložku s názvom nášho projektu. Po otvorení tejto zložky môžeme vidieť možnosť pridať nové zariadenie do projektu a všetky zariadenia v našom projekte (PLC, HMI..). Po rozkliknutí záložky s PLC, ktorému chceme zapnúť OPC server vyberieme možnosť *Device configuration*. Otvorí sa nám okno s našim PLC. V dolnej časti obrazovky sa nachádza menu, v ktorom musíme zvoliť v pravom hornom rohu záložku *Properties*, ktorá nám zobrazí nastavenia s názvom *General* pre naše PLC.

V tomto menu vyberieme:

- *OPC UA*
  - *General*: Môžeme nastaviť názov OPC servera.
  - *Server*
    - *General*: Zaškrtneme políčko *Activate OPC UA server*, pod tým môžeme vidieť adresu servera. Všimnime si, ak nemáme na PLC nastavenú pevnú IP adresu, ale máme nastavené že IP adresa je nastavená priamo v zariadení (*PROFINET interface -> IP protocol*), adresa vypadá nasledovne *opc.tcp://<dynamically>:4840*. V takomto prípade sa adresa určí až po nahratí projektu do PLC, kde *<dynamically>* nahradísa IP adresa PLC.
    - *Options*: Tu si vieme nastaviť port, na ktorom bude bežať OPC server, predvolený je 4840. Ďalej si tu vieme nastaviť minimálny zverejňovací interval a minimálny vzorkovací interval, ktorý bude server podporovať. Viac o týchto atribútoch je v sekcii 1.2.6.
    - *Security*: Môžeme tu zmeniť certifikát. Ďalej si tu môžeme nastaviť, aké zabezpečenia bude server podporovať pri komunikácii. Viac v sekcii 1.2.1. Potom máme na výber či chceme pre každé nové pripojenie klienta pridávať klientov certifikát ručne medzi dôveryhodné, alebo zaškrtneme políčko *Automatically accept all client certificates during runtime*, ktorým povolíme automatické akceptovanie všetkých príchodzích certifikátov.
    - *User authentication*: V tejto záložke môžeme povoliť/zakázať anonymné a meno/heslo prihlasovanie. Taktiež si tu môžeme vytvoriť účty, pomocou ktorých sa bude dať pripojiť pomocou OPC klienta.
  - *Export*: Môžeme vygenerovať XML dokument, ktorý bude obsahovať všetky PLC tagy a dáta blok premenné, ktoré majú zaškrtnutú možnosť *Accesible from HMI/OPC UA* v dáta bloku. Tak isto môžme toto povolenie nastaviť globálne v nastaveniach dáta bloku.

DataBlock

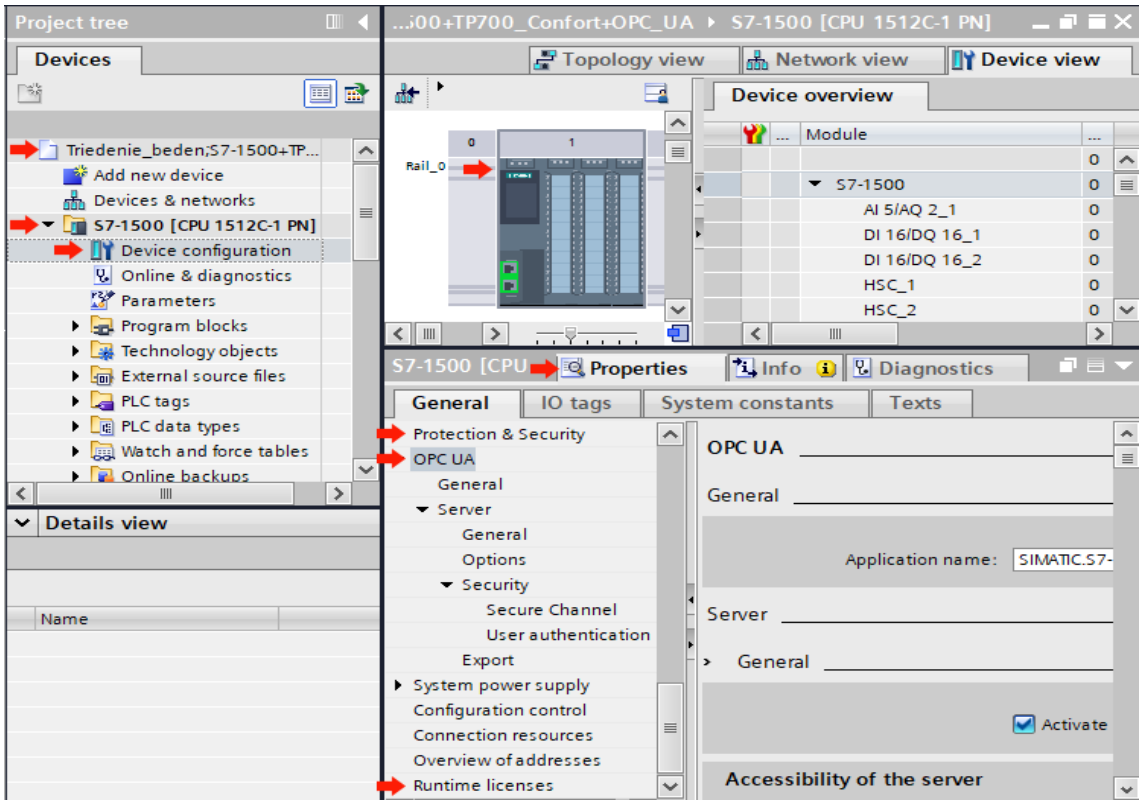
	Name	Data type	Offset	Start ...	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA
1	Static				☐	☐	☐
2	scan_Time_sens1	Time	0.0	T#0ms	☑	☑	☑
3	hold_scan_time_s...	Time	4.0	T#0ms	☑	☑	☑
4	pocet_malych	Int	8.0	0	☑	☑	☑

Obr. 2.1: Nastavenie premenných v dáta bloku.

– Ako posledné treba nastaviť licenciu pre OPC server v poslednej položke *Runtime licenses*, bez toho by sa nepreložil projekt. Stačí vybrať typ potrebnej licencie, ktorú tam zobrazuje. Typ licencie závisí od typu PLC, viď 2.2.

CPU type	ET 200SP CPU up to S7-1513(F)	1515 / 1516(F)	1517 / 1518(F)
Required license	Small	Medium	Large

Obr. 2.2: Typy licencii.[3]



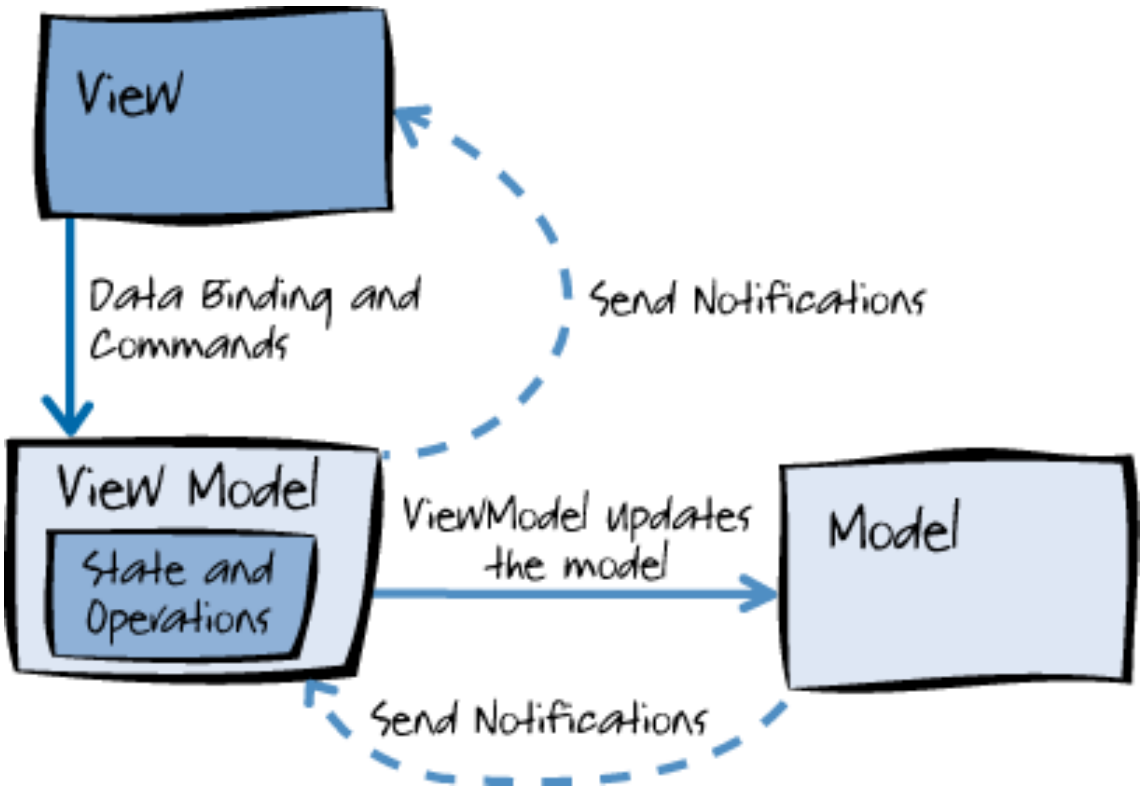
Obr. 2.3: Konfigurácia OPC servera v TIA portály V14.

3 OPC UA klient aplikácia

Aplikácia je vytvorená v jazyku C# *.NET Framework 4.6.2* a užívateľské rozhranie vo *Windows Presentation Foundation* (WPF). WPF je technológia vyvíjaná *Microsoftom*, ktorá v roku 2006 nahradila už dnes zastaralú technológiu *WinForms*. Užívateľské rozhranie sa definuje jazykom XAML. Je to deklaratívny značkovací jazyk odvodený od XML. WPF je založené na vektoroch, čiže pri zmene rozlíšenia monitora alebo približovaní a vzďaľovaní aplikácie nedochádza k strate kvality. Všetok XAML kód sa spracováva na grafickej karte, čo odľahčuje procesor.

3.1 Architektúra aplikácie

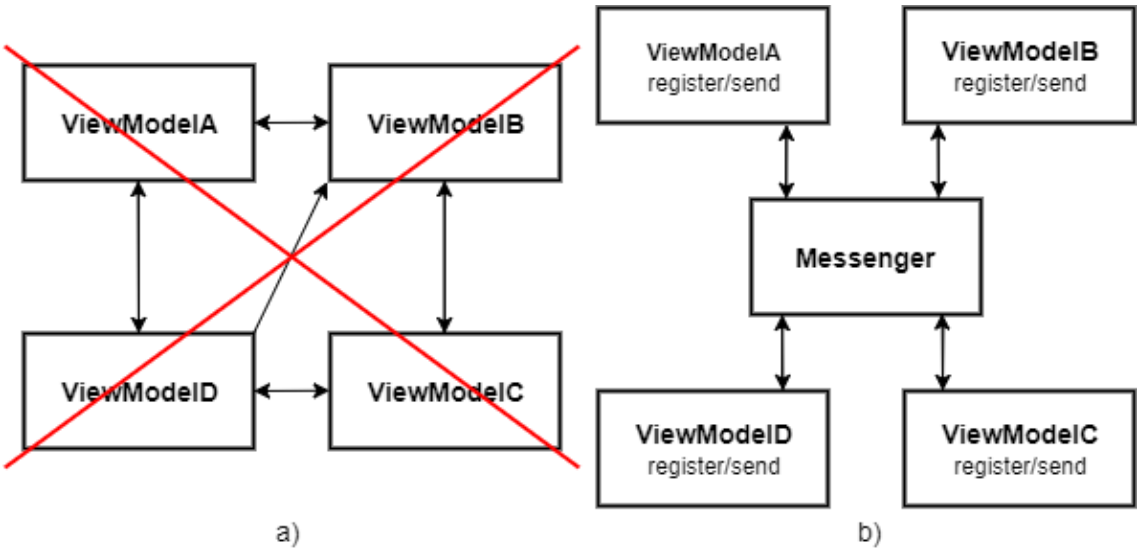
Aplikácia je navrhnutá podľa návrhového vzoru *Model-View-ViewModel* (MVVM). Medzi najväčšie výhody patrí oddelenie logiky zobrazovania od logiky programu, testovateľnosť kódu, znovu použiteľnosť kódu a lepšia udržovateľnosť kódu.



Obr. 3.1: Architektúra návrhového vzoru MVVM.[12]

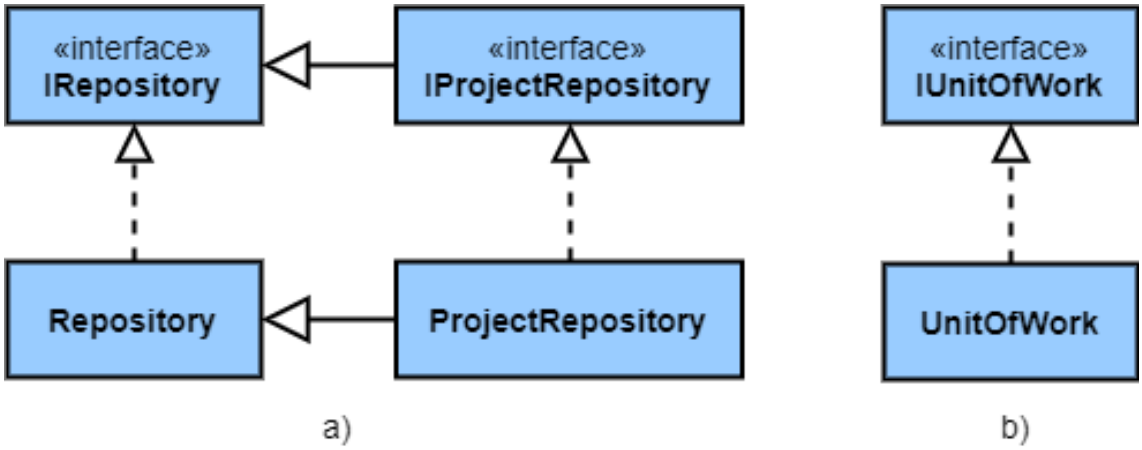
- ***View*** - Zobrazuje dáta užívateľovi a dáva mu možnosť ovládať program, či pridávať/upravovať/mazať dáta. Komunikuje s *View-Modelom* pomocou *data bindingu* a *commandov*. Dátové typy sa preväzujú s grafickými prvkami pomocou *data bindingu* a pri stlačení tlačítka sa namiesto udalostí (*events*), ktoré sa používajú v *Code-Behind*, používa rozhranie *ICommand*, pomocou ktorého vieme vykonať metódu vo *View-Modely*. Príklad *bindingov* sa nachádza v prílohe B. *View* má referenciu na *View-Model* a nie naopak, z čoho vyplýva výhoda znovu použiteľného kódu, kedy môžme nahradiť view a celá logika programu sa môže znova použiť.
- ***ViewModel*** - Drží kontext aktuálne zobrazenej obrazovky a pomocou rozhrania *INotifyPropertyChanged* notifikuje *View* o zmene dát. Žije po celú dobu zobrazenia. Nemá referenciu na *View*.
- ***Model*** - Reprezentuje dáta.

Na komunikáciu medzi *View-Modelmy* používam návrhový vzor *Messenger*. Používa sa kvôli tomu, aby nemusel mať každý *View-Model* referencie na ostatné *View-Modely*. Takto má referenciu len na *Messenger*, cez ktorý posiela správy. Je to vlastne prostredník, cez ktorý komunikujú všetky *View-Modely*. *View-Modely* si zaregistrujú typ správy aký chcú prijímať a *Messenger* im ich rozosiela.



Obr. 3.2: a) Nesprávna komunikácia medzi *View-Modelmy*. b) Komunikácia pomocou *Messengeru*.

Pre komunikáciu s databázou používam návrhový vzor *Repository* a *UnitOfWork*. *Repository* vzor sa používa na abstrakciu pri komunikácii databáze s *Bussines* vrstvou, tak aby *Bussines* vrstva nemala referenciu na databázu a zložitá logika filtrovania dát bola mimo tejto vrstvy na jednom mieste. Na nasledujúcom obrázku je zobrazený diagram tried tohto návrhového vzoru. Skladá sa z generického repozitáru, ktorý má metódy, ktoré potrebuje každý repozitár a to *GetById()*, *Add()*, *Remove()*, *Find(predicate)*. Ďalej ho rozširuje už špecifický repozitár pre každú tabuľku. Na obrázku je ako príklad použitý repozitár pre projekt, kde môže byť naimplementovaná metóda *GetAllWithEndpoints()* a ďalšie špecifické metódy podľa potreby.



Obr. 3.3: a) Implementácia návrhového vzoru *Repository*. b) Implementácia návrhového vzoru *UnitOfWork*.

UnitOfWork nám obaľuje celú logiku nad databázou. Drží v sebe jedinú referenciu na databázu v aplikácii. Stará sa o vytváranie instancii definovaných repozitárov, ktorým predáva túto instanciu databáze. *Bussines* vrstva má len referenciu na rozhranie *IUnitOfWork* a nie na implementáciu. *IUnitOfWork* má v sebe referenciu na rozhrania repozitárov. Takto pri zmene databázového systému zmeníme kód len na jednom mieste a to v triedach repozitárov, ktoré implementujú tieto rozhrania.

Celá aplikácia je rozdelená v štyroch projektoch. Z toho *Opc.Ua.Core* a *Opc.Ua.Client* sú externé knižnice, ktoré ponúka zadarmo OPC Foundation a je v nich nadefinovaný OPC UA komunikačný protokol, viac v sekcii 1.2.8. Zjednodušená bloková schéma aplikácie sa nachádza v prílohe C.1.

Hlavný projekt sa nazýva *OpcUa.Client.WPF* a obsahuje:

- Pohľady(*views*) - Vzhľad stránok a užívateľských kontroliek, z ktorých sa tieto stránky skladajú.
- Vlastné nadefinované atribúty pre XAML elementy.
- Konvertory, ktoré konvertujú jeden typ na druhý, väčšinou nejaký dátový typ na UI vlastnosť (napr. Ak bude vek užívateľa menší ako 18, zobraz text červenou farbou).
- Implementácia rozhrania *ICommand* s názvom *RelayCommand*, ktorá slúži na prepojenie tlačidiel s metódami vo *View-Modely*.
- *View-Modely*

Moja implementácia *RelayCommand* má dva parametre.

- Metóda s návratovou hodnotou void, ktorá sa má vykonať pri stlačení tlačidla.
- Metóda s návratovou hodnotou bool, ktorá určuje, kedy je tlačidlo stlačiteľné.

Použitie môžte vidieť v prílohe B.

Projekt s názvom *OpcUa.Client.Core* obsahuje:

- *Bussines vrstva*
  - Modely použité vo *View-Modeloch*.
  - *Mapper*, ktorý mapuje modely databázových tabuliek na modely vo *View-Modeloch* a naopak.
  - Naimplementovaný *Messenger* a správy.
  - Zhrnutú funkcionalitu OPC UA knižníc pre moju aplikáciu v triede s názvom *OpcUaClientApi*.
  - Repozitár pre každú databázovú tabuľku.
  - Naimplementovaný *UnitOfWork*.
- *Data Acces vrstva*
  - Modely databázových tabuliek.
  - Konfiguráciu databázy a migrácie vygenerované *Entity Frameworkom*.
  - Triedu databázy.

Týmto rozdelením projektov som sa snažil dosiahnuť oddelenie kódu, ktorý je viazaný k platforme Windows. Po prechode z OPC UA knižníc založených z *.NET Framework* na *.NET Standard* by sa dalo *OpcUa.Client.Core* projektu zmeniť framework na *.NET Standard*, tým pádom by sme mohli tento kód skompilovať aj na iných platformách a pridať nové *View*, ktoré by sme napojili na existujúce *View-Modely*, tak isto ako to na platforme Windows.

3.2 Nuget balíčky

V aplikácii som použil niekoľko balíčkov, ktoré mi uľahčili tvorbu aplikácie.

- **LiveCharts** Tento balíček používam na vykresľovanie grafov. Dokáže vykresliť rôzne typy grafov. Grafu môžeme nastaviť formát ôs, legendu, informáciu k vykreslenému bodu v grafe, približovanie v rôznych osiach, pohyb v grafe do všetkých strán a samozrejme štýlovanie jednotlivých čiar v grafe.
- **PropertyChanged.Fody** zjednodušuje písanie kódu tým, že nemusím v každom *View-Modely* pri každej *property* notifikovať *view* o zmene ako v príklade B, ale tento balíček to robí za nás. Vo výsledku nám sprehľadní a zjednoduší *View-Model* triedy.
- **Ninject** je balíček, ktorý pomáha používať techniky *Dependency Injection* a *Inversion of Control* a odstraňuje závislosť jednej časti aplikácie na druhej, kde aplikácia je závislá na rozhraní a nie implementácii.
- **EntityFramework** slúži na komunikáciu medzi aplikáciou a databázou. Je to objektovo-relačný mapér, tj. mapuje dáta z databázových tabuliek do objektov a hlavne za nás píše SQL dotazy na databázu.
- **MaterialDesign** obsahuje naštýlované elementy a animácie podľa android mobilných aplikácií. Použil som ho na skrášlenie vzhľadu aplikácie.

3.3 Trieda *OpcUaClientApi*

V tejto triede sa nachádza zhrnutá funkcionalita z OPC UA knižníc, prispôsobená mojej aplikácii. Na obrázku 3.4 môžete vidieť diagram tejto triedy.

Pri vytváraní instancie tejto triedy sa nakonfiguruje celá OPC aplikácia. Táto konfigurácia je pevne daná v kóde a užívateľ ju nemôže meniť. Nastavujú sa tam informácie o aplikácii, certifikát pre aplikáciu, transportné kvóty a logovanie chýb.

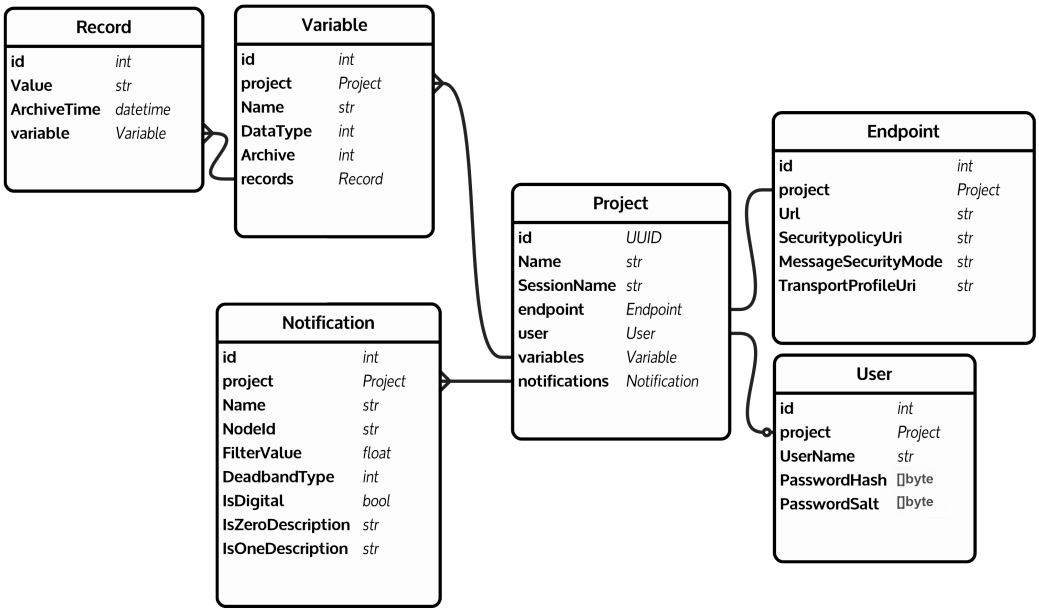
V tejto triede sa nachádzajú metódy pre vyhľadávanie serverov a koncových bodov, prehľadávanie adresového priestoru servera, vytváranie/zatváranie pripojenia, vytváranie *Subscription* a monitorovacích premenných, načítavanie/zapisovanie hodnôt, načítavanie informácií o uzle, registrovanie/odregistrovanie uzlov, pomocné metódy na získanie dátového typu z uzla a metódy pre overovanie certifikátov či znovupripojenie *Session* pri vypadnutí pripojenia.

Obr. 3.4: Diagram *OpcUaClientApi* triedy.

3.4 Ukladanie dát

Pre ukladanie dát som zvolil relačný databázový systém od Microsoftu *MS SQL Server*. Pri každom novom napojení na server sa do databázy ukladá projekt s koncovým bodom, na ktorý sa užívateľ pripojil a ak sa pripojil pomocou mena a hesla, tak sa ukladá aj užívateľ s týmito údajmi. Užívateľove heslo sa ukladá do databázy zahashované. Pre zvýšenie bezpečnosti sa pred hashovaním pridá k heslu *Salt*(náhodné vygenerovaný text), ktorý sa spolu so zahashovaným heslom ukladá

do databázy. Len hashovanie hesla je nepostačujúca ochrana pri slabých heslách. Použitím *Saltu* to isté heslo produkuje vždy iný hash, takže z databázy nevieme povedať či dvaja užívatelia majú to isté heslo a či má užívateľ to isté heslo aj v inom systéme. Ďalej zvyšuje obtiažnosť zistenia hesla pri porovnávaní hashov s dopredu vypočítanými hashmi. V databáze sú ďalej uložené premenné, ktoré archivujem, archivované hodnoty a typy notifikácii.



Obr. 3.5: Vzťahy medzi databázovými tabuľkami.

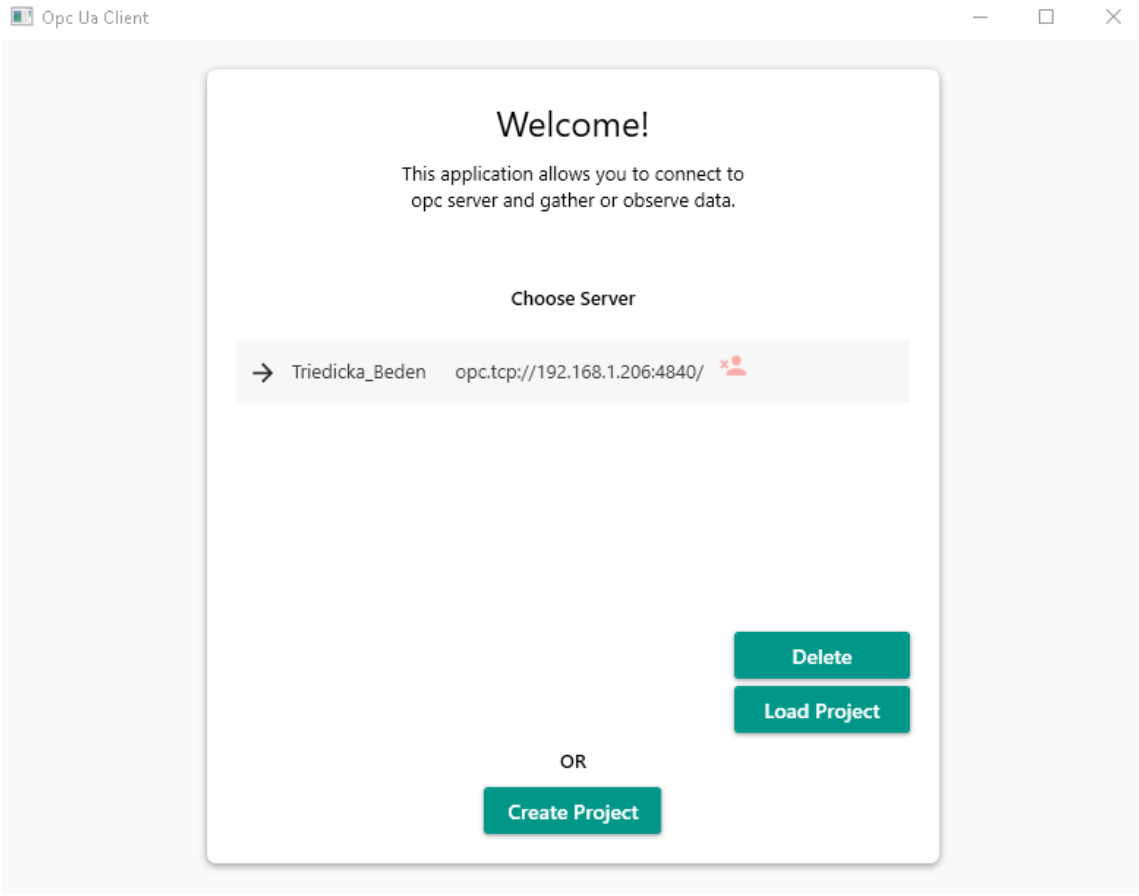
3.5 Funkcie aplikácie

Celá aplikácia sa skladá z troch obrazoviek.

- Úvodná obrazovka.
- Prihlasovacia obrazovka
- Hlavná obrazovka

3.5.1 Úvodná obrazovka

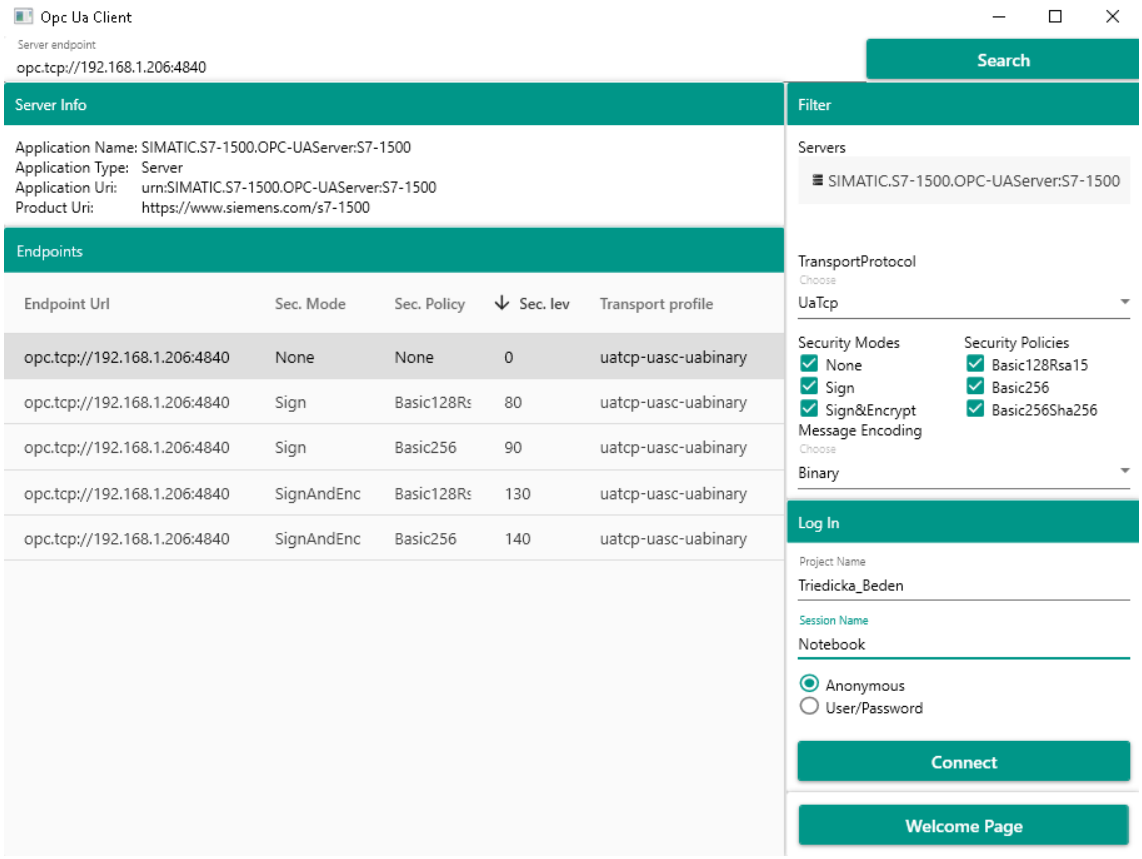
Táto obrazovka slúži na privítanie užívateľa. Nachádza sa tu list uložených projektov. Na konci každého projektu sa nachádza zelená alebo červená ikona užívateľa, podľa toho či daný projekt bol vytvorený pomocou mena a hesla. Ďalej tu má užívateľ na výber medzi vytvorením nového projektu alebo zmazaním a načítaním existujúceho projektu.



Obr. 3.6: Úvodná obrazovka.

3.5.2 Prihlasovacia obrazovka

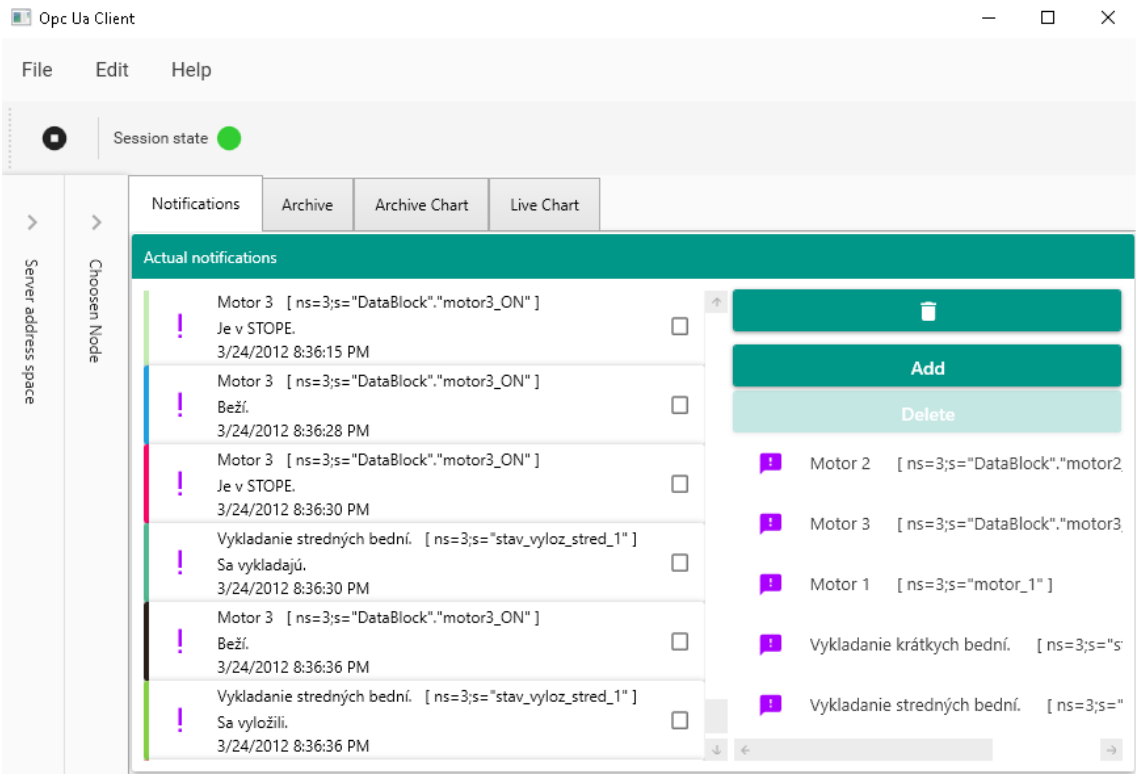
Na tejto obrazovke sa vyhľadávajú OPC servery a ich koncové body. V hornej časti obrazovky máme po ľavej strane textové pole, do ktorého sa zadáva adresa servera a po pravej strane tlačidlo, s ktorým spúšťame hľadanie. Pod tým sa nachádzajú informácie o nájdenom serveri a pod nimi sa nachádzajú nájdené koncové body, ktoré ponúka server. Na pravej strane sa nachádza filter, podľa ktorého si môžeme filtrovať koncové body, podľa nájdených serverov, transportného protokolu, zabezpečenia a šifrovania správ. Pod týmto filtrom sa nachádza prihlasovanie. Máme tu na výber medzi anonymným prihlasovaním a prihlasovaním pomocou mena a hesla. Taktiež si tu môžeme určiť názov projektu a názov *Session*. Úplne dole sa nachádza tlačidlo pre návrat na úvodnú obrazovku.



Obr. 3.7: Prihlasovacia obrazovka.

3.5.3 Hlavná obrazovka

Po úspešnom prihlásení alebo nahratí existujúceho projektu sa dostávame na hlavnú obrazovku. Táto obrazovka sa skladá z troch častí.



Obr. 3.8: Hlavná obrazovka.

Horná časť

V tejto časti sa nachádza menu.

V prvej záložke je na výber možnosť *Open*, kde sa dostaneme naspäť na úvodnú obrazovku, s možnosťou nahrať existujúci projekt z databázy. Ak vyberieme možnosť *New*, dostaneme sa na prihlasovaciu obrazovku, kde sa môžme pripojiť na nový OPC UA server. Tak isto tu máme možnosť ukončiť aplikáciu.

V ďalšej záložke sa nachádzajú funkcie kopírovať, vystrihnúť a prilepiť. A v poslednej záložke sa nachádzajú informácie o aplikácií, ako verzia či tlačidlo na otvorenie prehliadača s odkazom na projekt verzovaný na *GitHube*.

Pod menu sa nachádza panel nástrojov, kde môžeme ukončiť komunikáciu a prejsť na úvodnú obrazovku. Vedľa tohto tlačidla sa nachádza indikátor stavu pripojenia s OPC serverom. V prípade prerušenej komunikácie svieti kontrolka na červeno.

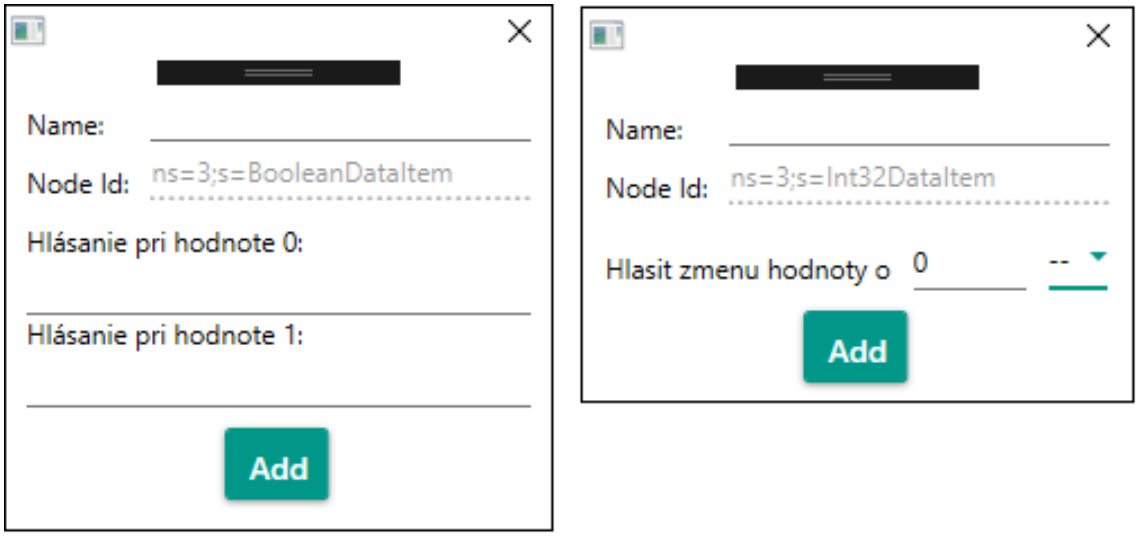
Ľavá časť

V ľavej časti sa nachádzajú dve rozťahovacie okná, kde v prvom sa nachádza adresový priestor servera a v druhom informácie o vybranom uzle z tohto adresového priestoru. Adresový priestor sa zobrazuje hierarchicky. Každý typ uzla je zobrazený inou ikonkou. V informáciách o uzle sa nachádzajú tlačidlá, ktoré sa nachádzajú v dolnej časti a zobrazujú sa len, ak máme vybraný uzol typu *Variable*. Pomocou týchto tlačidiel vieme načítať aktuálnu hodnotu alebo ju prepísať novou.

Pravá časť

Na pravej strane sa nachádzajú štyri záložky.

1 V prvej záložka s názvom *Notifications* si môžeme pomocou tlačidiel *Add* a *Delete* umiestnených na pravej strane pridávať/odoberať premenné, o ktorých chceme dostávať notifikácie. Nad nimi sa nachádza tlačidlo na zmazanie všetkých zobrazených notifikácií. Po stlačení tlačidla *Add* sa zobrazí buď okno na pridanie notifikácie, ktorá nadobúda dve hodnoty (true/false) alebo notifikácie, ktorá bude hlásiť zmenu hodnoty o nejaké percento, absolútnu hodnotu alebo každú zmenu hodnoty. Tieto okná sa zobrazujú podľa toho, akého dátového typu máme práve vybranú premennú v adresovom priestore. Táto záložka sa nachádza na obrázku 3.8.

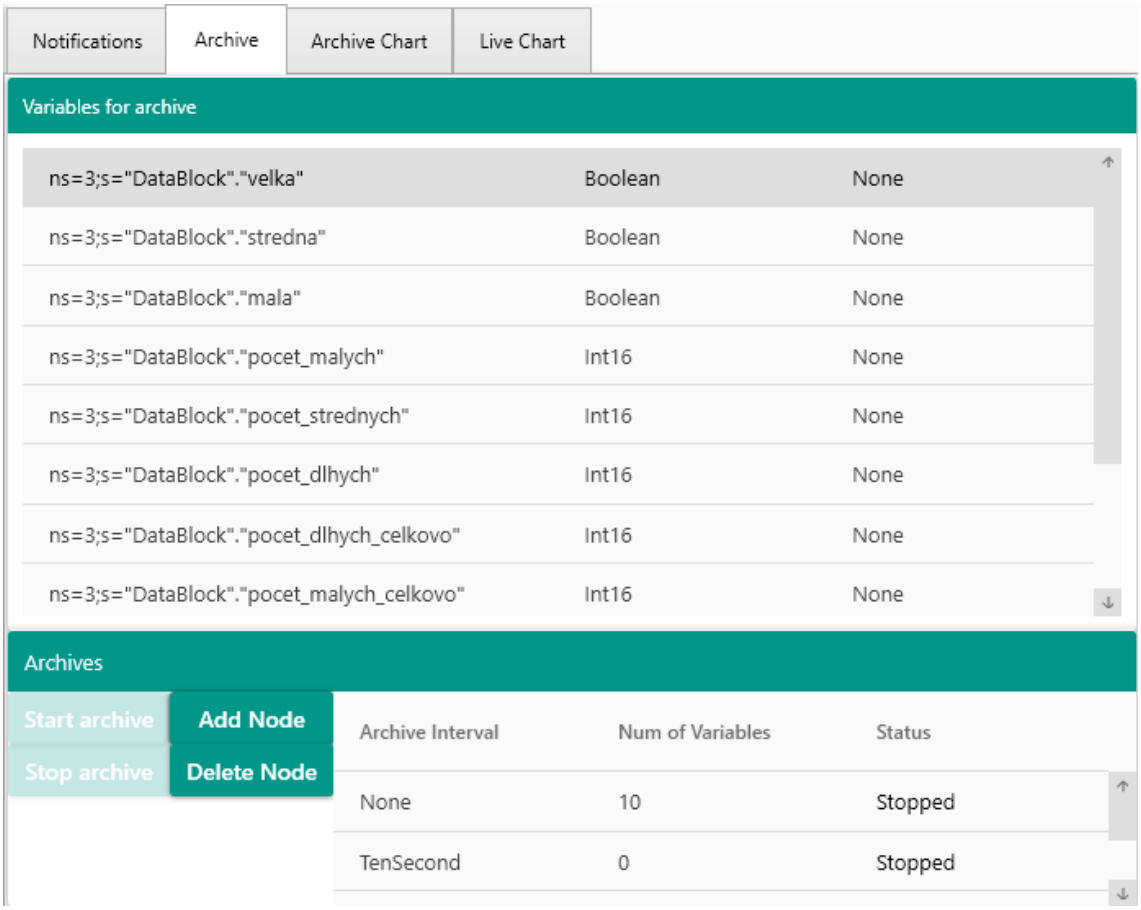


Obr. 3.9: Vyskakovacie okná pre pridanie notifikácie.

2 V ďalšej záložke s názvom *Archive* sa nachádza ovládanie archivácie. V strede okna sa nachádza list so všetkými premennými, ktoré sa majú archivovať. Tento list obsahuje id uzlu, dátový typ tohto uzlu a typ archívu. V aplikácii sú definované štyri druhy archívov a to:

- Typ 0 - dostáva notifikácie o zmenách hodnôt a tie archivuje.
- Typ 10, 20, 30 je archivácia v sekundách. Pre načítavanie hodnôt je použité registrované čítanie.

V ľavej dolnej časti sa nachádzajú tlačidla, ktoré slúžia na pridávanie/odoberanie premenných z archívu a spúšťanie/zastavovanie archívu. Vedľa tejto časti sa nachádza list, kde sú zobrazené typy archívov, počet premenných v každom archíve a status či beží, alebo nebeží archivácia.



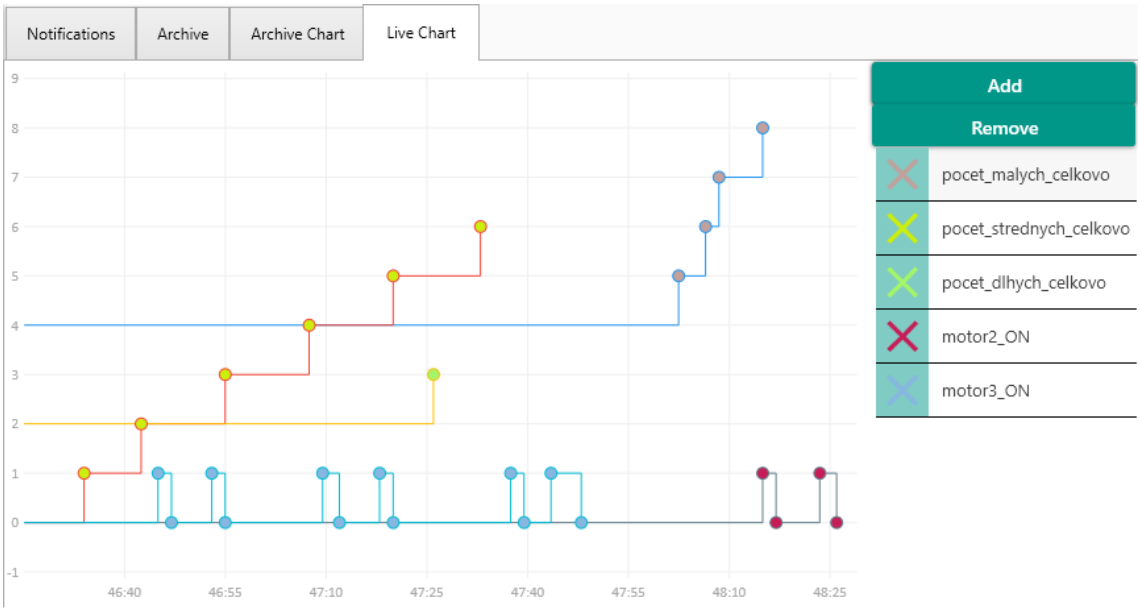
Obr. 3.10: Záložka pre ovládanie archívov a pridávanie/odoberanie premenných pre archivovanie.

Ak chceme pridať nejaký uzol do archívu, musíme si v adresovom priestore vybrať uzol typu *Variable* a v liste archívov vybrať archív, do ktorého chceme pridať tento uzol. Pokiaľ nie sú správne vybrané položky pre danú akciu, tlačidla sú zablokované. Podobné pravidlá platia aj pre ostatné tlačidlá v celej aplikácii.

3 V predposlednej záložke s názvom *Archive Chart* si môžeme zobraziť archivované hodnoty v grafe z listu, ktorý sa nachádza na pravej strane. V tomto liste sú zobrazené všetky archivované premenné a jedna položka v liste sa skladá z názvu id uzla a typu archívu pre danú premennú. Dáta sa zobrazujú kliknutím na vybranú položku v liste. Dá sa zvoliť viac položiek naraz. Dáta z grafu odstránime opätovným klikom po položke. Nad týmto listom sa nachádza tlačidlo na zmenu módu približovania a resetovaním zobrazenia grafu na východziu hodnotu. V grafe sa dá približovať v osiach x, y, xy a taktiež posúvať do všetkých strán kliknutím a potiahnutím myšou. Po prejdení myšou nad bodom v grafe sa nám zobrazí hodnota a čas, kedy bola táto hodnota archivovaná.

Obr. 3.11: Zobrazovanie grafu pre archivované hodnoty.

4 V poslednej záložke s názvom *Live Chart* sa nachádza graf, ktorý slúži na aktuálne sledovanie zmeny hodnôt. V pravej časti sa nachádzajú tlačidlá pre pridávanie či odoberanie uzlov na sledovanie. Pod tlačidlami sa nachádza list, ktorý sa skladá z krížiku, ktorý má farbu rovnakú ako príslušné body patriace tomuto uzlu. Vedľa tejto značky sa nachádza názov uzlu. List je obmedzený na päť uzlov a každému uzlu sa uchováva maximálne sto hodnôt, aby sme nepreťažili aplikáciu. X-ová os je nastavená tak, aby bola dve sekundy pred najnovšou zmenou hodnoty a dve minúty za ňou.



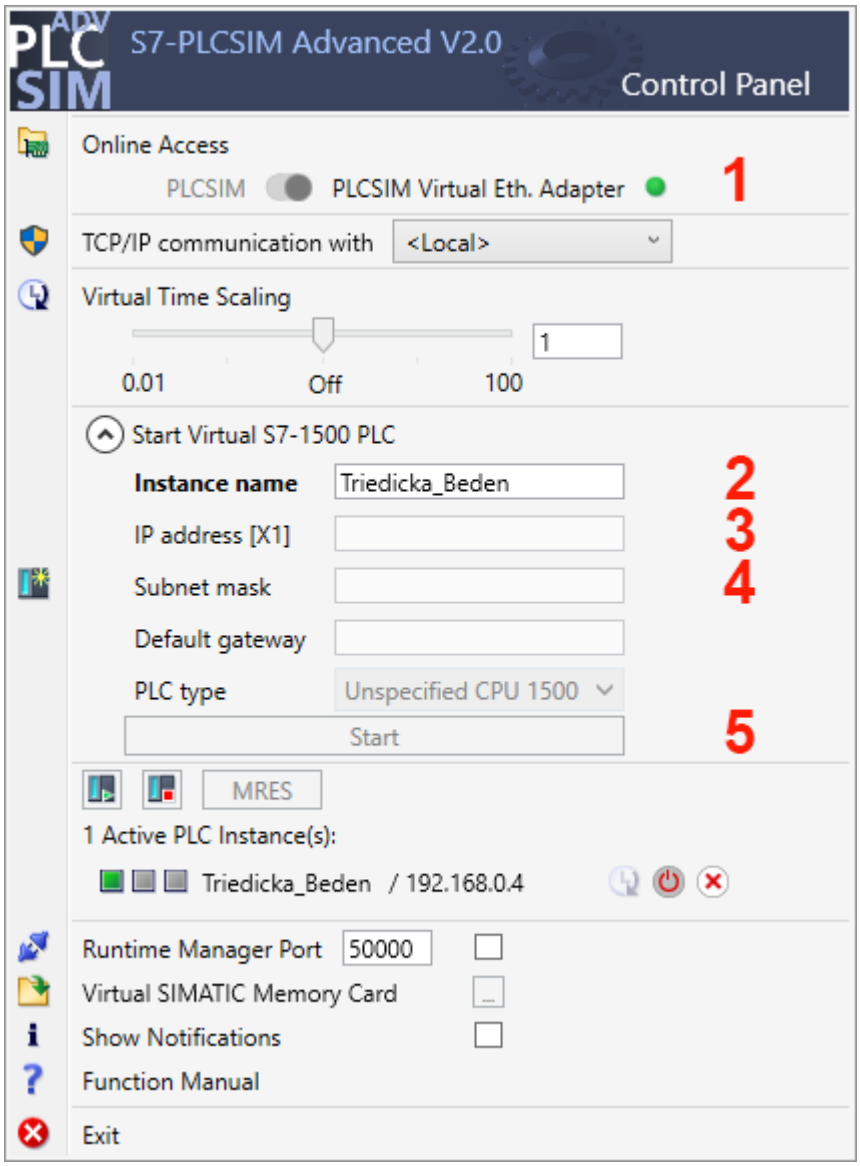
Obr. 3.12: Zobrazenie sledovania aktuálnych procesných hodnôt.

3.5.4 Testovanie OPC UA klient aplikácie

Komunikáciu klienta s OPC serverom na PLC som testoval pomocou *PLCSIM Advanced V2.0*, ktoré simulované PLC zobrazí v sieti. *PLCSIM* vytvorí na počítači ďalšiu sieť, ktorú nájdete v nastavení sieťového adaptéra. Tejto sieti nastavíme IP, ja som zvolil 192.168.0.15 .

Ďalej je potrebné vytvoriť instanciu PLC na *PLCSIM Advanced*.

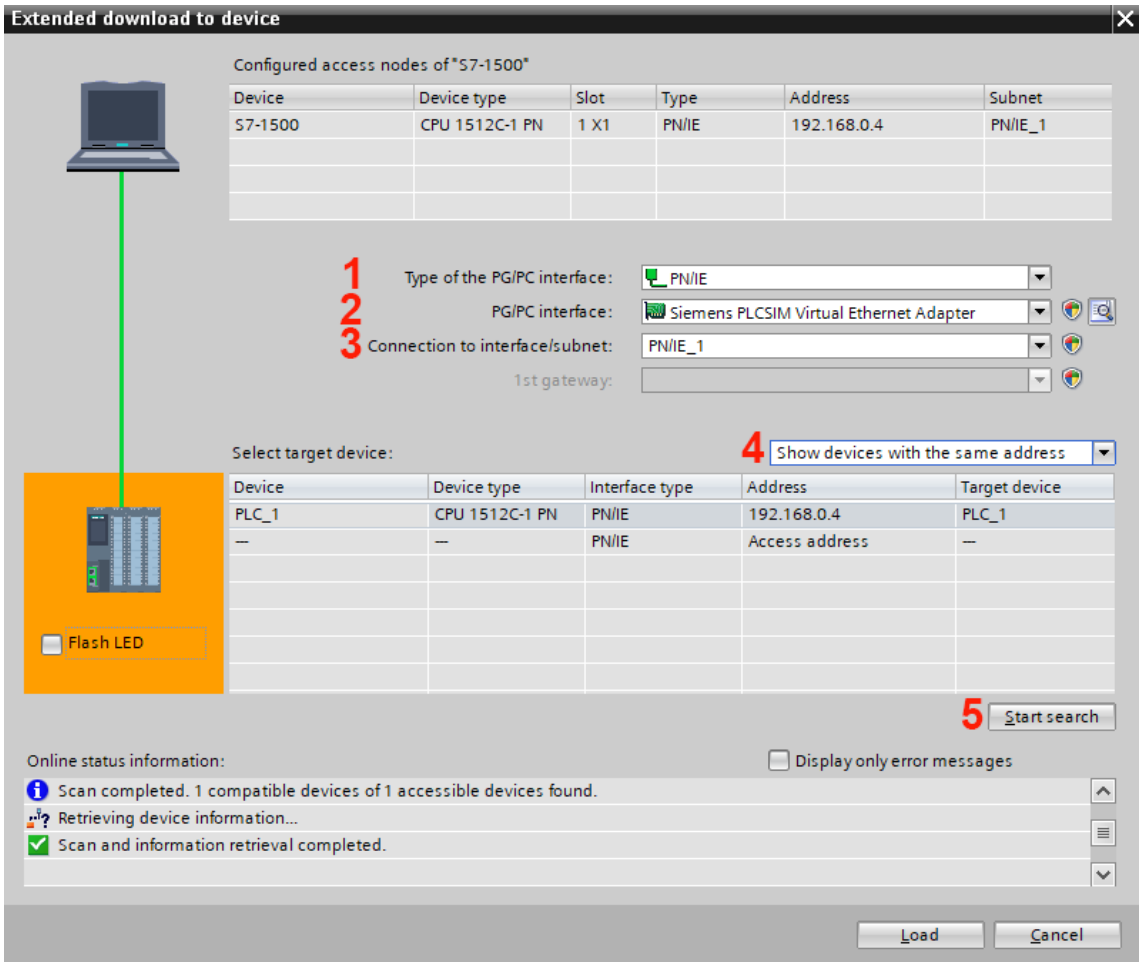
1. Prepnúť z *PLCSIM* na *PLCSIM Virtual Ethernet Adapter*.
2. Nastaviť meno instancie.
3. IP adresu zadať tak, aby bola v sieti spolu so simulovanou sieťou. V mojom prípade som zvolil 192.168.0.4 .
4. Masku podsiete nastaviť na 255.255.255.0 .
5. Kliknúť na tlačidlo štart.



Obr. 3.13: Nastavenie *PLCSIM Advanced*.

Ako posledné treba nahrať projekt do tejto instancie PLC. V TIA Portáli, si otvoríme nastavenia projektu a v záložke *protection* povolíme simuláciu dáta blokov. Ďalej nastavíme PLC IP ako na *PLCSIM* a dáme nahrávať projekt.

1. Typ PG/PC rozhrania nastavíme PN/IE.
2. PG/PC rozhranie nastavíme na *Siemens PLCSIM Virtual Ethernet Adapter*.
3. Pripájanie pomocou PN/IE_1.
4. Nastavíme zobrazenie zariadení len s IP adresou akú máme nastaveniach PLC.
5. Začneme hľadať.



Obr. 3.14: Nastavenie hľadania simulovaného PLC v sieti.

OPC klienta som tak isto testoval s testovacími OPC UA servermi, ktoré sú voľne dostupné na stránkach spoločností, ktoré ich vytvorili:

- *Prosys OPC UA Simulation Server* - napísaný v Jave.
- *UaCPPServer* - napísaný v C++
- *Softing OPC UA .Net Demo Server* - napísaný v C#

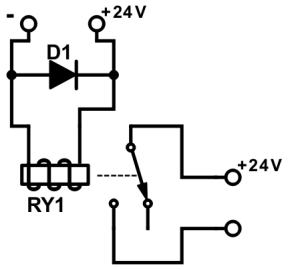
4 Laboratórny model *Třídička beden*

Pre testovanie aplikácie OPC UA klient som mal k dispozícii model *Třídička beden*. Predtým, ako som mohol začať programovať, potreboval som tento model trochu upraviť. Medzi hlavné dôvody úpravy patrí zmena typu PLC v laboratóriu a obmedzujúce možnosti modelu pre realizovanie zložitejšej úlohy.

4.1 Úprava a popis modelu

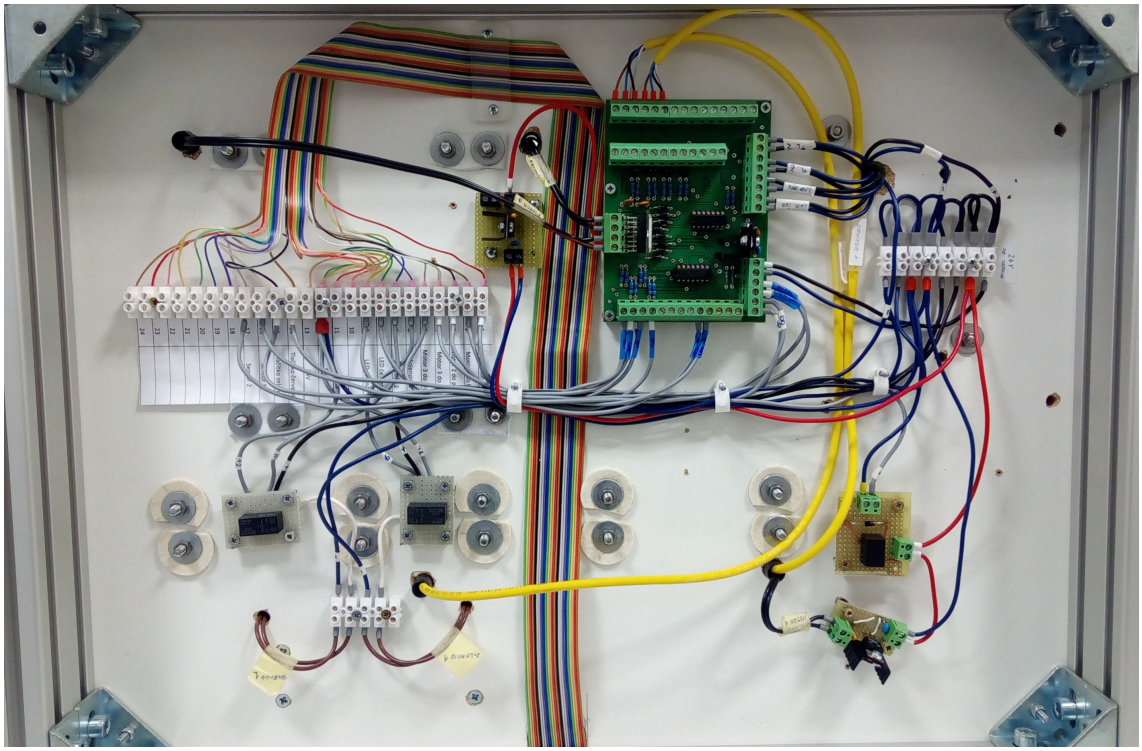
Na tomto modeli sa nachádzajú tri motory, ktoré sú riadené cez univerzálnu riadiacu dosku. Nanešťastie má táto doska len dva výstupy, z ktorých sa dajú napájať motory. Motory 2 a 3 boli pripojené z jedného výstupu cez 12V stabilizátor a Motor č.1 bol privedený z druhého výstupu tiež cez 12V stabilizátor. Tieto 12V stabilizátory sú tam použité, kvôli rýchlosti motorov, pri 24V sa pás pohyboval príliš rýchlo, čo znemožňovalo triedenie. Najväčšia nevýhoda bola, že všetky tri motory sa dali ovládať len jedným smerom.

Keďže po páse č.1 majú debne len prichádzať (t.j bude využitý len jeden smer), rozhodol som sa, že tento motor nemusí byť riadený cez univerzálnu dosku, ale bude riadený cez relé, s ktorým budem púšťať napätie do motora cez 12V stabilizátor. Použil som 24V relé na plošné dosky a umiestnil som ho na univerzálnu plošnú dosku, kde som pridal dve wago svorky, z toho jedna slúži na napájanie relé a druhá slúži ako výstupná, viď schému na obrázku č. 4.1. Z toho dôvodu mi ostali dva volné výstupy na univerzálnej doske, s ktorými ovládam motory 2 a 3 obojsmerne a hlavne samostatne. Kvôli tomu, aby som mohol tieto motory ovládať obojsmerne a napájanie bolo len 12V nemohol som výstup z univerzálnej riadiacej dosky priviezť na stabilizátor, lebo pri otočení smeru, by som otočil polaritu na vstupe stabilizátora. Kvôli tomu som potreboval na univerzálnej doske upraviť napájanie H mostíka z 24V na 12V. To som dosiahol tak, že som preškriabal cestičku ktorá napájala H mostík a priviedol som 12V zo stabilizátora.



Obr. 4.1: Schéma zapojenia relé.

Na privádzanie signálov z PLC je použitý 25 pinový samorezný konektor CANON 25, ktorý bol pôvodne dlhší, ale ja som ho skrátil a pripevnil na okraj modelu a vyrobil jeden meter dlhý proti kus s ktorým sa prepája PLC s modelom. Z tohto konektora bolo pôvodne použitých len 12 pinov, tak som pridal ďalšiu svorkovnicu a spravil predprípravu pre všetkých 25 pinov, viď obrázok č.4.2. V novo vybavenej učebni s PLC-1500 som si vybral 25 pinový vstupný konektor, kde bolo z PLC vyvedených 11 vstupov a 11 výstupov, jeden pin je 24V a dva zvyšné piny zem, viď tabuľku č. 4.1. Ďalej som na modeli upravil kabeláž, niektoré vodiče som skrátil a poprepájal. Urobil som nové označenia káblov a popísal som jednotlivé vstupy/výstupy z PLC.



Obr. 4.2: Upravený model *Třídička beden* zo spodnej strany.

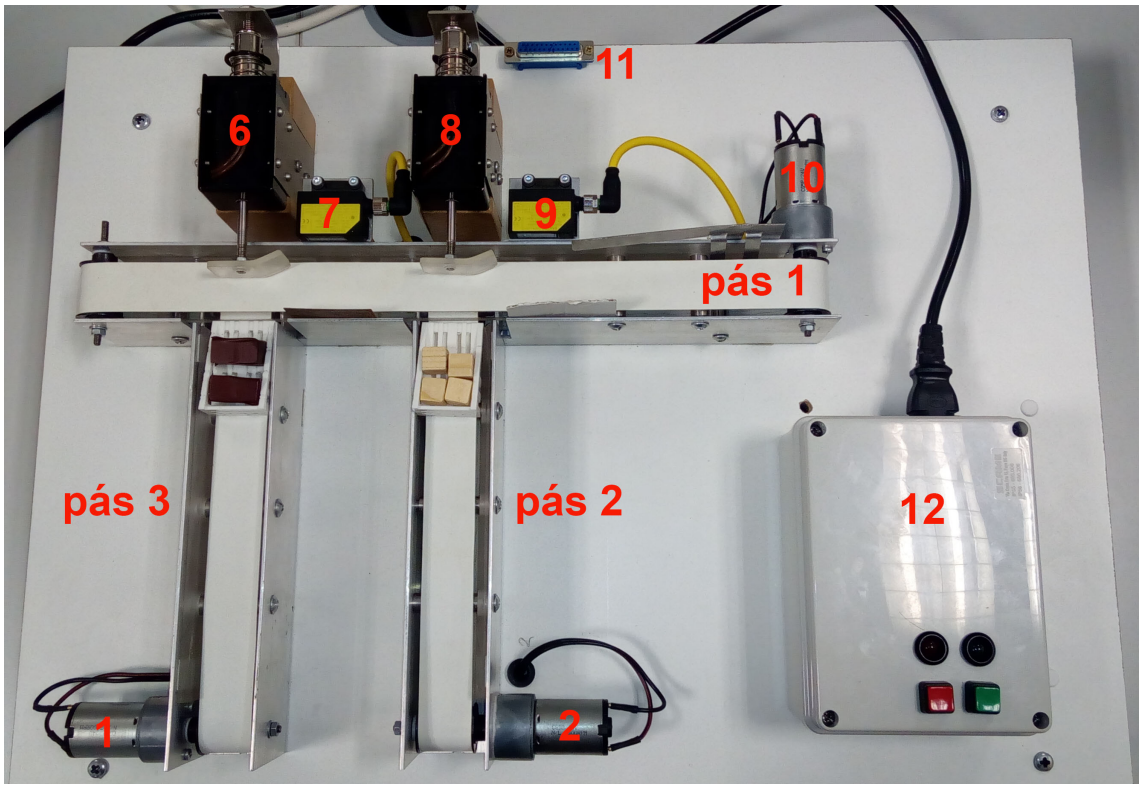
Nakoľko som potreboval debničky po niečom na páse prenášať, vytvoril som v programe Autodesk Inventor 3D model palety. Týmto paletám som pridal aj zábrany, aby debničky neprepadali za paletu. Následne sa tieto debničky vytlačili na 3D tlačiarni. Výsledný tvar palety, sa nachádza v prílohe D.1.

Tab. 4.1: Zobrazenie zapojenie svorkovnice a pinov na konektoroch CANON 25 spolu s PLC adresami.

	č. pinu svorky		č. pinu na modeli	č. pinu na PLC	č. svorky PLC karta	Adresa PLC
	1	Motor 1	15	24	38	Q7.7
	2	Motor 3 k pásu	3	11	37	Q7.6
V	3	Motor 3 od pásu	16	23	36	Q7.5
Ý	4	Motor 2 k pásu	4	10	35	Q7.4
S	5	Motor 2 od pásu	17	22	34	Q7.3
T	6	Solenoid 1	5	9	33	Q7.2
U	7	Solenoid 2	18	21	32	Q7.1
P	8	LED červená	6	8	31	Q7.0
Y	9	LED zelená	19	20	28	Q6.7
	10	-	7	7	27	Q6.6
	11	-	20	19	26	Q6.5
GND	12	GND	14	25	GND	-
	12	GND	1	13	GND	-
24V	13	24V	2	12	9	-
	14	Tlačítko červené	8	6	18	I13.7
	15	Tlačítko zelené	21	18	17	I13.6
V	16	Senzor 1	9	5	16	I13.5
S	17	Senzor 2	22	17	15	I13.4
T	18	-	10	4	14	I13.3
U	19	-	23	16	13	I13.2
P	20	-	11	3	12	I13.1
Y	21	-	24	15	11	I13.0
	22	-	12	2	8	I12.7
	23	-	25	14	7	I12.6
	24	-	13	1	6	I12.5

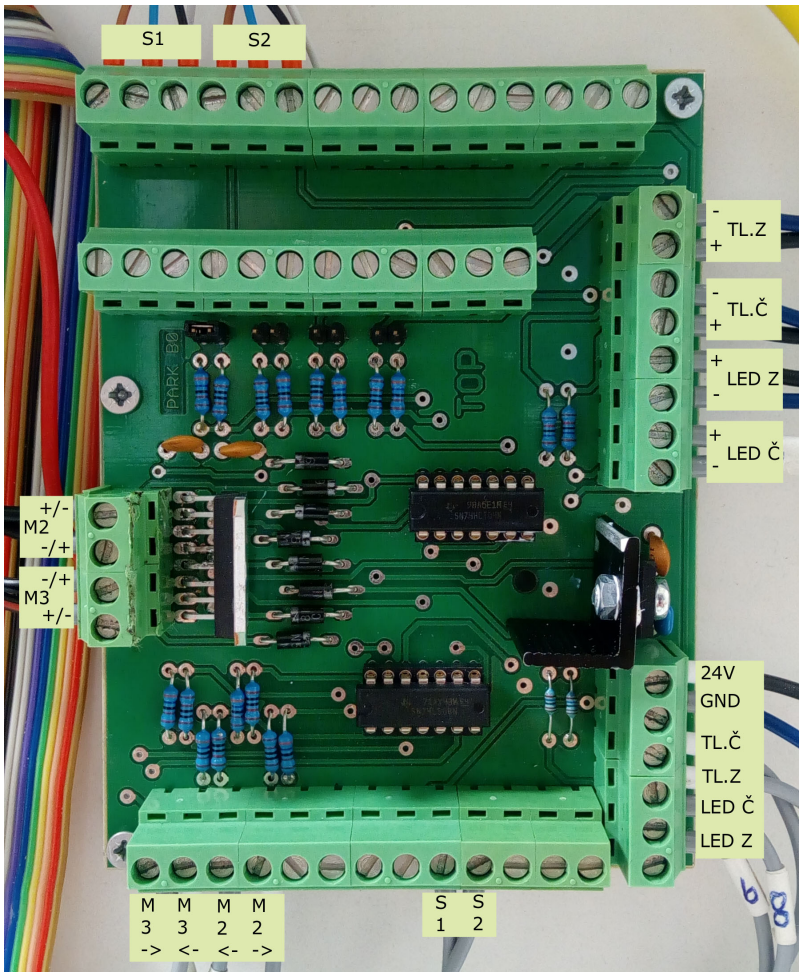
4.1.1 Popis upraveného modelu

Zobrazený model s očíslovanými prvkami môžeme vidieť na obrázku č. 4.3. Model sa skladá z troch dopravníkových pásov, z toho pás číslo 1 ovláda motor 1(č.10), pás číslo 2 ovláda motor 2(č.2) a pás číslo 3 ovláda motor 3(č.1). Pás číslo jedna sa dá ovládať len jedným smerom a pás číslo 2 a 3 sa dajú ovládať oboma smermi. Ďalej tu máme 2 solenoidy(č.6 a 8), ktorými debničky vysúvame na vedľajšie pásy. Pri solenoidoch sa nachádzajú optické závory(č.7 a 9), ktoré slúžia na detekciu dební. Ďalej tu máme panel(č.12, kde sa nachádzajú dve tlačítka a dve LED diódy. Z boku hlavného panela sa nachádza spínač, ktorým púšťame napätie do modelu. Model je napájaný zo siete napájacím sieťovým káblom 230V/50Hz. V paneli sa nachádza transformátor ktorý transformuje toto napätie na 24V. Vstupné a výstupné signály z modelu sú vyvedené 25 žilovým plošným káblom, na konci ktorého je 25 pinový samorezný konektor CANON(č.11), ktorý je pripevnený na kraji modelu.



Obr. 4.3: Model *Třídička beden*.

Zo spodnej strany modelu sa nachádza univerzálna riadiaca doska, do ktorej sú privádzané/odvádzané signály z/do PLC. Cez túto dosku PLC dostáva signály z optických závor, tlačidiel a posiela signály na zopnutie LED diód a ovládanie motorov 2 a 3 oboma smermi. Zobrazenú riadiacu dosku s popísanými vstupmi/výstupmi môžeme vidieť na obrázku 4.4. Doska je napájaná 24V z výstupu transformátora, ktorý je umiestnené v ovládacom paneli na vrchu modelu. Ďalej sa zo spodnej strany modelu nachádzajú 3 relé, ktoré sú ovládané signálmi prichádzajúcimi z PLC a zopínajú 2 solenoidy a hlavný motor (č.1). Ďalej tu máme 24 svoriek, do ktorých z jednej strany prichádza 25 žilový plošný kábel(zeme som prepojil, preto stačilo 24 svoriek, viď tabuľku č. 4.1) a z druhej strany sú vyvedené signály do jednotlivých prvkov. Zem z PLC a napájanie modelu sú prepojené. Všetky signály, ktoré vedú zo svorkovníc sú označené číslami, aby sa dalo rýchlo zistiť, kde čo vedie.



Obr. 4.4: Univerzálna riadiaca doska.

4.2 Zadanie úlohy:

Navrhnite riadenie modelu dopravníkového pásu dební, ktorý má za úlohu triediť debne podľa dĺžky. Tento model sa skladá z troch pásov, z toho po páse č. 1 sú posielané debne rôznych dĺžok a na pásoch č. 2 a 3 sú umiestnené palety, ktoré čakajú na naplnenie. Snímačom č.1 detekujte dĺžku dební a trieďte ich nasledujúcim spôsobom:

- krátke debničky trieďte na paletu, ktorá je na páse číslo dva
- stredne dlhé, trieďte na paletu, ktorá sa nachádza na páse číslo tri
- dlhé debne nechajte prejsť po páse až na koniec dopravníka
- na paletu pre krátke debne sa zmestia štyri debne
- na paletu pre dlhé debne sa zmestia 2 debne

Po naplnení niektorej z paliet je nutné paletu poslať na koniec pásu, kde trvá vyloženie 5 sekúnd. Po vyprázdnení pošlete paletu naspäť pri hlavný pás. Počas toho ako sa niektorá paleta vykladá, hlavný pás stále beží, až pokiaľ nepríde debna, ktorá nemá práve nachystanú paletu. Ak sa práve vykladá paleta s krátkymi debnami a snímač zosníma krátku debnu hlavný pás sa zastaví, ak zosníma stredne dlhú alebo dlhú beží ďalej. Pokiaľ sa stlačí červené tlačítko dvakrát po sebe do dvoch sekúnd, znamená to úplny stop, hlavný pás prestane bežať a palety na vedľajších pásoch prejdú na vyloženie. Pokiaľ sa toto tlačítko stlačí len raz, znamená to pauzu a vypne sa len hlavný pás. Pauzu môžete dať len v stave počas skenovania veľkostí dební. Zeleným tlačítkom sa preruší pauza. Počas triedenia je potrebné rátať celkový počet krátkych, stredných a dlhých dební. Na záver vytvorte vizualizáciu na príslušnej obrazovke TP700 Comfort.

4.2.1 Hardwarová konfigurácia PLC

Ako riadiacu jednotku používam PLC SIMATIC S7-1500, ktorá má v sebe zabudované 2 karty po 16 digitálnych vstupov/výstupov a jednu analógovú kartu, ktorá má 5 analógových vstupov a dva výstupy. Taktiež má v sebe zabudované rozhranie *PROFINET*, cez ktoré je PLC prepojené s dotykovou obrazovkou SIMATIC HMI TP700 Comfort. PLC obsahuje taktiež 6 vysoko rýchlostných čítačov a 4 pulzné generátory, ktoré však pri mojej úlohe nevyužívam. Ako je na obrázku č. 4.5 vidieť, digitálne vstupy sú namapované na bytoch 10-13 a digitálne výstupy na bytoch 4-7. Ďalej som si v nastaveniach PLC povolil systémové pamäťové byty a hodinové pamäťové bity, ktorých je 8 a namapoval som si ich na stý byte (*General->System and clock memory*). Tieto bity používam pri vizualizácii (popísané v sekcií 4.2.3).

Module	Rack	Slot	I address	Q address	Type	Article no.	Firmware
	0	100					
	0	0					
S7-1500	0	1			CPU 1512C-1 PN	6ES7 512-1CK00-0AB0	V2.1
AI 5/AQ 2_1	0	1 8	0...9	0...3	AI 5/AQ 2		
DI 16/DQ 16_1	0	1 9	10...11	4...5	DI 16/DQ 16		
DI 16/DQ 16_2	0	1 10	12...13	6...7	DI 16/DQ 16		
HSC_1	0	1 16	14...29	8...19	HSC		
HSC_2	0	1 17	30...45	20...31	HSC		
HSC_3	0	1 18	46...61	32...43	HSC		
HSC_4	0	1 19	62...77	44...55	HSC		
HSC_5	0	1 20	78...93	56...67	HSC		
HSC_6	0	1 21	94...109	68...79	HSC		
Pulse_1	0	1 32	110...113	80...91	PWM		
Pulse_2	0	1 33	114...117	92...103	PWM		
Pulse_3	0	1 34	118...121	104...115	PWM		
Pulse_4	0	1 35	122...125	116...127	PWM		
PROFINET interface_1	0	1 X1			PROFINET interface		
Port_1	0	1 X1 P1			Port		
Port_2	0	1 X1 P2			Port		

Obr. 4.5: Hardwarová konfigurácia PLC.

4.2.2 Riešenie

Pre riešenie tejto úlohy som si zvolil grafický jazyk *Ladder Diagram*, kvôli jednoduchosti. Ako prvé som si definoval v PLC tagoch vstupy a výstupy, ktoré mám privedené na PLC kartu cez 25 pinový konektor CANON. Tieto tagy som si rozdelil do tabuľky *digitálne_vstupy* a *digitálne_výstupy*. Ďalej som si spravil tabuľku *stavy*, kde mám definované všetky stavy, ktoré používam pri riešení úlohy. Do tabuliek som to rozdeľoval tak, aby som mal lepší prehľad o tom, kde sa čo nachádza a dokázal to rýchlejšie nájsť.

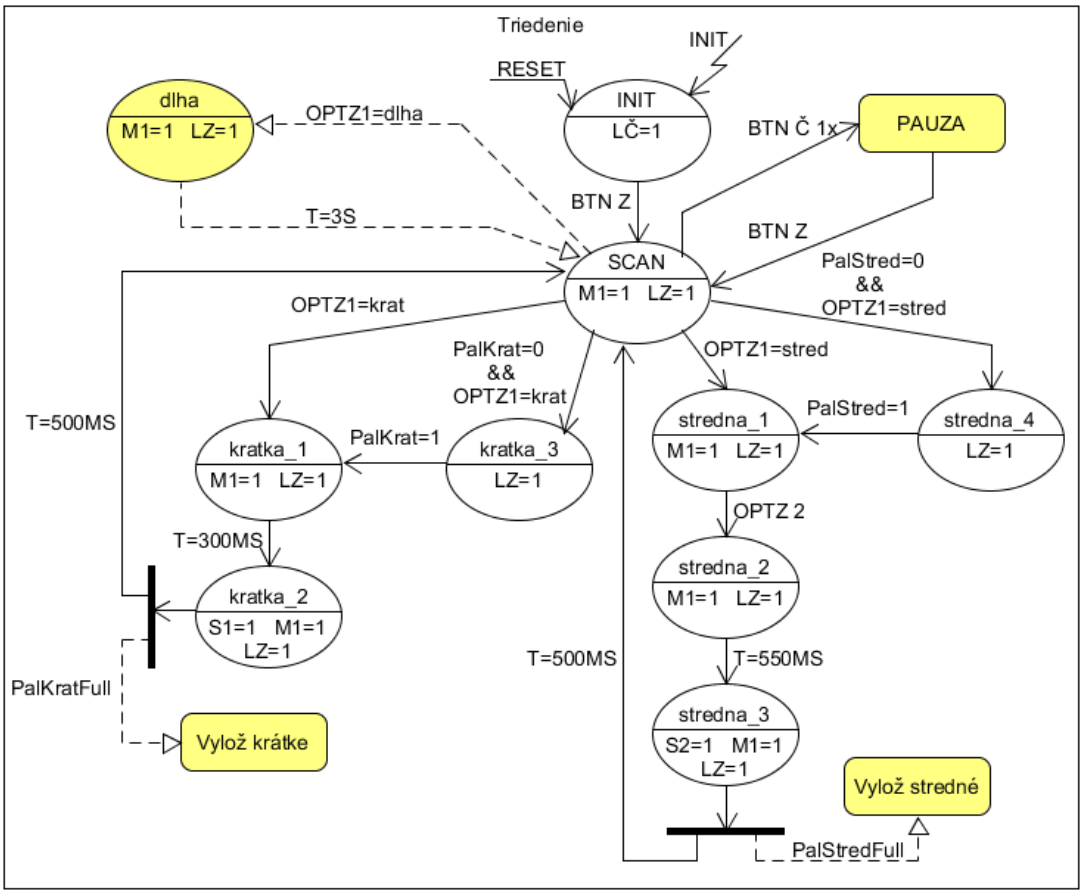
V projekte mám vytvorený jeden dáta blok, v ktorom mám definované všetky ostatné premenné, ktoré používam v programe či už pomocné, pre vizualizáciu alebo pre štatistiku. Všetky tieto dáta mám uložené v databloku preto aby som s nimi mohol komunikovať pomocou OPC klienta.

Úlohu som riešil pomocou stavového automatu. Celá úloha by sa dala popísať tromi stavovými automatmi a to jeden na triedenie, jeden na vyprázdňovanie paliet a posledný pre ovládanie červeného tlačítka. Tieto 3 stavové automaty bežia paralelne, nezávisle na sebe. Celú úlohu som rozčlenil do niekoľkých funkcií, ktoré volám z *mainu* (OB1), poprípade z inej funkcie.

4.2.3 Program

Funkcia *Stop/Pauza*:

V tejto funkcii riešim nastavenie pauzy a úplného stopu podľa toho či stlačil užívateľ červené tlačítko raz (pauza) alebo do 2 sekúnd ho stlačil aj druhý raz čo znamená úplný stop. Jednou nevýhoda je, že pauza sa dá dať len v stave skenovania dĺžky debne a to z toho dôvodu, že ak by sa nastavila pauza počas procesu triedenia v ktorom sa prechádza medzi stavmi po určitých časových intervaloch, ktoré sú presne nastavené a už tento čas nesúhlasil a debna by sa nedostala tam kam patrí. Tento problém by sa dal odstrániť umiestnením snímaču zvrchu nad solenoid, vtedy by sme presne vedeli kedy sa nachádza debna pri solenoide. V stave *Pauza* sa zastavia všetky pásy a v stave úplného stopu sa nastavia na nulu všetky pomocné premenné, aktuálny počet dební na palete, stavy a dostaneme sa do počiatočného stavu *INIT*.

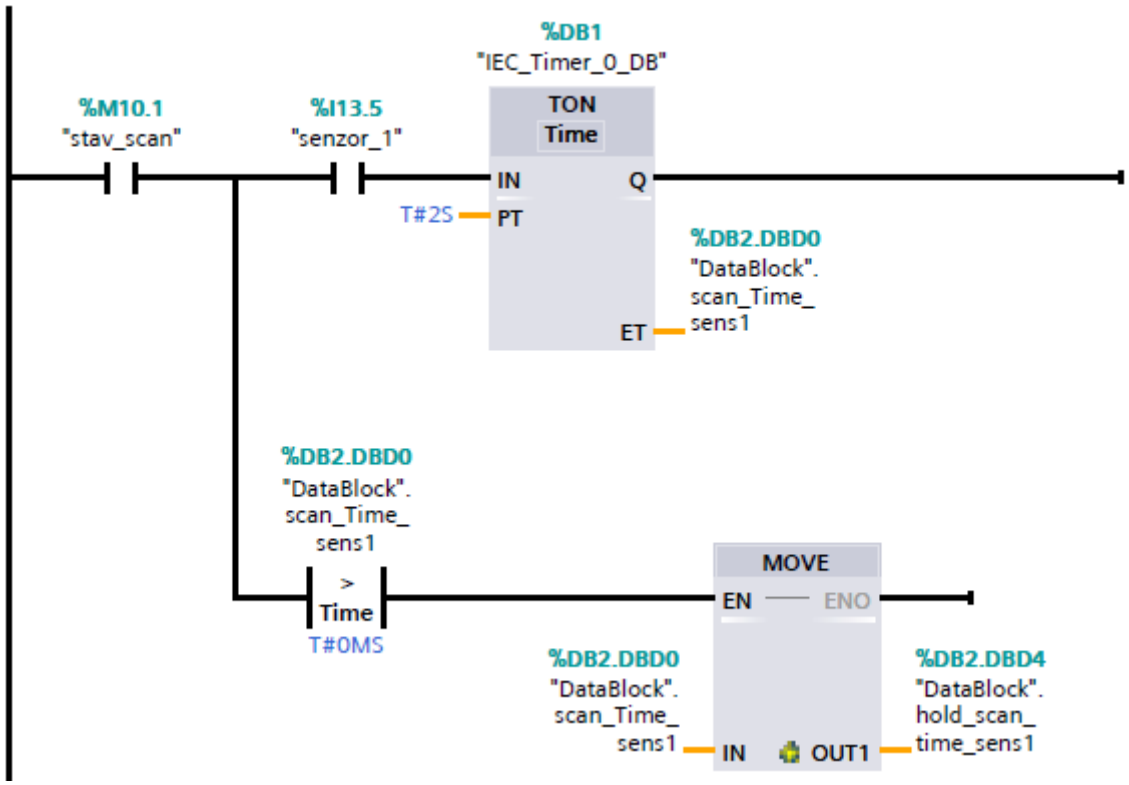


Obr. 4.6: Stavový automat pre riadenie modelu *Třídička beden*.

Funkcia *triedenie*:

V tejto funkcií prebieha algoritmus triedenia dební, ktorý zahŕňa inicializáciu, rozoznávanie rôznych dĺžok dební, spúšťanie solenoidov a rátanie aktuálneho počtu dební na palete. Z tejto funkcie ďalej volám funkcie na vyprázdňovanie paliet. Na začiatku funkcie prebieha inicializácia a nastavenie stavu *INIT*. Z toho stavu sa dostávame do stavu *SCAN* po stlačení zeleného tlačítka. Dĺžky dební skenujem senzorom 1.

Na obrázku 4.7 je zobrazený algoritmus zisťovania dĺžok dební na ktorom môžeme vidieť, že dĺžky dební zisťujem len v stave *SCAN*. Časovač je spustený len tak dlho, pokiaľ snímame snímačom 1 debničku. Na tomto časovači sme nastavili čas dlhší, ako trvá prechod najdlhšej debničky. Ďalšou podmienkou je,že ak je tento čas dlhší ako nula premiestňujem z aktuálneho času časovača čas do pomocnej premennej, v ktorej tejto čas udržujem. Toto presúvanie je potrebné z toho dôvodu, že v momente, keď stratí časovač napájanie, aktuálny čas časovača sa stratí a nemal by som čo porovnávať. Týmto spôsobom si meriam čas, aký bol snímač zopnutý a tento čas ďalej porovnávam a vyhodnocujem o akú dĺžku debne sa jedná a podľa toho nastavujem ďalšie stavy a inkrementujem počet danej debničky na palete. Vysvetlivky skratiek v stavovom automate sa nachádzajú v prílohe E.



Obr. 4.7: Algoritmus skenovania dĺžok dební.

Funkcia vyprázdňovania malých a stredných dební:

Funkcie vykladania dební spúšťam v momente, keď solenoidom zasuniem poslednú debnu tým, že nastavím pomocnú premennú na *true*. To, že viem o plnej palete mi zaistí čítač, ktorý inkrementujem po naskenovaní príslušnej debne. Tento čítač nastaví na výstupe pomocnú premennú, ak dočíta do čísla, ktoré je nastavené ako maximálny počet dební na palete. Čítač reštartujem na konci funkcie vyloženia dební. Triedenie prebieha naďalej pokiaľ nepríde debna, ktorá nemá svoju paletu na mieste.

Do tejto funkcie sa dostávame, ak nám čítač signalizuje, že je plná paleta. Na začiatku funkcie je umiestnený kontakt, ktorý reaguje len na nábežnú hranu. Tento kontakt tam je kvôli opätovnému spusteniu. Následne prebieha stavový automat, kde sa medzi stavmi prepínam časovačmi, keďže som nemal k dispozícii snímače. Týmito časovačmi simulujem odchod palety na koniec pásu, vyloženie a návrat palety. V dolnej časti tejto funkcie spúšťam potrebné výstupy podľa stavov. Ďalej po vyložení reštartujem čítač, ktorým rátam na príslušnej palete debne a po návrate signalizujem ukončenie vyloženia nastavením pomocného bitu. V programe sa nachádzajú dve instancie tejto funkcie, jedna pre vykladanie malých dební a druhá pre vykladanie stredných dební.

Funkcia *Vizualizácia*:

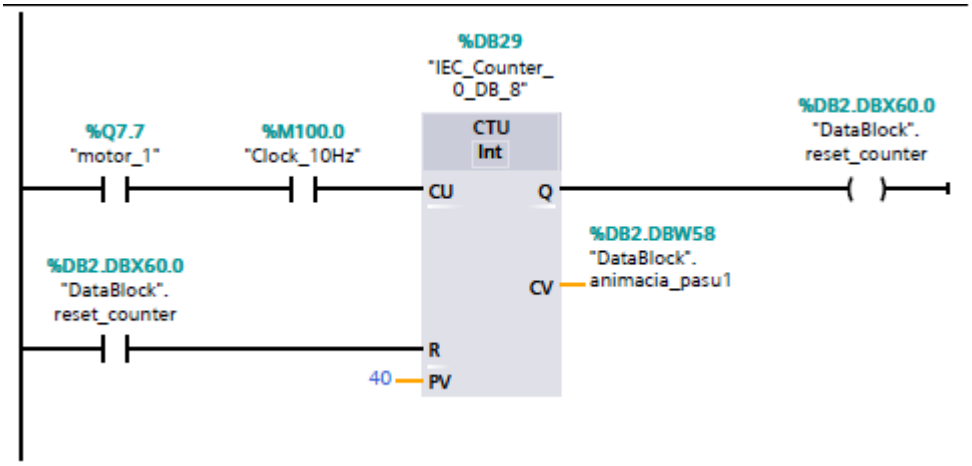
V tejto časti kódu si vyhodnocujem pomocné premenné pre vizualizáciu, keďže vo *WinCC* sa nedá dať do podmienok zobrazovania nejaká dlhšia logická podmienka. Ďalej som potreboval pre animáciu pohybu dební po páse zvyšovať nejakú celočíselnú hodnotu. Toto som vyriešil hodinovým pamäťovým bitom a čítačom. Keďže som vedel, koľko trvá prechod debne od skenovania po potrebné miesto a potreboval som trochu väčší rozsah hodnôt pre plynulú animáciu, použil som 10Hz hodinový bit, ktorý sa nachádza na nultom bite (u mňa M100.0).

Vychádzal som z nasledujúceho výpočtu:

$$T = \frac{1}{f} = \frac{1}{10} = 0.1s => MaxHodnota = \frac{\check{c}as \quad prechodu \quad debne[s]}{0,1s}$$

V tomto výpočte som zistil v akom rozmedzí sa mi bude meniť táto pomocná premenná. Takže za tento hodinový bit som dal čítač, ktorému som nastavil, aby rátal do tejto vypočítanej hodnoty a aktuálnu hodnotu čítača mám použitú v animácii. Tento čítač reštartujem nejakým nasledujúcim stavom.

Na obrázku č. 4.8 môžeme vidieť výpočet hodnoty pre animáciu pásu. Jediný rozdiel oproti animácii pohybu debničiek je, ten že čítač sa sám reštartuje, keď doráta.



Obr. 4.8: Algoritmus inkrementovania pomocnej premennej pre animáciu.

Funkcia *štatistika*:

Ako poslednú tu máme štatistiku, v ktorej prebieha počítanie celkového počtu všetkých dĺžok dební, ďalej sa tu budú rátať prevádzkové časy motorov, solenoidov, LED diód, počet stlačení tlačidiel, počet páuz a stopov a iných hodnôt. S týmito premennými budem ďalej komunikovať pomocou OPC klienta.

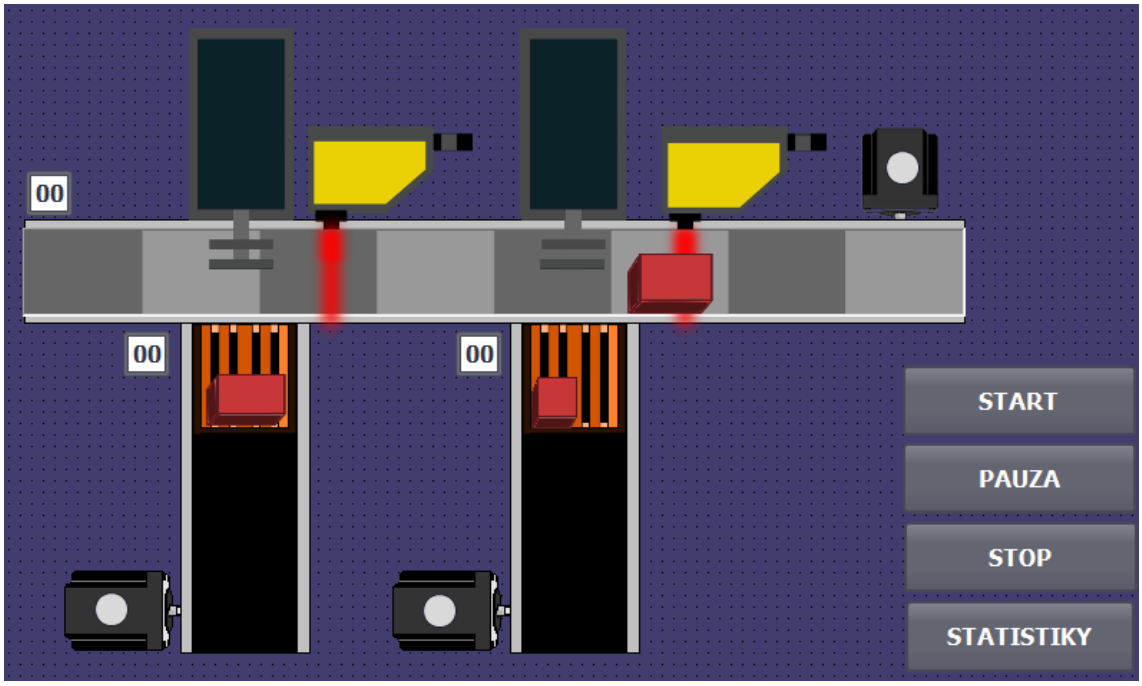
4.2.4 Vizualizácia

Vizualizácia je vytvorená vo *WinCC*, ktoré je súčasťou TIA Portálu. Z PLC programu som si prepojil s vizualizáciou potrebné premenné, s ktorými komunikujem každých 100ms kvôli plynulosti vizualizácie. Tieto premenné mám tak isto rozdelené vo viacerých tabuľkách kvôli lepšej prehľadnosti.

Vizualizáciu som spravil podľa vzhľadu modelu. Nachádza sa tu všetko, čo aj na modeli, ale namiesto ovládacieho panelu sú tu tlačidlá. Vo *WinCC* knižniciach som nenašiel postačujúce ikonky pre vizualizáciu a z toho dôvodu som si vytvoril vlastné vo vektorovom programe *Inkspace*. Zhotovil som si ikonky pre solenoidy, optické závory, palety a debničky, ktoré vyzerajú ako 3D. Taktiež som si zhotovil štyri obrázky pásov vždy o niečo posunutých, ktoré som použil pre animáciu pohybu hlavného pásu.

Na obrázku č. 4.9 môžeme vidieť, že chod motorov zobrazujem kruhom, ktorý je umiestnený na motore. Tento kruh sa rozsvieti na zeleno, ak je motor v chode. To akým smerom ide motor je znázornené na pásoch 2 a 3 pohybom palety a na páse 1 animáciou pohybu pásu a pohybu dební po páse. Taktiež si môžeme všimnúť, že z optických závor svieti lúč, ktorý sa preruší, ak je snímač zopnutý. To som

dosiahol jednoduchým zobrazením druhého obrázka, ktorý má prerušený lúč. Na ten istý princíp funguje solenoid, ktorý sa ako keby vysunie. Na okraji každého pásu zobrazujem počet aktuálnych dební na paletách. Na týchto paletách sa taktiež zobrazujú naložené debničky, ktoré po vyložení zmiznú.

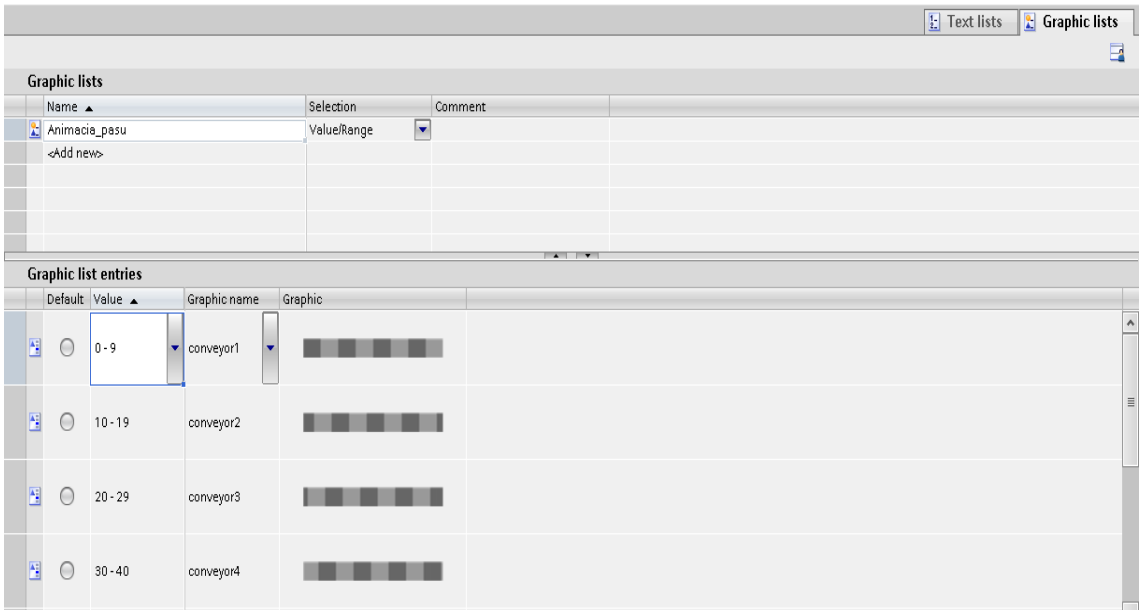


Obr. 4.9: Hlavná obrazovka vizualizácie.

Na pravej strane obrazovky sa nachádza ovládacie menu. Vo vizualizácii som funkciu červeného tlačidla rozdelil na dve samostatné tlačidla *Pauza* a *Stop*. Funkciu týchto tlačidiel sme si už vysvetlili vyššie. Tlačidlom štart spúšťame program a dostávame sa z pauzy, toto tlačidlo svieti na zeleno, ak beží triedenie a na červeno, ak sme v stave *INIT*, tj. program je v stope. Tlačidlo pauza svieti na červeno, ak sme v pauze. Posledným tlačidlom *Štatistika* sa dostávame na druhú obrazovku, kde sú zobrazené celkové počty zosnímaných dební. Tieto tlačidlá nie sú namapované na reálne vstupy tlačidiel, lebo stláčanie tlačidiel nefungovalo poriadne, z toho dôvodu som si vytvoril pomocné premenné, ktoré som dal paralelne k reálnym tlačidlám do programu.

Tvorba animácie

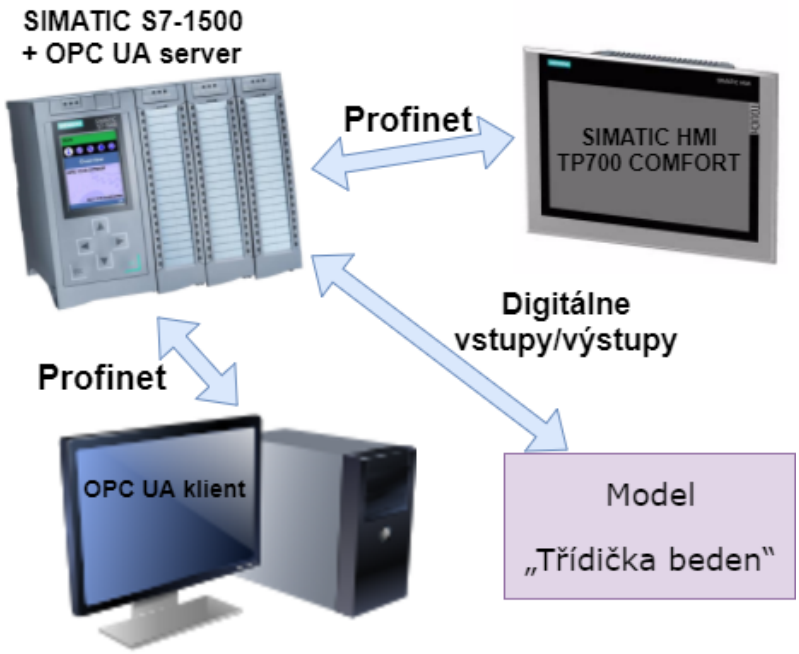
Animácia sa dá zhotoviť v *Text and graphic lists*, ktoré sa nachádza v menu pre vizualizáciu na ľavej strane, kde si zvolíme v pravom hornom rohu *Graphic lists*. V tomto okne si pridáme do *Graphic lists* názov animácie a v stĺpci *Selection* si vyberieme *Value/Range*, a to nám umožní meniť obrázky podľa hodnoty. Ďalej si v záložke *Graphic list entries* nastavíme obrázok v stĺpci *Graphic name*, ktorému nastavíme v akom rozsahu hodnôt sa má tento obrázok zobrazovať v stĺpci *Value*. To či sme si vybrali správny obrázok si môžeme overiť v stĺpci *Graphic*, kde je obrázok zobrazený. Tento postup môžeme vidieť na obrázku č. 4.10. Ak chceme túto animáciu použiť stačí ju premiestniť na obrazovku a nastaviť jej premennú podľa ktorej sa má animácia hýbať.



Obr. 4.10: Tvorba animácie.

5 Implementácia OPC UA aplikácie na model *Třídička beden*

Pre rozbehnutie komunikácie a zber dát je potrebné zapojiť zariadenia podľa nasledujúceho diagramu.



Obr. 5.1: Schéma zapojenia úlohy.

OPC UA klient nie je pevne viazaný len na laboratórny model *Třídička beden*. Dajú sa pomocou neho zbierať a pozorovať dáta z každého riadenia procesu, ktoré ovláda PLC s aktivovaným OPC UA serverom. Ak máme pripojené zariadenia podľa diagramu nahráme program pomocou TIA Portálu V14 do PLC a do dotykovej obrazovky. Po tomto kroku je model plne funkčný a ovládateľný z dotykovej obrazovky alebo z panelu, ktorý sa nachádza na modeli.

OPC UA klienta je potrebné nainštalovať. Inštalačný program sa nachádza v priloženom CD v záložke *OpcUaClient/Publish*. Pre nainštalovanie aplikácie je potrebné mať Windows 7 SP1 alebo Windows 8.1 alebo Windows 10 s .NET Frameworkom 4.6.2 . Ďalej musí byť v počítači nainštalovaný MS SQL server a prihlásený užívateľ musí mať práva k vytváraniu databáz na MS SQL serveri. Priečinky presunieme na plochu a spustíme inštaláciu s názvom *setup.exe*. Po inštalácii sa nám aplikácia sama spustí s uvítacím oknom. Klikneme na *Create Project* a otvorí sa nám prihlasovacia obrazovka. Do textového poľa s názvom *Server endpoint* zadáme adresu v tvare opc.tcp://<PLC IP>:4840 a vyhľadáme koncové body, ktoré PLC ponúka. Zvolíme

si koncový bod, zaklikneme anonymné prihlasovanie a pripojíme sa. Ďalšie funkcie aplikácie sú popísané v sekcii 3.5, spolu s obrázkami zo zbierania a pozorovania dát z modelu *Třídička beden*.

6 Záver

Cieľom tejto práce bolo zoznámiť sa s komunikačným protokolom OPC UA. Vytvoriť aplikáciu, ktorá bude pomocou tohto protokolu komunikovať s PLC Simatic S7-1500 a zbierať dáta z laboratórneho modelu *Třídička beden*. Taktiež som mal za úlohu upraviť tento model pre novovybavenú učebňu, tak aby sa dal model prepojiť s PLC pomocou 25 pinového konektoru CANON. Tiež som mal navrhnúť zložitejšiu úlohu pre triedenie ako bola pôvodná, aby sa mohol tento model využiť aj vo výuke a ďalej vytvoriť riadiaci programu a vizualizáciu pre navrhnutú úlohu.

Prácu som začal vysvetlením, čo je to OPC kde som popísal v krátkosti staršiu verziu OPC Classic a následne som popísal aj najnovšiu verziu OPC UA. Je to veľmi rozsiahli komunikačný protokol, ktorý sa dá využiť na komunikáciu medzi rôznymi systémami na všetkých leveloch riadenia podniku. Jeho najväčšou výhodou je, že je platformovo nezávislý a dá sa implementovať do rôznych zariadení ako snímače, mobilné telefóny, embedded systémy, kontroléry a taktiež v MES a ERP systémoch. Stále viac a viac výrobcov zariadení pridáva tento komunikačný protokol do svojich zariadení.

Následne som pokračoval vysvetlením nastavení OPC servera v TIA Portáli V14 a popísaním implementácie OPC servera, ktorý je implementovaný v PLC S7-1500.

V ďalšej kapitole som popísal architektúru OPC UA klient aplikácie a jej funkcie, taktiež ako som testoval túto aplikáciu a čo som pri testovaní používal. Celú aplikáciu som sa snažil vytvoriť, tak aby nebola viazaná len na model *Třídička beden*, ale dala sa použiť pre zber a pozorovanie dát z každého riadenia procesu. Podarilo sa mi naštudovať tento rozsiahly komunikačný protokol či už z knihy, alebo OPC UA knižníc, ktoré sú veľmi rozsiahle a na môj vkus sú napísané veľmi neprehľadne, a použiť ho v reálnej aplikácii. Aplikácia by sa dala rozšíriť o načítavanie polí hodnôt, štruktúr a pridať viac možnosti užívateľovi na nastavenie OPC UA klienta.

V nasledujúcej časti som sa zaoberal úpravou laboratórneho modelu a navrhovaním algoritmu triedenia cez stavový automat, ktorý je zobrazený na obrázku č.4.6. Na tomto obrázku sú čiarkovanými šípkami označené časti kódu, ktoré sa vykonávajú paralelne s algoritmom triedenia. Implementovaný algoritmus má nedostatok pri triedení viacerých dební naraz, t.j. ďalšia bedňa musí prísť k optickej závore, až keď sa bedňa pred ňou už zatriedila. Ďalší nedostatok je, že bedne musia prichádzať po páse kolmo na snímač a čo najbližšie kvôli dobrej reakcii. Ďalej som tu riešil snímanie dĺžok dební pomocou optickej závory, ktoré je popísané v sekcii 4.2.3 a zaznamenávanie stlačenia červeného tlačidla, ktorým som vyhodnocoval či ide o pauzu, alebo stop. Pre vizualizáciu som si niektoré ikonky vytvoril vlastné, kvôli nepostačujúcej knižnici, ktorá sa nachádza vo *WinCC*. Ďalej som vyriešil problém s animáciou pásu a animáciou pohybu debien po páse, ktorý je popísaný v sekcií

4.2.3.

V poslednej časti som popísal postup zapojenia PLC, dotykovej obrazovky, laboratórneho modelu a OPC UA klienta pre zber, upravovanie a pozorovanie procesných dát.

Literatúra

[1] STIANKO, Mgr. Martin a Ing. Jaromír PETERKA. OPC v průmyslové komunikaci. AUTOMA: časopis pro automatizační techniku [online]. 2004, 06/2004 [cit. 2018-05-07]. Dostupné z: http://automa.cz/cz/casopis-clanky/opc-v-prumyslove-komunikaci-2004_06_32378_2345/

[2] MAHNKE, Wolfgang, Stefan-Helmut LEITNER a Matthias DAMM. OPC Unified Architecture. Springer-Verlag Berlin Heidelberg, 2009. ISBN 978-3-540-68898-3.

[3] OPC UA .NET Client for the SIMATIC S7-1500 OPC UA Server: S7-1500 / OPC UA / .NET / C# [online]. 02/2018. Siemens, 2018 [cit. 2018-05-07]. Dostupné z: https://support.industry.siemens.com/cs/document/109737901/opc-ua-net-client-for-the-simatic-s7-1500-opc-ua-server?dti=0&lc=en-WW

[4] SIMATIC S7-1500, ET 200MP, ET 200SP, ET 200AL, ET 200pro Communication. SIMATIC S7-1500, ET 200MP, ET 200SP, ET 200AL, ET 200pro Communication: Function Manual [online]. 12/2017. Postfach 48 48, 90026 NÜRNBERG, GERMANY: Siemens AG, Division Digital Factory, 2017, s. 126-218 [cit. 2018-05-07]. Dostupné z: https://support.industry.siemens.com/cs/document/59193560/simatic-s7-1500-et-200sp-et-200pro-web-server?dti=0&lc=en-WW

[5] OPC Introduction. Unified Automation [online]. [cit. 2018-05-07]. Dostupné z: http://documentation.unified-automation.com/uasdkdotnet/2.5.4/html/L1OpcIntroduction.html

[6] OPC UA Information Models. Unified Automation [online]. [cit. 2018-05-07]. Dostupné z: http://documentation.unified-automation.com/uasdkdotnet/2.5.4/html/Doc_OpcUa_OpcUaInformationModels.html

[7] OPC UA Fundamentals. Unified Automation [online]. [cit. 2018-05-07]. Dostupné z: http://documentation.unified-automation.com/uasdkdotnet/2.5.4/html/L1OpcUaFundamentals.html

[8] OPC and OPC UA explained. Novotek [online]. [cit. 2018-05-07]. Dostupné z: https://www.novotek.com/en/solutions/kepware-communication-platform/opc-and-opc-ua-explained

[9] RINALDI, John. How OPC UA Clients Discover Servers (Part 1) [online]. December 02 2015 [cit. 2018-05-07]. Dostupné z: https://www.automation.com/automation-news/article/how-opc-ua-clients-discover-servers-part-1

[10] RINALDI, John. How OPC UA Clients Discover Servers (Part 2) [online]. December 15 2015 [cit. 2018-05-07]. Dostupné z: https://www.automation.com/automation-news/how-opc-ua-clients-discover-servers-part-2

[11] OPC UA Compact HMI Panels for Automation Applications. In: American Industrial Systems Inc. AIS: a better future [online]. 2016, 5/23/2016 [cit. 2018-05-07]. Dostupné z: http://www.aispro.com/news/opc-ua-compact-hmi-panels-for-automation-applications

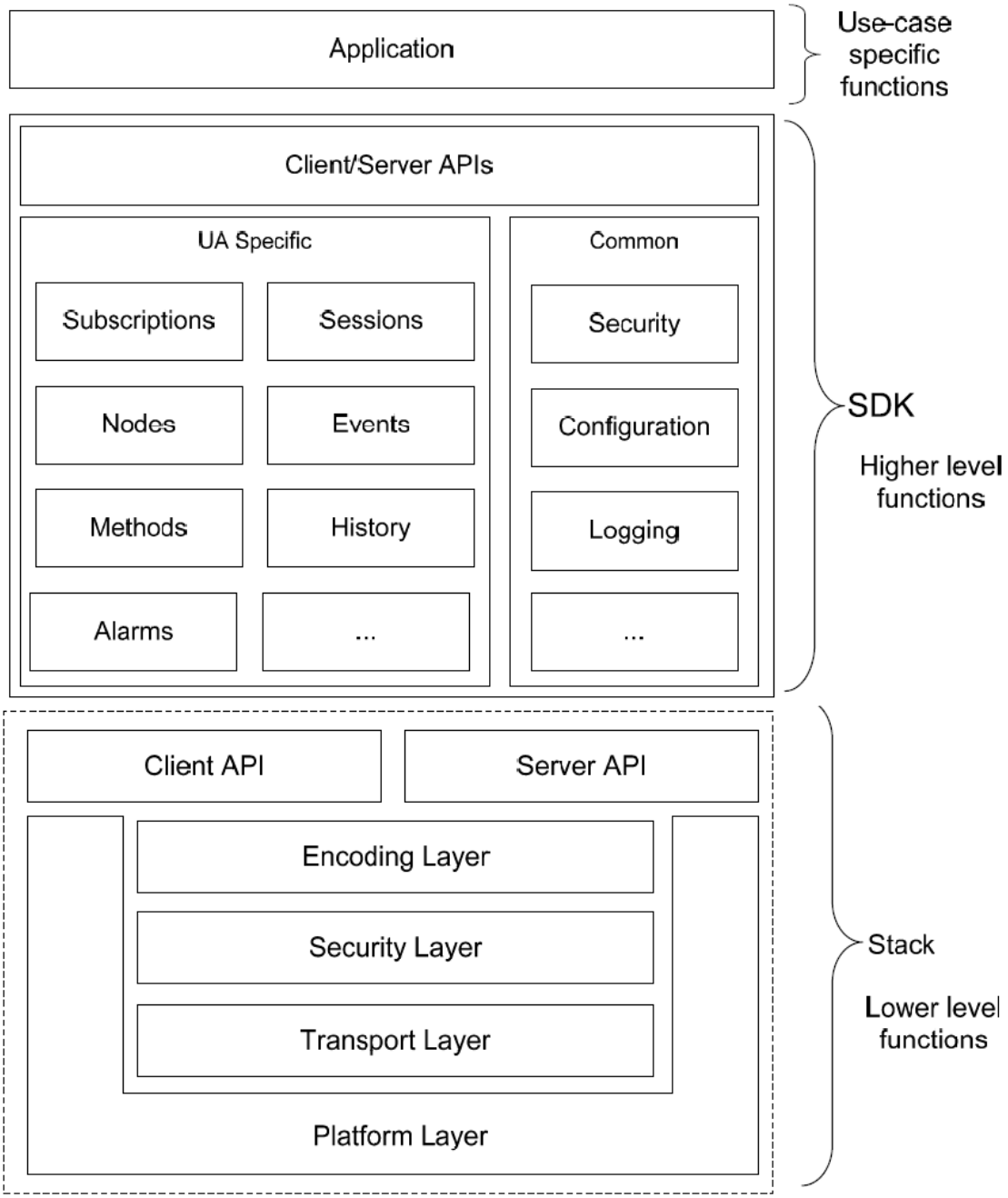
[12] The MVVM Pattern. In: MSDN Microsoft [online]. 10, 2012 [cit. 2018-05-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/hh848246.aspx

Zoznam symbolov, veličín a skratiek

OPC	OLE for control process
OLE	Object Linking and Embedding
COM	Component Object Model
DCOM	Distributed Component Object Model
HMI	Human Machine Interface
SCADA	Supervisory Control And Data Acquisition
PLC	programmable logic controller
DCS	Distributed control system
MES	Manufacturing execution systems
ERP	Enterprise resource planning
XML	Extensible Markup Language
SDK	Software Development Kit
UI	User Interface
XAML	Extensible Application Markup Language
WPF	Windows Presentation Foundation
MVVM	Model-View-ViewModel
ms	milisekunda
CPU	Central Processing Unit

Zoznam príloh

A Štruktúra OPC UA aplikácie



Obr. A.1: Softwarové vrstvy OPC UA aplikácie.[2]

B Príklad MVVM

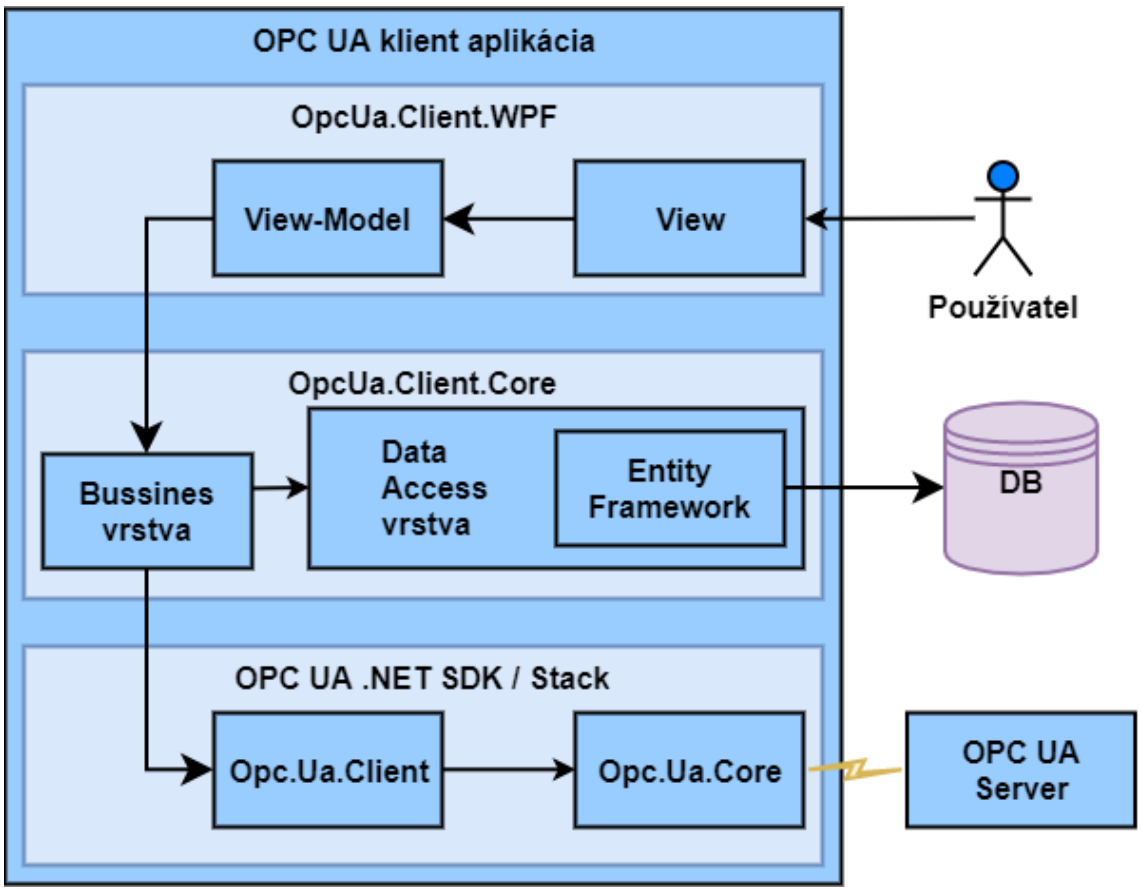
Výpis B.1: View-Model

```
1  public class MainWindowViewModel : INotifyPropertyChanged
2  {
3    private string _firstName;
4    public string FirstName
5    {
6      get => _firstName;
7      set
8      {
9        _firstName = value;
10       OnPropertyChanged();
11     }
12   }
13   public ICommand SaveUserCommand { get; }
14   public MainWindowViewModel() { SaveUserCommand = new
         RelayCommand(SaveUser, SaveUserCanUse); }
15   private void SaveUser() { /* Logic */ }
16   private bool SaveUserCanUse() => !string.IsNullOrEmpty(
         FirstName);
17
18   public event PropertyChangedEventHandler PropertyChanged;
19   protected virtual void OnPropertyChanged([CallerMemberName]
         string propertyName = null)
20   {
21     PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(
           propertyName));
22   }
23 }
```

Výpis B.2: View

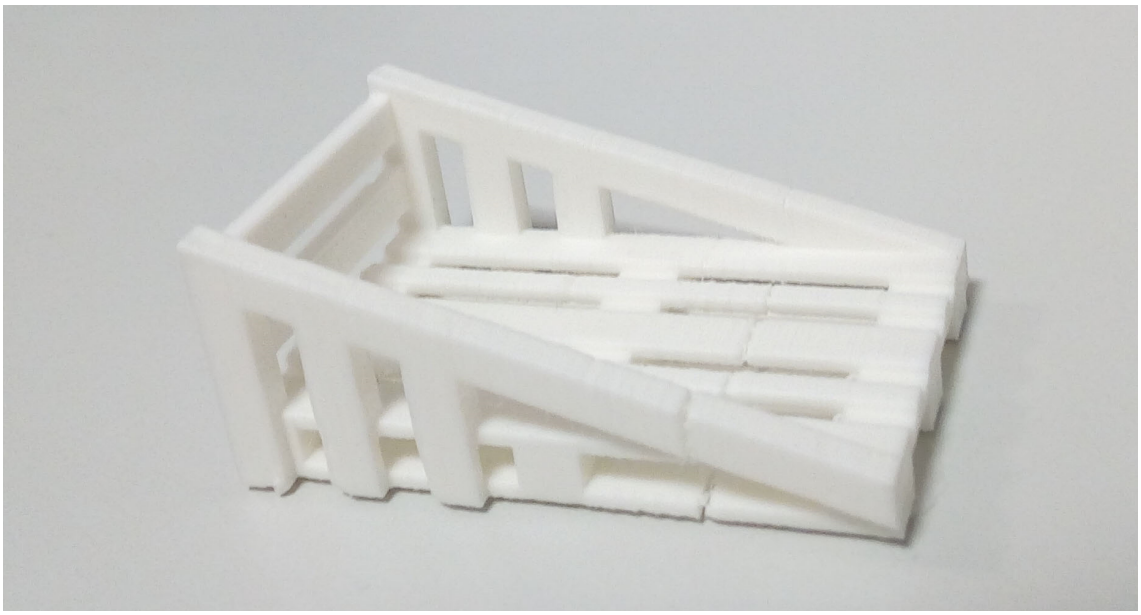
```
1  <Window ...
2          DataContext="{Binding MainWindowViewModel, Source={
               StaticResource ViewModelLocator}}">
3      <StackPanel>
4          <TextBox Text="{Binding FirstName, Mode=TwoWay,
       UpdateSourceTrigger=PropertyChanged}"/>
5          <TextBox Text="{Binding LastName, Mode=TwoWay,
               UpdateSourceTrigger=PropertyChanged}"/>
6          <Button Content="Save" Command="{Binding SaveUserCommand}
               "/>
7      </StackPanel>
8  </Window>
```

C Zjednodušený diagram aplikácie



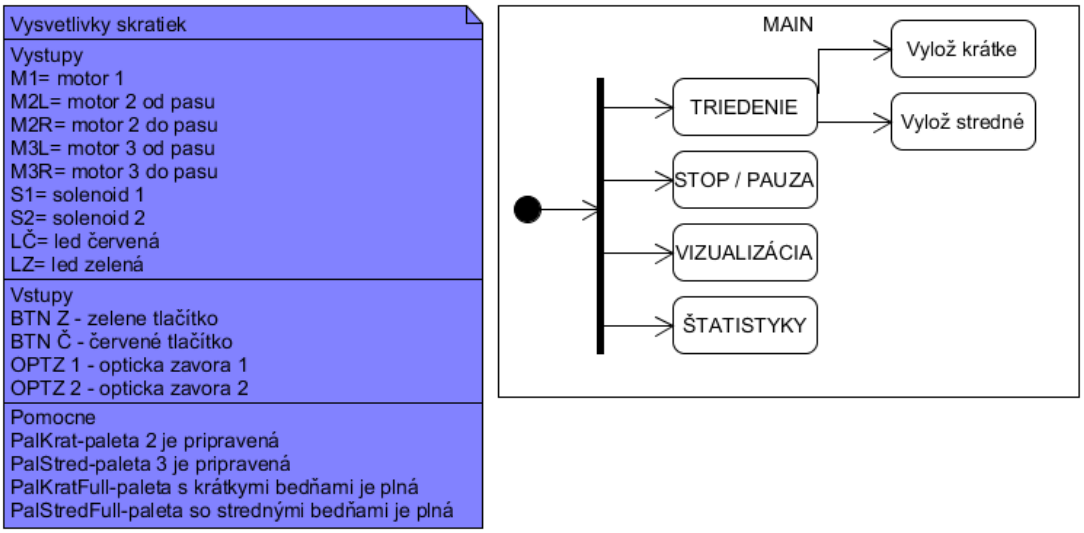
Obr. C.1: Zjednodušený diagram aplikácie.

D Paleta

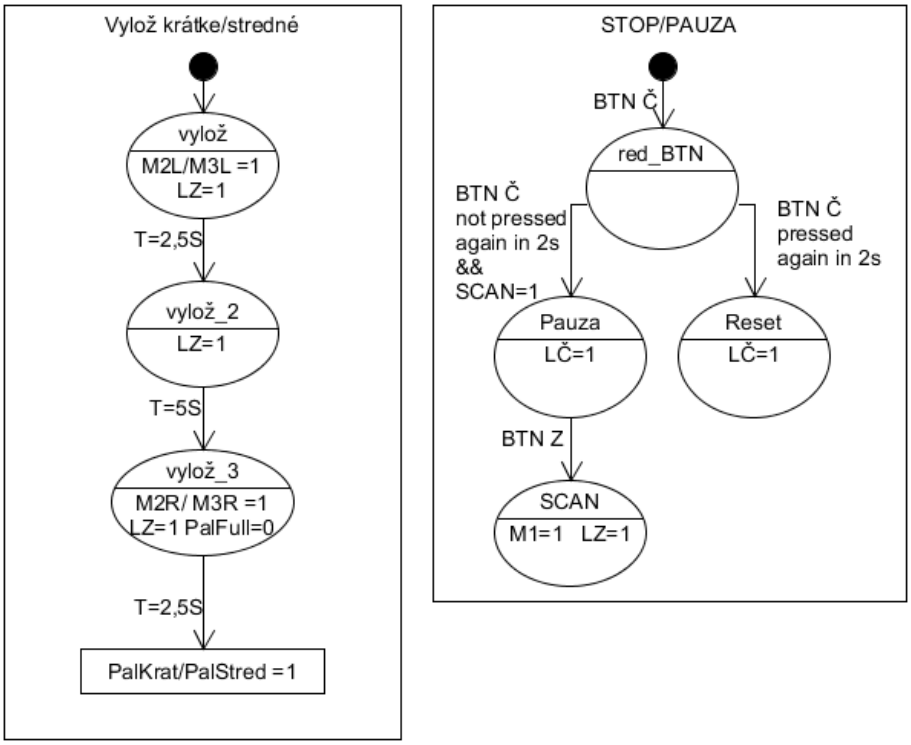


Obr. D.1: Výsledný tvar palety vytlačenej na 3D tlačiarni.

E Stavové automaty



Obr. E.1: Vysvetlivky skratiek v stavových automatoch a stavový automat *Main*.



Obr. E.2: Stavový automat vyloženia paliet a vyhodnotenie stlačeného červeného tlačidla.

F Obsah priloženého CD

- `OpcUaClient`
  - `Publish` ..... inštalačný súbor aplikácie
    - `Setup.exe`
    - `Application_Files`
    - `OpcUa.Client.WPF.application`
  - `OpcUa.Client.App` ..... C# projekt vytvorený vo Visual Studio 2017
- `Třídička-beden_TiaPortal_V14` ..... program pre dotykovú obrazovku a PLC
- `Model_EuroPalety` .... 3D model palety vymodelovaný v Autodesk Inventor 2017
  - `Europaleta.ipt`
  - `Europaleta.stl`
- `Bakalárska_práca-OPC_UA_klient_pro_laboratorní_model_Třídička_beden-Marek_Magáth.pdf` ..... elektronická verzia dokumentácie bakalárskej práce