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

SIMULACE VÝROBNÍ LINKY
SIMULATION OF PRODUCTION LINE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jakub Kutík**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jan Pásek, CSc.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

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

Student: Bc. Jakub Kutík ***ID:*** 198321
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Simulace výrobní linky

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je vytvoření simulátoru dopravníkové linky, který bude možno využít pro testování řídícího algoritmu PLC Siemens SIMATIC S7.

1. Proveďte literární rešerši používaných nástrojů pro simulaci výrobních strojů a linek.
2. Seznamte se s inženýrským rámcem TIA Portal a možnostmi simulačního nástroje PLCSIM Advanced od firmy Siemens.
3. Navrhněte a vytvořte program pro simulaci chování dopravníkové linky.
4. Vytvořte vizualizaci a uživatelské rozhraní simulačního programu s možností změny konfigurace linky a simulace různých chybových stavů.
5. Otestujte a demonstrujte vytvořený simulátor s vlastním řídícím algoritmem.

DOPORUČENÁ LITERATURA:

1. S7-PLCSIM Advanced, Function manual, Siemens, 12/2017 [online], ke stažení z:

https://support.industry.siemens.com/cs/document/109760835/simatic-s7-1500-s7-plcsim-adv-nced-?dti=0&pnid=24442&lc=en-WW

2. VITÁK, J. Simulátor výrobních linek. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 53 s. Vedoucí diplomové práce Ing. Jan Pásek, CSc.

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 13.5.2019

Vedoucí práce: Ing. Jan Pásek, CSc.
Konzultant: Ing. Petr Pokorný

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

Abstrakt

Cílem této diplomové práce je vytvoření vlastní aplikace pro simulaci výrobních linek. V první části práce jsou vysvětleny hlavní důvody, výhody a současné trendy v oblasti virtuálního zprovoznění a je popsáno několik používaných simulačních a inženýrských nástrojů od firmy Siemens. Druhá část se věnuje podrobnému popisu vlastností, možností a aplikačního rozhraní simulačního nástroje PLCSIM Advanced. Třetí část je věnována návrhu a realizaci vlastní simulační aplikace, ve které je možné vytvořit různé dopravníkové linky a ve spolupráci s PLCSIM Advanced otestovat uživatelský PLC program. Tento simulátor byl vytvořen pomocí herního enginu Unity3D a programovacího jazyku C#. V poslední části je pomocí vlastního simulátoru vytvořena jednoduchá simulační úloha, a je otestována pomocí PLC programu a vizualizace, vytvořenými v inženýrském rámci TIA Portal.

Klíčová slova

Virtuální zprovoznění, simulace, Průmysl 4.0, PLCSIM Advanced, TIA Portal, C#, Unity3D.

Abstract

The aim of this thesis is to create a custom application to simulate production lines. The first part explains the main reasons, benefits and current trends in the field of virtual commissioning and describes several simulation tools and engineering tools from Siemens. The second part is devoted to a detailed description of features, capabilities and application interface of the PLCSIM Advanced simulation tool. The third part is devoted to design and realization of the custom simulation application, which can be used for create conveyor lines and validate user PLC program in cooperation with PLCSIM Advanced tool. This simulator was created using Unity3D game engine and C# programming language. In the last part of the thesis, a simple simulation task is created in the simulator and is tested using PLC program and visualization created in the TIA Portal engineering framework.

Keywords

Virtual commissioning, simulation, Industry 4.0, PLCSIM Advanced, TIA Portal, C#, Unity3D.

Bibliografická citace:

KUTÍK, Jakub. Simulace výrobní linky. Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/119233. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Jan Pásek.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma *Simulace výrobní linky* jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **13. května 2019**

……
podpis autora

Poděkování

Děkuji vedoucímu mé diplomové práce Ing. Janu Páskovi, CSc. a konzultantovi Ing. Petru Pokornému za cenné odborné rady a připomínky ke zpracování této práce. Dále děkuji rodině a přátelům za podporu při tvorbě této diplomové práce.

V Brně dne: **13. května 2019** ……

podpis autora

Obsah

Úvod ... **13**

1 Virtuální zprovoznění ... **14**

1.1 Průmysl 4.0 ... 14

1.2 Digitální dvojče ... 15

1.3 Cíle a důvody digitalizace ... 16

1.4 Simulační nástroje ... 17

1.4.1 NX Mechatronics Concept Designer ... 18

1.4.2 TECNOMATIX ... 18

1.4.3 Simcenter Amesim ... 21

1.4.4 SIMIT ... 22

1.4.5 PLCSIM Advanced ... 22

1.5 TIA Portal ... 23

2 PLCSIM Advanced ... **25**

2.1 Vlastnosti ... 25

2.1.1 Porovnání s PLCSIM V15 (TIA Portal) ... 25

2.1.2 Control panel ... 26

2.1.3 Komunikace ... 26

2.1.4 Kompatibilita ... 27

2.1.5 Virtuální čas ... 27

2.1.6 Rozdíly mezi simulovaným a reálným PLC ... 27

2.2 Možnosti komunikace ... 28

2.2.1 Lokální komunikace ... 28

2.2.2 Distribuovaná komunikace ... 28

2.3 Nahrání programu do PLCSIM instance ... 30

2.3.1 Přímo do virtuálního PLC z TIA Portalu ... 30

2.3.2 Virtuální paměťová karta ... 31

2.4 Virtuální čas ... 32

2.4.1 Systémový čas virtuálního PLC ... 32

2.4.2 Časové měřítko a rychlost simulace ... 32

2.5 Synchronizace simulace ... 33

2.5.1 Pozastavení simulace PLC - stav „Freeze“ ... 33

2.5.2 Synchronizační bod ... 34

2.5.3 Operační módy ... 34

2.6 Aplikační rozhraní (API) ... 35

2.6.1 Runtime procesy a knihovny ... 36

2.6.2 Poskytovaná rozhraní ... 37

2.6.3 Přístup k PLC instancím ... 38

2.6.4 Rozhraní ISimulationRuntimeManager ... 38
2.6.5 Rozhraní IInstance ... 39
2.6.6 Rozhraní IRemoteRuntimeManager ... 42
3 Návrh simulátoru ... 43
3.1 Požadavky ... 43
3.2 Volba nástrojů ... 44
3.2.1 2D vs. 3D ... 44
3.2.2 Unity3D 2018.3 Personal ... 44
3.2.3 MS Visual Studio 2019 Community ... 44
3.2.4 C# ... 45
3.3 Úvod do Unity3D ... 45
3.3.1 Základní pojmy ... 45
3.3.2 Fyzikální engine ... 46
3.3.3 Skriptování ... 46
3.4 Realizace simulačního prostředí ... 48
3.4.1 Vzhled aplikace po spuštění ... 48
3.4.2 Přidání nového zařízení ... 49
3.4.3 Generování materiálu ... 54
3.4.4 Kamera ... 55
3.4.5 Save/Load systém ... 55
3.4.6 Komunikace s PLCSIM Advanced ... 56
3.4.7 PLCSIMInterface ... 58
3.4.8 Přidání PLC ... 65
3.4.9 Mapování signálů ... 67
4 Simulační úloha ... 69
4.1 Vytvoření dopravníkové linky ... 69
4.2 Vytvoření PLC programu ve STEP 7 (TIA Portal) ... 70
4.3 Vytvoření vizualizace ve WinCC (TIA Portal) ... 72
4.4 Otestování simulace ... 73
5 Závěr ... 75

Seznam symbolů a zkratek

Zkratky:

PLC	-	Programmable Logic Controller
HMI	-	Human-Machine Interface
HW	-	Hardware
SW	-	Software
I/O	-	Input/Output
CPU	-	Central Processing Unit
PLM	-	Product Lifecycle Management
CAD	-	Computer Aided Design
CAM	-	Computer Aided Manufacturing
CAE	-	Computer Aided Engineering
ERP	-	Enterprise Resource Planning
OPC	-	OLE for Process Control
HiL	-	Hardware in the Loop
SiL	-	Software in the Loop
IoT	-	Internet of Things
IoS	-	Internes of Services
CPS	-	Cyber-Physical System
API	-	Application Programming Interface
GUI	-	Graphical User Interface
TIA	-	Totally Integrated Automation
WinCC	-	Windows Control Center
WPF	-	Windows Presentation Foundation
TCP	-	Transmission Control Protocol
IP	-	Internet Protocol
DLL	-	Dynamic-link library
MMF	-	Memory-mapped file
LAD	-	Ladder Diagram
SRM	-	Simulation Runtime Manager

Seznam obrázků

Obr. 1-1: Vývoj průmyslu [6] 14
Obr. 1-2: Pravidlo deseti (převzato z prezentace firmy Siemens) 16
Obr. 1-3: Rozdělení simulačních nástrojů firmy Siemens [7] 17
Obr. 1-4: Rozdělení simulačních nástrojů firmy Siemens [7] 17
Obr. 1-5: Ukázka prostředí NX MCD [10] 18
Obr. 1-6: Ukázka prostředí Plant Simulation [10] 19
Obr. 1-7: Ukázka prostředí Process Simulate [10] 20
Obr. 1-8: Ukázka prostředí Simcenter Amesim [9] 21
Obr. 1-9: Ukázka prostředí SIMIT 22
Obr. 1-10: Koncept Totally Integrated Automation 24
Obr. 2-1: Ovládací panel PLCSIM Advanced 26
Obr. 2-2: Lokální komunikace 28
Obr. 2-3: Příklad distribuované komunikace PC <-> PC 29
Obr. 2-4: Příklad distribuované komunikace PC <-> VM 29
Obr. 2-5: Povolení PLCSIM Virtual Switch 30
Obr. 2-6: Záložka „Online access“ ve stromu projektu v TIA Portal 30
Obr. 2-7: Nastavení projektu v TIA Portal 31
Obr. 2-8: Přidání uživatelské čtečky karet 31
Obr. 2-9: Volba cílového zařízení v dialogovém okně „Load preview“ 32
Obr. 2-10: Časový diagram výskytu synchronizačního bodu 34
Obr. 2-11: Schéma runtime komponent PLCSIM Advanced 36
Obr. 2-12: Schéma přístupu externí aplikace k API 37
Obr. 2-13: Lokální a vzdálený přístup k PLC instancím 38
Obr. 3-1: Vzhled kosimulační aplikace po spuštění 48
Obr. 3-2: UI - horní panel 49
Obr. 3-3: UI – dolní panel 49
Obr. 3-4: Vzhled hlavního menu 49
Obr. 3-5: Okno pro přidání nového zařízení 49
Obr. 3-6: Panel válečkového dopravníku 50
Obr. 3-7: Vygenerovaný dopravník 52
Obr. 3-8: Ukázka přerušené a nepřerušené optické závory 54
Obr. 3-9: Schéma výměny dat mezi PLCSIMInterface a kosim aplikací 59
Obr. 3-10: Panel PLC komunikace 66
Obr. 3-11: Ovládací panel PLC 66
Obr. 3-12: Seznam signálů 67
Obr. 3-13: Ukázka propojeného a nepropojeného signálu 68
Obr. 4-1: Schéma komunikace nástrojů simulační úlohy 69

Obr. 4-2: Vytvořená dopravníková linka a její signály 69
Obr. 4-3: Programové bloky TIA projektu 70
Obr. 4-4: Část programu funkce FC1 v jazyu LAD 70
Obr. 4-5: Funkční blok FB1 pro ovládání pohonu dopravníku 71
Obr. 4-6: Část bloku FB1 pro start a reverzaci pohonu 72
Obr. 4-7: Obrazovka vizualizace pro ovládání linky 72
Obr. 4-8: Propojení signálů a PLC tagů 73
Obr. 4-9: Automatické řízení linky 74

Seznam tabulek

Tabulka 1: Porovnání funkcí PLCSIM produktů 25
Tabulka 2: Operační módy - cyklové řízení 35
Tabulka 3: Operační módy - časová synchronizace 35
Tabulka 4: Struktura MMF souboru pro ovládání a informace o PLC 61

ÚVOD

Průmyslovým světem nyní hýbe tzv. čtvrtá průmyslová revoluce a s ní související iniciativa Průmysl 4.0. Jejím spouštěčem je, mimo jiné, změna tržní situace, resp. zvyšující se rychlost této dlouhodobé změny. Zvyšuje se poptávka, požadavky zákazníků se mění stále rychleji, technologie dělá velké pokroky, výrobky jsou čím dál složitější atd. V důsledku toho produkty velmi rychle stárnou a jejich životní cyklus se zkracuje. Tím se zvyšuje tlak na snižování nákladů, flexibilitu výroby a zrychlování vývoje a uvedení produktu na trh. Proti této snaze však působí zvyšující se složitost vývoje, výroby a podpory produktu a nutnost práce s obrovským množstvím dat. Jako důsledek těchto změn byl zaveden pojem „řízení životního cyklu výrobku“, neboli PLM (Product Lifecycle Management), jako proces života výrobku od jeho zrození (prvotní myšlenky, konceptu) až po jeho likvidaci.

Zkracování doby vývoje a nutnost rychle reagovat na neustále se měnící požadavky cílových zákazníků s sebou nese i jistá úskalí, jako třeba méně času pro odladění chyb. Dnešní trhy však, díky vysoké konkurenci, neodpouštějí špatnou kvalitu nebo výrobní chyby. Firmy proto hledají nové efektivnější způsoby výroby, vývoje a testování.

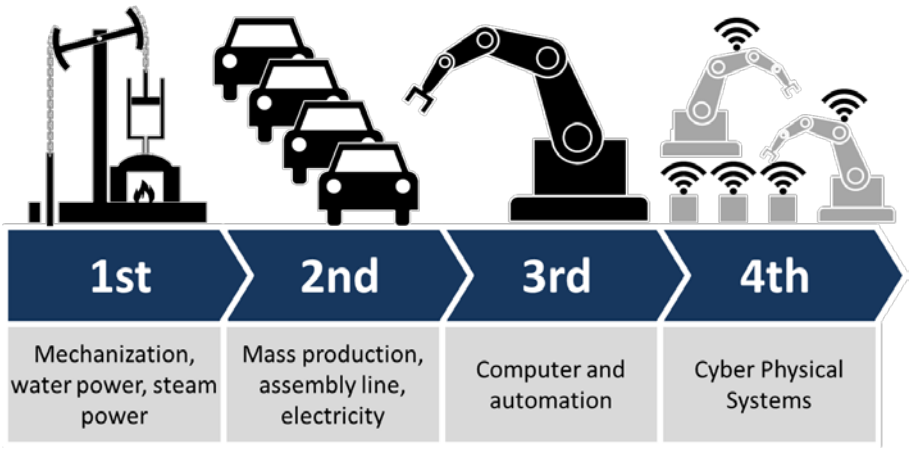
Jedním z trendů, které pomáhají zefektivnit a zkrátit vývoj produktu nebo výrobního zařízení, je digitalizace. Ta zahrnuje pojmy jako digitální továrna, digitální dvojče nebo virtuální zprovoznění. Právě virtuální zprovoznění je hlavní téma této diplomové práce.

1 VIRTUÁLNÍ ZPROVOZNĚNÍ

Virtuální zprovoznění umožňuje odladit výrobní zařízení ještě před fyzickou realizací, provedením různých simulací, testů a validací na jeho virtuálního modelu. Je jedním z hlavních rysů Industry 4.0.

1.1 Průmysl 4.0

Iniciativa nazvaná „Industrie 4.0“ (Průmysl 4.0) vznikla v Hannoveru roku 2011. Vznikla v důsledku rozmachu internetu a myšlenky propojení systémů fyzického reálného světa se světem virtuálním. Hlavním cílem je dosáhnout plně automatizované výroby, která bude schopna se autonomně přizpůsobovat vnějším vlivům, jako je změna poptávky atd. Toho je možné dosáhnout tím, že všechna výrobní zařízení budou propojeny v jedné globální síti, kde si budou autonomně vyměňovat informace (IoT). Vzniknou tak tzv. „Smart factories“ („Chytré továrny“) obsahující tzv. kyberneticko-fyzikálních systémy CPS (Cyber Physical System) jako decentralizované autonomní entity, které budou schopny spolu komunikovat, monitorovat fyzické procesy a dělat decentralizovaná rozhodnutí. [6][5]



Obr. 1-1: Vývoj průmyslu [6]

Mezi hlavní rysy Průmyslu 4.0 (podle [4]) patří:

- **Interoperabilita** - schopnost CPS, lidí a všech komponent podniku vzájemně komunikovat prostřednictvím IoT a IoS.
- **Digitalizace a virtualizace** - propojování fyzických systémů s virtuálními modely a simulačními nástroji.
- **Decentralizace** – schopnost CPS rozhodovat se autonomně na základě komunikace s ostatními CPS, bez centrálního řízení.

- **Rozhodování v reálném čase** - schopnost shromažďovat a analyzovat data a na jejich základě se okamžitě rozhodovat a poskytovat je dále.
- **Orientace na služby** - architektury SOA (Service Oriented Architectures)
- **Modularita a rekonfigurabilita** - adaptace na měnící se požadavky nahrazením nebo rozšířením jednotlivých modulů a schopnost autonomní rekonfigurace na základě automatického rozpoznání situace.
- **Umělá inteligence** – adaptace, učení z velkých dat, hledání optimálních řešení.

1.2 Digitální dvojče

Nejjednodušší a nejobecnější definice digitálního dvojčete (DT – Digital Twin) je asi taková, že se jedná o digitální repliku fyzického zařízení v reálném čase. Další používané definice jsou např.:

„*Digitální dvojče je integrovaná multifyzická, vícerozměrová, pravděpodobnostní simulace komplexního produktu, která využívá nejlepší dostupné informace pro zrcadlení života svého příslušného dvojčete*“. (Glaessgen & Stargel, 2012, volný překlad).

„*Digitální dvojče je skutečným mapováním všech složek v životním cyklu produktu pomocí fyzických dat, virtuálních dat a interakčních dat mezi nimi*.“ (Tao, Sui, Liu, Qi, Zhang, Song, Guo, Lu & Nee, 2018, doslovný překlad).

Definic DT je mnoho, ale každá zdůrazňuje spojení mezi fyzickým modelem a jeho odpovídajícím virtuálním protějškem.

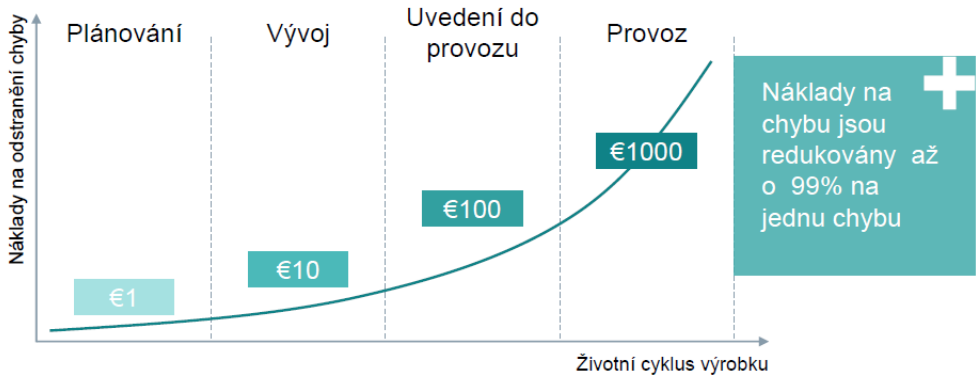
Kompletní digitální dvojče se skládá z fyzického produktu, virtuální produktu a spojení mezi nimi.

Důležitost DT spočívá v osmi hlavních funkcích, které v přístupech Průmyslu 4.0 zastává:

- Návrh a výroba produktu
- Virtuální prototypování
- Simulace, ověření funkce a zprovoznění produktu
- Efektivní a agilní výroba
- Vizualizace a monitorování životního cyklu produktu
- Školení personálu
- Nalezení a udržování optimálních parametrů při provozu výrobku
- Odhalení a odstranění nedostatků a chyb výrobku

1.3 Cíle a důvody digitalizace

Hlavním cílem je úspora nákladů díky zkrácení času potřebného pro vývoj a uvedení do provozu. Odhadem 40% nákladů na robotickou automatizovanou buňku tvoří služby (vývoj, programování a podpůrné systémy). Tento čas je možno významně zkrátit díky tomu, že jednotlivá oddělení (konstrukce, elektroprojekce a software) mohou provádět vývoj paralelně na virtuálním modelu. Další významná úspora vzniká tím, že díky simulacím jsou odhaleny chyby již v raných fázích životního cyklu výrobku. Obecně platí „pravidlo deseti“ (Obr. 1-1), které popisuje náklady na odstranění chyby v jednotlivých fázích životního cyklu. Čím dříve je chyba odhalena, tím jsou náklady na její odstranění nižší (řádově). [5]



Obr. 1-2: Pravidlo deseti (převzato z prezentace firmy Siemens)

Další výhodou je snížení rizik při skutečném oživování reálného stroje, kdy hrozí poškození zařízení vinou neodhalených mechanických kolizí. Simulace může také velmi pomoci při vyjasnění zadání mezi dodavatelem a zákazníkem a lze uspořit na prototypech. Lze také provést ověření taktu, verifikaci bezpečnosti a postupů při poruše a ověření HMI.

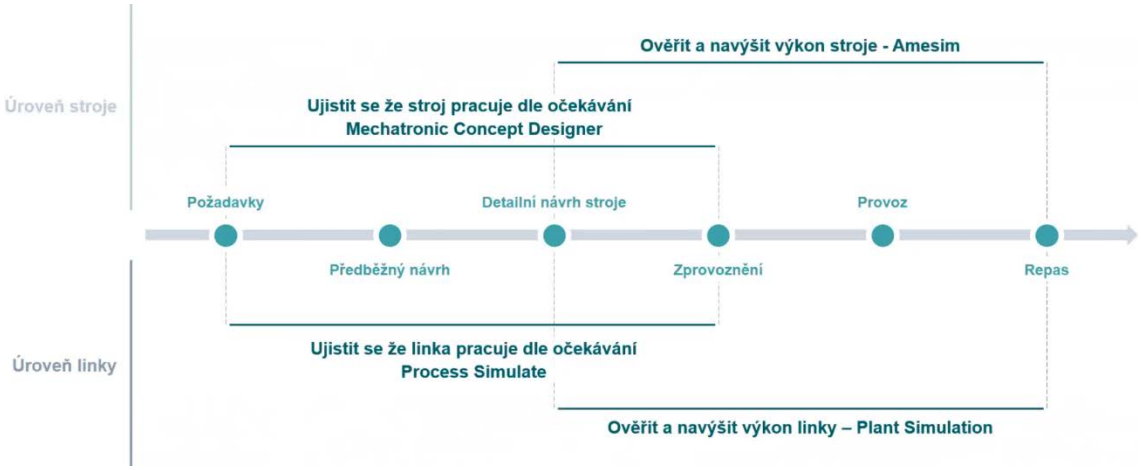
Díky zkrácení doby potřebné pro oživení a uvedení do provozu reálného stroje se významně snižují náklady na lidskou sílu, protože většinou je potřeba držet na místě při oživování programátory, elektrikáře i mechaniky. Další úspory jsou možné díky snížení objemu testovacího materiálu.

Další velkou výhodou je možnost zaškolení obsluhy a údržby zařízení na virtuálním modelu, což má za následek rychlejší náběh a méně chyb v začátcích produkce.

1.4 Simulační nástroje

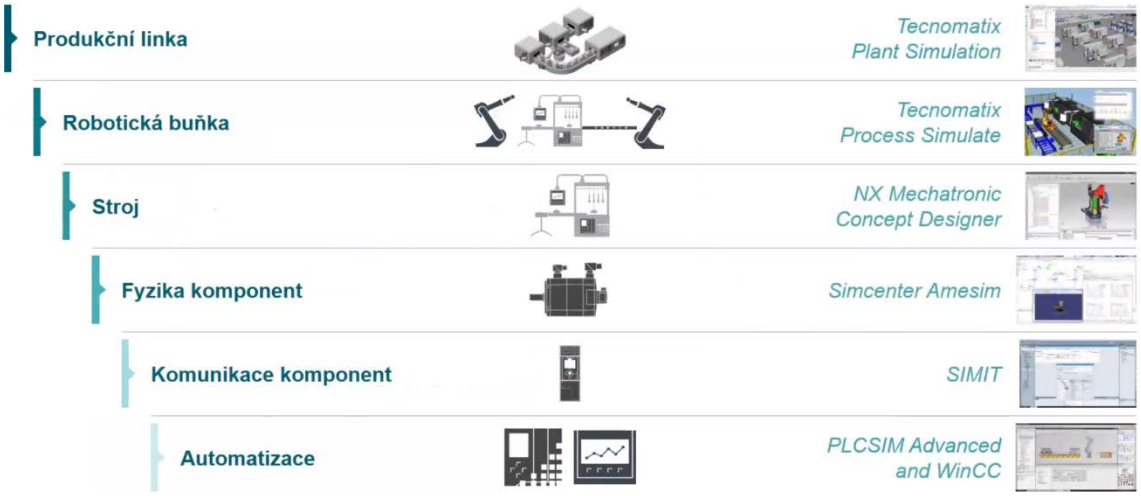
Pro virtuální zprovoznění existuje více SW nástrojů od různých výrobců. V této části představím portfolio nástrojů firmy Siemens, které zahrnuje všechny potřebné nástroje pro různé požadavky na komplexnost simulace projektu.

Siemens (hlavně jeho divize PLM) vyvíjí a udržuje spoustu simulačních nástrojů pro různé potřeby, úrovně simulace a fáze vývoje. Obr. 1-3 ilustruje rozdělení některých hlavních simulačních nástrojů podle úrovně (stroj/linka) a fáze návrhu zařízení.



Obr. 1-3: Rozdělení simulačních nástrojů firmy Siemens [7]

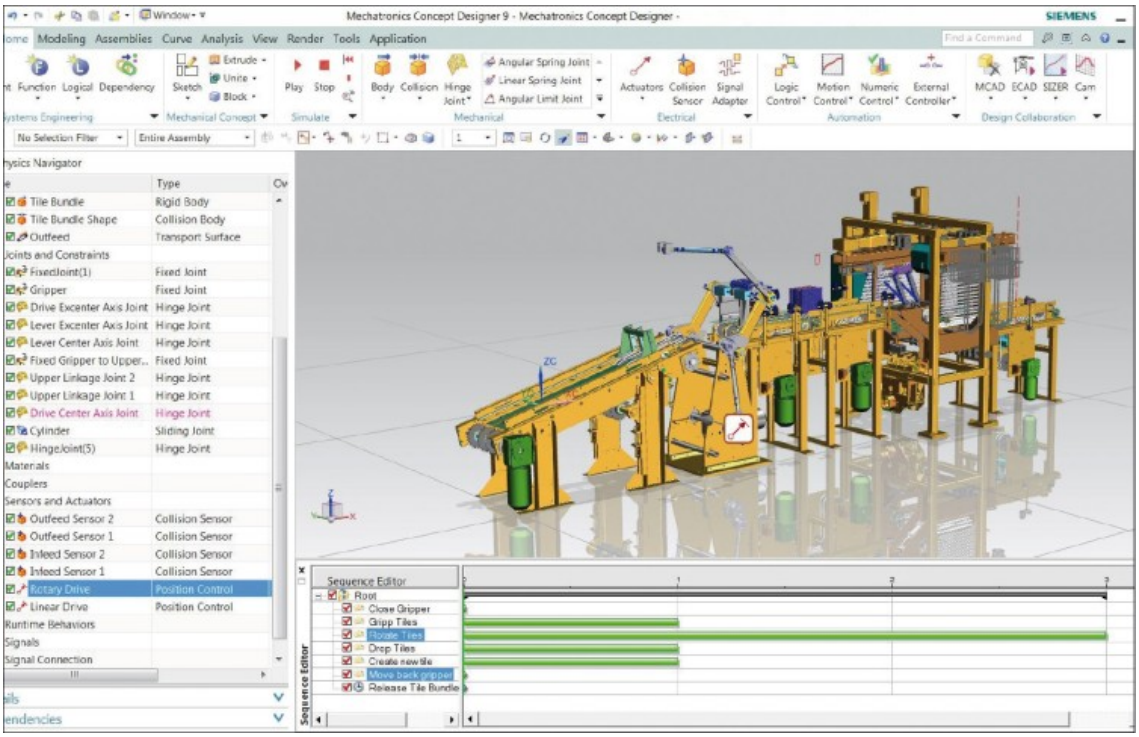
Další obrázek (Obr. 1-4) ukazuje trochu jiné rozdělení simulačních nástrojů podle detailu simulovaných částí.



Obr. 1-4: Rozdělení simulačních nástrojů firmy Siemens [7]

1.4.1 NX Mechatronics Concept Designer

NX MCD je CAD/CAM/CAE software pro detailní modelování strojů od jednotlivých dílů po rozsáhlé sestavy. Kromě modelování lze provádět výpočty, simulace, analýzy, tvorbu výkresové dokumentace, programování, simulaci NC obráběcích strojů, atd. Všechny tyto oblasti jsou vzájemně provázány a tím se zvyšuje efektivita celého řešení. Nástroj je postavený na jednotném, otevřeném a moderním technologickém základě a je ho možno integrovat do podnikového informačního (ERP) systému.



Obr. 1-5: Ukázka prostředí NX MCD [10]

NX MCD je vhodný pro detailní modelování a simulaci na nejnižší a nejpodrobnější úrovni simulace (tvarově náročných dílů, jednoho stroje, maximálně jedné buňky). Pro návrh celé výrobní linky nebo haly jsou mnohem vhodnější nástroje digitální továrny TECNOMATIX.

1.4.2 TECNOMATIX

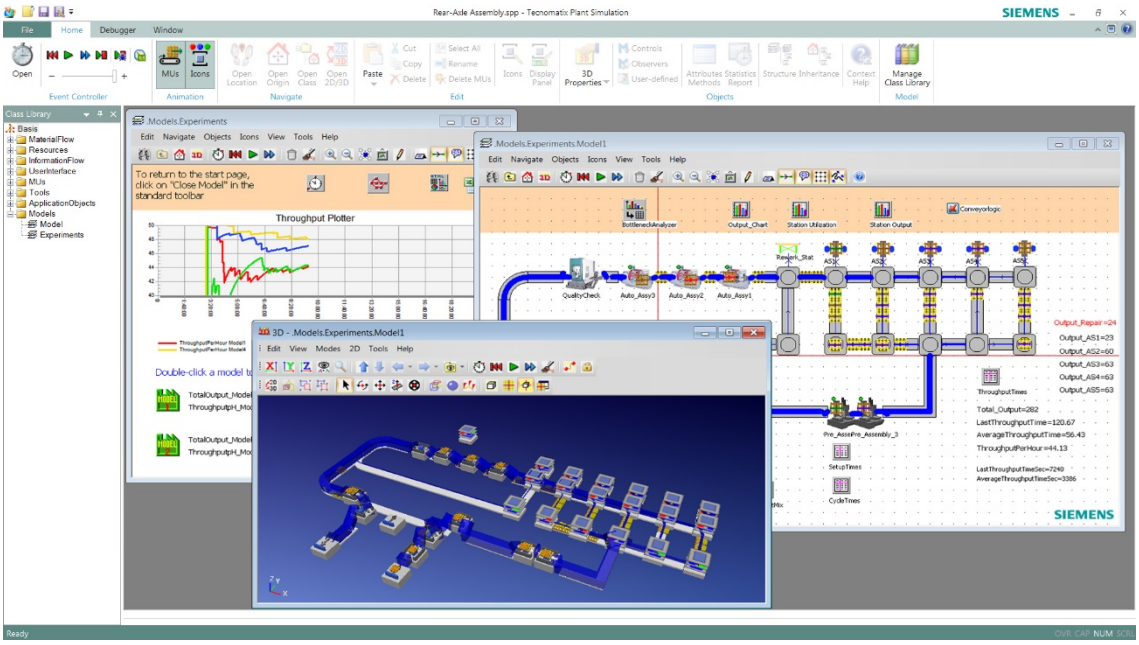
Tecnomatix je kompletní portfolio řešení digitální továrny, které propojuje všechny výrobní disciplíny s výrobním inženýrstvím. A to od návrhu a plánování, přes simulaci a ověřování, až po samotnou výrobu a její řízení. Tecnomatix je postaven na základech otevřené správy životního cyklu výrobku (PLM) nazvané Teamcenter manufacturing platform a poskytuje nejvšestrannější sadu nástrojů digitální továrny na dnešním trhu. [3]

Digitální továrna je virtuálním obrazem reálné výroby, který zobrazuje reálné výrobní procesy ve virtuálním prostředí. Systémy digitální továrny představují další krok v postupném zavádění specializovaných nástrojů pro podporu procesů v celém životním cyklu výrobku.

Tecnomatix obsahuje velké množství nástrojů pro digitální továrnu, já uvedu pouze ty nástroje, které jsou využívány hlavně pro simulace a virtuální zprovoznění.

1.4.2.1 Plant Simulation

Nástroj pro dynamickou simulaci diskrétních událostí, který umožňuje vytvářet digitální modely výrobních a logistických systémů a následně zkoumat jejich charakteristiky a optimalizovat výkonnost.



Obr. 1-6: Ukázka prostředí Plant Simulation [10]

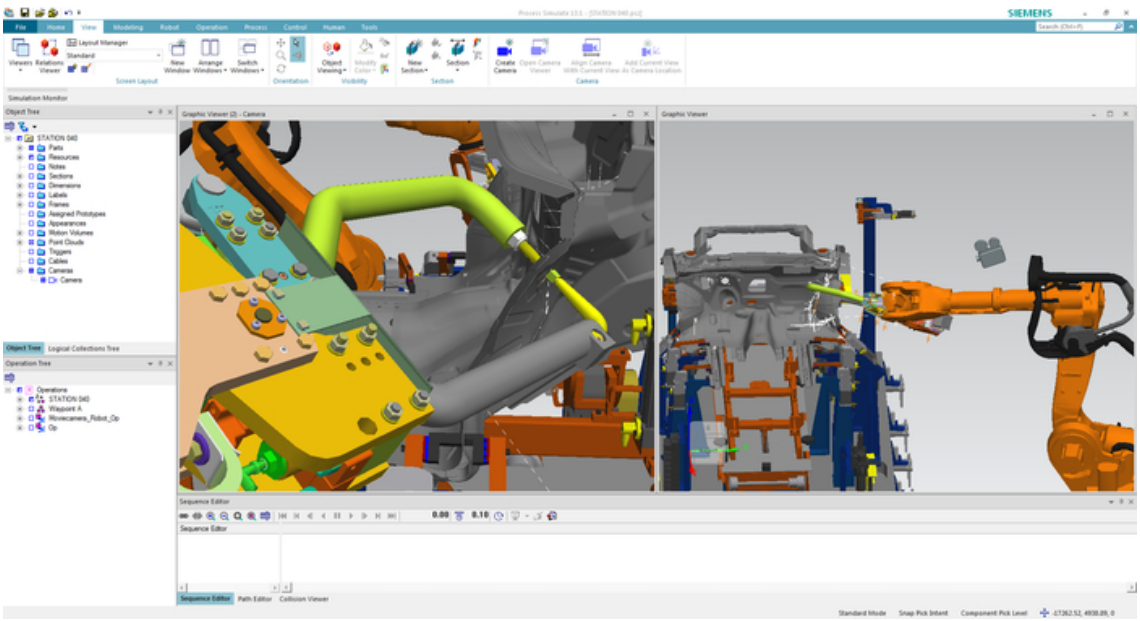
Tecnomatix Plant Simulation je vhodný pro virtuální analýzu a testování nejvyšší úrovně výroby, tedy celé linky či továrny. Konkrétně pro tyto účely:

- Analýza toku materiálu, propustnosti systému a úzkých míst.
- Optimalizace velikosti dávky a snížení velikosti zásob.
- Detekce vzájemných závislostí, snížit předimenzování.
- Simulaci různých variant řešení podle scénářů typu „Co se stane, když…“.
- Ověření logiky řízení dopravníků a PLC programu.
- Diagnostické testy systému.
- Odladění poruchových stavů.
- Školení obsluhy.

1.4.2.2 Process Simulate

Nástroj Tecnomatix Process Simulate (PS) umožňuje detailní simulaci výrobního procesu na úrovni buňky až linky. Pro modelování lze využít nástroje, které jsou součástí vývojového prostředí, nebo lze importovat modely z jiných CAD nástrojů. Je primárně určen pro tyto účely:

- Analýza kvality procesu.
- Ověření mechanických sekvencí.
- Ověření kolizních drah a testy bezpečnostních blokací.
- Analýza propustnosti (čas taktu).
- Analýza a validace bezpečnosti.
- Ověření PLC programu spolu s programy robotů a odladění poruchových stavů
- Diagnostika systému.



Obr. 1-7: Ukázka prostředí Process Simulate [10]

PS Assembly

PS Assemly umožňuje uživatelům ověřit proveditelnost procesu montáže. Nabízí simulace, analýzu a verifikaci montážních postupů a detekci kolizí při montážích.

PS Robotics

Simulační řešení pro ověření robotických operací (svařování, lakování atd.) ve 3D prostředí, zjištění pracovních zón robotů, jejich dosahu a detekci možných kolizí. Výhodou je možnost off-line programování robotů.

PS Human

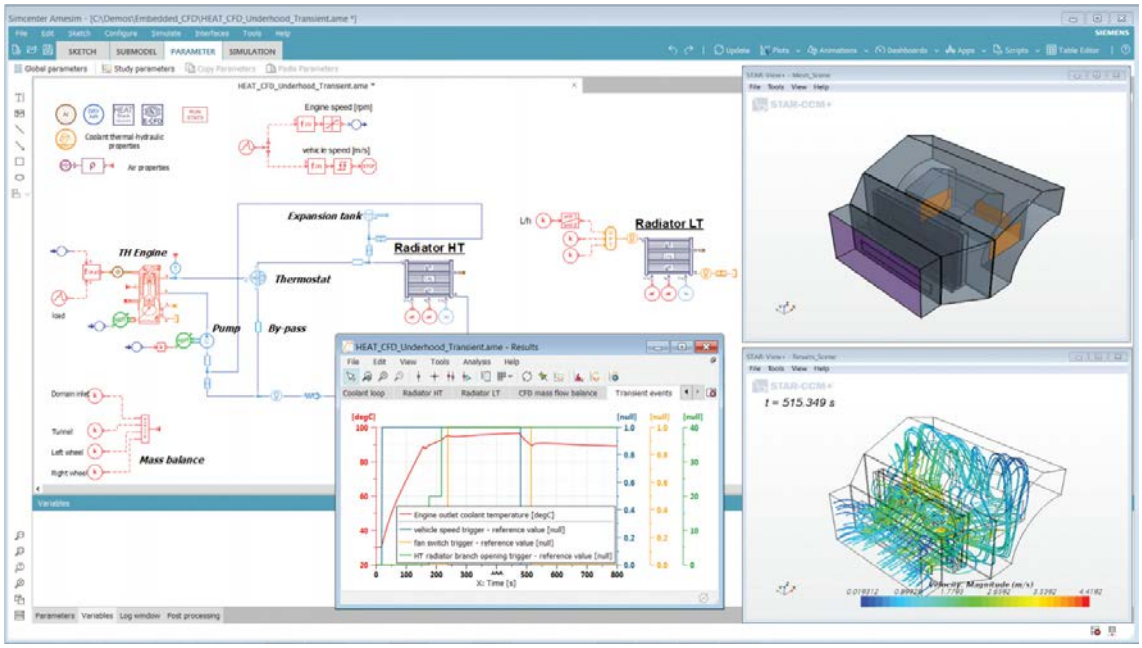
Simulace výrobních procesů s využitím manuální lidské práce (např. finální montáže v interiéru aut). Lze provádět ergonomické analýzy a zkoumat zatížení pracovníků. Tím je možno zvýšit efektivitu a ergonomii manuální práce, předcházet pracovním úrazům a nemocem z povolání.

1.4.3 Simcenter Amesim

Amesim je nástroj pro ověření, analýzu, optimalizaci a predikci výkonu mechatronických systémů v pozdější fázi vývoje, za provozu, nebo při repasi. Umožňuje vytvářet detailní simulace fyziky elektrických, hydraulických i mechanických systémů včetně řízení optimální teploty komponent atd. Je vhodný pro řešení otázek jako:

- Je pohon dostatečně výkonný?
- Jaká je časová odezva systému?
- Jaký maximální tlak může být dosažen?
- Existuje riziko vibrací?

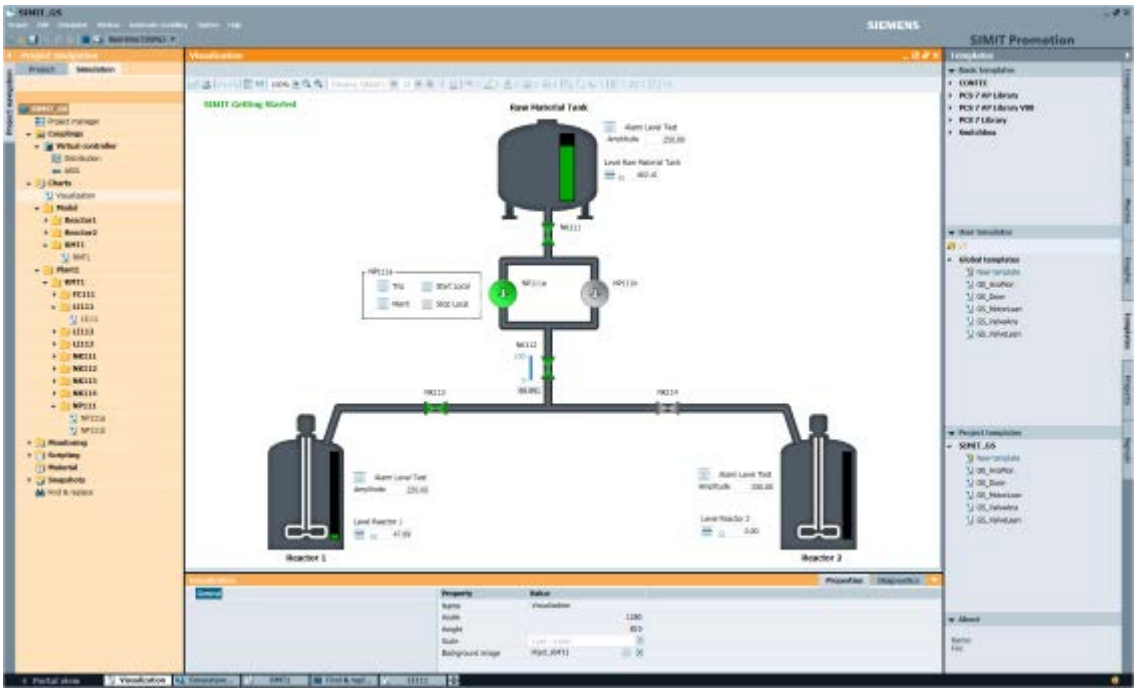
Amesim nabízí přímé napojení na fyzické PLC nebo na virtuální PLC (PLCSIM Advanced).



Obr. 1-8: Ukázka prostředí Simcenter Amesim [9]

1.4.4 SIMIT

SIMIT je „real-time“ simulační platforma pro simulaci a validaci výrobního procesu, odladění řídicího programu PLC na virtuálním modelu a zaškolení obsluhy. Nabízí simulaci chování elektrických komponent a periferií, signálů a technologického procesu. Umožňuje provádět signálové testy, „loop checks“, testy blokací, operátorských prvků, alarmů a systémových hlášení, ověření sekvencí, hodnoty hysterezí atd.



Obr. 1-9: Ukázka prostředí SIMIT

Tento simulační nástroj je určený pouze pro PLC řady SIMATIC S7 a lze ho připojit k reálnému PLC (HiL – Hardware in the Loop) nebo k simulovanému PLC s využitím PLCSIM Advanced (SiL – Software in the Loop).

1.4.5 PLCSIM Advanced

PLCSIM Advanced je nástroj pro vytvoření virtuálního dvojčete PLC kontroléru SIMATIC S7-1500. Tvoří základ pro simulace typu SiL, kde je simulované výrobní zařízení řízeno simulovaným PLC. Nabízí přímá aplikační rozhraní pro další simulační nástroje, včetně SIMIT, NX MCD, Tecnomatix Plant Simulation a Process Simulate a Simcentrer Amesim.

Tento nástroj je jedno z hlavních témat této diplomové práce a je mu věnována celá druhá kapitola.

1.5 TIA Portal

TIA (Totally Integrated Automation) Portal je inženýrský rámec, který sdružuje všechny potřebné SW nástroje pro automatizaci výrobního zařízení.

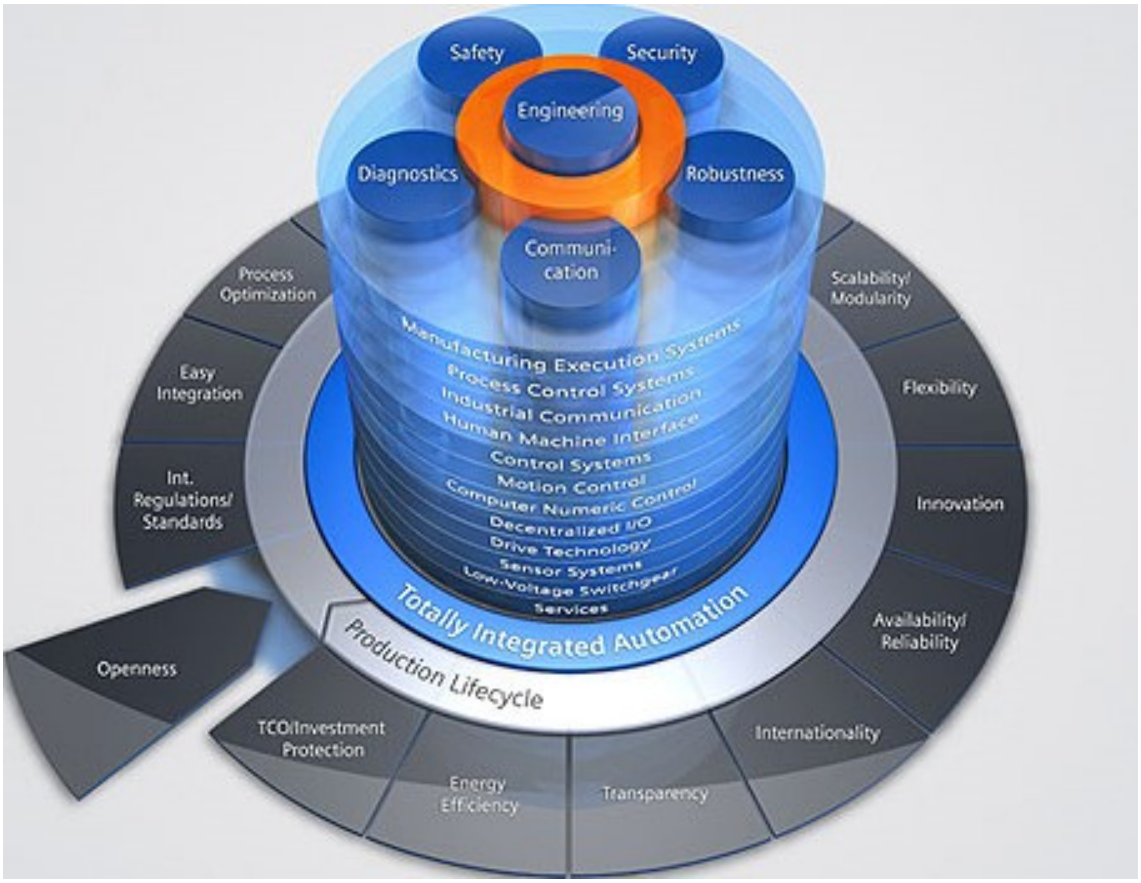
TIA Portal v sobě obsahuje (nebo umožňuje integrovat) tyto inženýrské nástroje:

- STEP 7 - modul pro programování a konfiguraci PLC.
- WinCC - nástroj pro tvorbu vizualizace pro HMI panely.
- StartDrive - nástroj pro konfiguraci a nastavení frekvenčních měničů.
- SIMOTION Scout - modul pro programování motion systémů.

Další volitelné součásti, které jsou obsaženy nebo je možné doinstalovat, jsou např.:

- STEP 7 Safety – rozšíření STEP 7 o programování safety aplikací.
- TIA Portal Openness – otevřené aplikační rozhraní (API), umožňující realizaci příkazů v prostředí TIA Portal z externí aplikace. Lze tak pomocí skriptů např. automatizovat některé často se opakující činnosti (třeba generování bloků a symbolů podle standartu).
- PLCSIM – nástroj pro simulaci fyzického PLC.
- TIA Portal Multiuser – nástroj umožňující paralelní práci více lidí ve stejném projektu. Obsahuje přehlednou indikaci právě editovaných objektů.
- SIMATIC Visualization Architect (SiVArc) – automatické generování HMI vizualizace na základě uživatelského PLC programu (ze STEP 7).
- SIMATIC OPC UA – otevřený komunikační standard pro komunikaci se zařízeními různých výrobců.
- TIA Portal Cloud Connector – umožňuje přístup k PLC a HMI z privátního cloudu (s nainstalovaným TIA Portalem).
- SIMATIC ODK 1500S – možnost začlenění C/C++ programů do PLC.
- SIMATIC Target 1500S for Simulink – doplněk pro MATLAB Simulink. Umožňuje integraci Simulink modelů přímo v programovém cyklu S7-1500.
- SIMATIC ProDiag - detailní monitorování zařízení s minimálními náklady na konfiguraci a vizualizaci. Je schopen detekovat chyby v řízeném procesu a poskytovat informace o typu, místě výskytu a příčině těchto chyb.
- SIMATIC Energy Suite - zvyšuje transparentnost energetických toků a pomáhá šetřit energii. Zjednodušené programování komponent pro měření energie a intuitivní konfigurace správy energie.
- SIMATIC OPC UA – podpora přímo v PLC S7-1500, nabízí pokročilejší, flexibilnější a otevřenější možnosti komunikace a přímý přístup k datům z PLC odkudkoliv, především ze zařízení třetích stran.

- SIMATIC WinCC/WebUX – nabízí moderní metody přístupu k vizualizaci z tabletů a chytrých telefonů.



Obr. 1-10: Koncept Totally Integrated Automation

Hlavní výhody TIA Portalu jsou tedy integrace prostředí všech inženýrských nástrojů a společné tagy, díky čemuž odpadá nutnost tvořit duplicitní tagy v každém nástroji zvlášť a umožňuje hledání pomocí křížových referencí skrz všechny nástroje. Další užitečnou vlastností je kompletní přehled topologie a komunikací jednotlivých zařízení.

Poměrně novou vlastností je integrace programování robotů do TIA Portalu. Roboty je nyní možné programovat přímo ve STEP 7 využitím knihoven dodaných výrobci robotů. Klasický způsob programování robotů, kdy se z PLC programu volají rutiny uložené v kontroléru robotu, tak může být nahrazen novým přístupem, kdy se z PLC programu pomocí funkcí řídí přímo dráhy robotu. Výhody jsou: jedno návrhové prostředí, možnost sjednocení HMI panelu zařízení (linky) a robotu, programování robotů od více výrobců stejným způsobem (pouze se vymění knihovny), méně času na výuku atd.

2 PLCSIM ADVANCED

Tato kapitola obsahuje souhrn vybraných důležitých informací z manuálu firmy Siemens k nástroji PLCSIM Advanced ([1], přes 350 stran), doplněné o osobní postřehy a zkušenosti z jeho používání. Informace se vztahují se k verzi PLCSIM Advanced V2.0.

2.1 Vlastnosti

PLCSIM Advanced je, jak už bylo uvedeno, digitální dvojče kontroléru Siemens SIMATIC S7-1500 a umožňuje přesně simulovat všechny jeho vlastnosti, funkce, komunikace a web server.

2.1.1 Porovnání s PLCSIM V15 (TIA Portal)

TIA Portal již obsahuje simulační nástroj PLCSIM, který také dovede do jisté míry simulovat chování PLC kontrolérů SIMATIC S7-1200 a S7-1500, ale jeho možnosti jsou omezené. PLCSIM je napevno integrovaný v TIA Portalu a nemůže běžet ani být nainstalován bez něj. Jeho uživatelské rozhraní je rovněž integrováno v TIA Portalu.

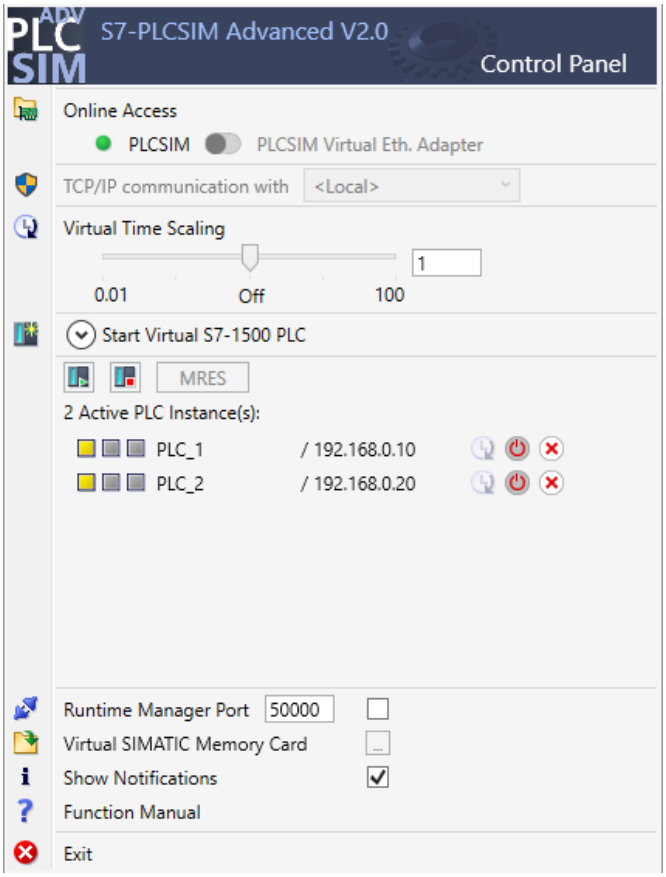
PLCSIM Advanced je mnohem více než jen PLCSIM s několika funkcemi navíc. Je to nezávislý nástroj, který lze instalovat samostatně a obsahuje vlastní uživatelské rozhraní (Control panel). Hlavní rozdíly jsou vypsány v následující tabulce:

Tabulka 1: Porovnání funkcí PLCSIM produktů

	PLCSIM Advanced V2.0	PLCSIM V15
Runtime	nezávislý	společně s TIA Portal
GUI	Control panel	TIA Portal
Komunikace	Softbus, TCP/IP	Softbus
Podporované CPU	S7-1500	S7-1200, S7-1500
API	ano	ne
Web server	ano	ne
OPC UA	ano	ne
Procesní diagnostika	ano	ano
S7 komunikace	ano	ne
Chráněné bloky	ano	pouze S7-1500
Vícenásobné instance	až 16	až 2
Distribuované instance	ano	ne
Virtuální čas	ano	ne
Komunikace s reálnými PLC/HMI	ano	ne
Použití DNS	ano	ne

2.1.2 Control panel

Control panel je uživatelské rozhraní, které umožňuje ovládat hlavní funkce PLCSIM Advanced, jako přidávání, odebírání a ovládání PLC instancí, nastavení komunikace a ovládání virtuálního času. Panel se však vůbec nemusí používat a všechny tyto funkce lze ovládat z externí aplikace přes API. Lze ho zobrazit po kliknutí pravým tlačítkem myši na příslušnou ikonu v oznamovací oblasti hlavního panelu Windows.



Obr. 2-1: Ovládací panel PLCSIM Advanced

2.1.3 Komunikace

PLCSIM Advanced podporuje několik typů komunikace a komunikačních protokolů:

- Součástí instalace je virtuální Ethernet adaptér pro simulaci komunikace prostřednictvím protokolu TCP/IP.
- Podporuje komunikaci lokální (Softbus, TCP/IP) i distribuovanou (TCP/IP).
- Každá PLC instance má svůj vlastní web server.
- Podporuje komunikaci přes OPC UA.
- Podpora DNS.
- Podpora S7 protokolu – lze komunikovat s reálnými PLC a HMI

Jednotlivé PLC instance mohou samozřejmě komunikovat mezi sebou a mohou dokonce komunikovat se simulovanými PLC staršího nástroje PLCSIM v5.x, tedy s virtuálními PLC starší řady S7-300. Nelze však komunikovat s modulem PLCSIM v TIA Portalu a komunikace s kontroléry S7-1200 tak není možná.

2.1.4 Kompatibilita

PLCSIM Advanced V2.0 je kompatibilní s TIA Portal ve verzích V14 a V15 a se všemi CPU S7-15xx s verzemi firmwaru: V1.8, V2.0, V2.1, V2.5.

2.1.5 Virtuální čas

Důležitá funkce je synchronizace časů PLCSIM Advanced se všemi kosimulačními aplikacemi. Nestane se tedy, že by jedna aplikace běžela rychleji než jiná.

2.1.6 Rozdíly mezi simulovaným a reálným PLC

Ačkoli PLCSIM Advanced simuluje reálný kontrolér velmi přesně, nedokáže přesně napodobit všechny detaily. Takže i když je řídící program nahrán a virtuální PLC běží bez chyb, neznamená to, že se bude chovat naprosto přesně jako reálné PLC. Hlavní rozdíly jsou popsány v následujících odstavcích.

Protože PLCSIM Advanced běží na PC s operačním systémem Windows, čas cyklu nebo vykonání instrukce nebude stejný jako u fyzického PLC. Důvodem je, že sdílí prostředky (jádro CPU) s dalšími aplikacemi, běžícími pod Windows, a operační systém mezi nimi přepíná. Pro zajištění co nejvíce deterministického chování je vhodné, aby každá PLC instance měla vlastní volné jádro CPU jen pro sebe.

Programové bloky, které mají nastaven „know-how protection“, můžou být simulovány, pouze pokud jsou v TIA Portal povoleny pro simulaci. Pro povolení této volby je potřeba znát jejich heslo.

Mezi další omezení, která platí pro všechna podporovaná CPU, patří:

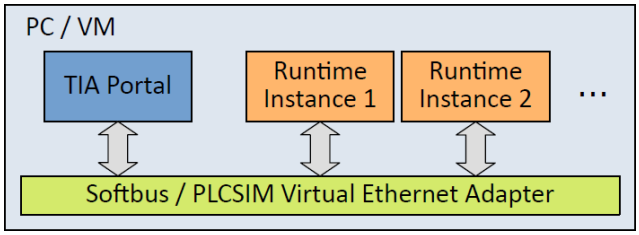
- Chybějící podpora instrukcí DP_TOPOL a PORT_CFG.
- Nelze simulovat sběrnicové systémy (Profinet IO, Profibus DP).
- Nelze simulovat konfigurované I/O moduly (ani u kompaktních PLC).
- Nejsou podporovány některé pro simulaci nepotřebné funkce, jako třeba formátování paměťové karty, nebo aktualizace firmwaru.
- Chybí podpora záznamu dat (data logging).
- Nelze simulovat receptury.
- Nelze simulovat ochranu proti kopírování (copy protection).

Existují ještě některá omezení pro Web server, OPC UA, motion control a další. Všechna detailní omezení lze dohledat v manuálu [1].

2.2 Možnosti komunikace

2.2.1 Lokální komunikace

Při lokální komunikaci jsou všechny PLC instance i TIA Portal na stejném PC. Lokální komunikace je možná buď přes Softbus nebo přes protokol TCP/IP (Ethernet). PLC instance mohou takto komunikovat mezi sebou a s TIA Portalem, nelze však komunikovat s reálným HW (PLC, HMI atd.). Schéma tohoto způsobu komunikace je na Obr. 2-2.



Obr. 2-2: Lokální komunikace

2.2.1.1 PLCSIM Softbus

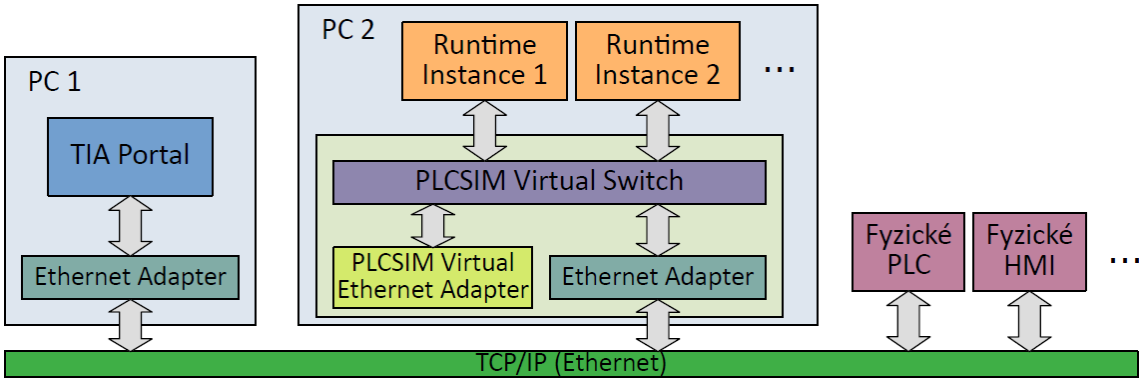
Toto je výchozí nastavení komunikace v PLCSIM Advanced. Při tomto způsobu komunikace nemají instance IP adresu (respektive mají všechny stejnou IP adresu, která není brána v potaz). Nelze proto navázat komunikaci s fyzickým HW a nelze využívat OPC UA ani web server. Výhodou je, že žádná data nemohou být omylem nahrána do fyzického PLC.

2.2.1.2 TCP/IP (Virtual Ethernet Adapter)

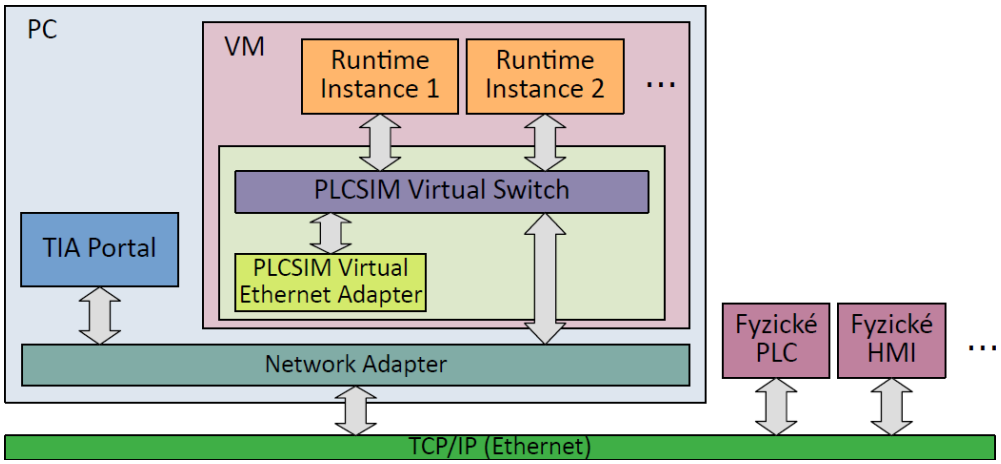
Komunikace je zprostředkována přes PLCSIM Virtual Ethernet Adapter, který je součástí instalace PLCSIM Advanced, a chová se stejně jako fyzické síťové rozhraní. Každá instance má vlastní IP adresu a je tak možné využívat OPC UA i web server.

2.2.2 Distribuovaná komunikace

Distribuovaná komunikace je možná pouze přes TCP/IP. Instance PLCSIM Advanced komunikují s dalšími zařízeními a SW nástroji přes PLCSIM Virtual Switch. Díky tomu lze komunikovat nejen se simulačními nástroji a simulovanými zařízeními, ale i s reálnými zařízeními (PLC/HMI), přičemž mohou být na jiném PC (Obr. 2-3) nebo na virtuálním stroji (VM, Obr. 2-4).

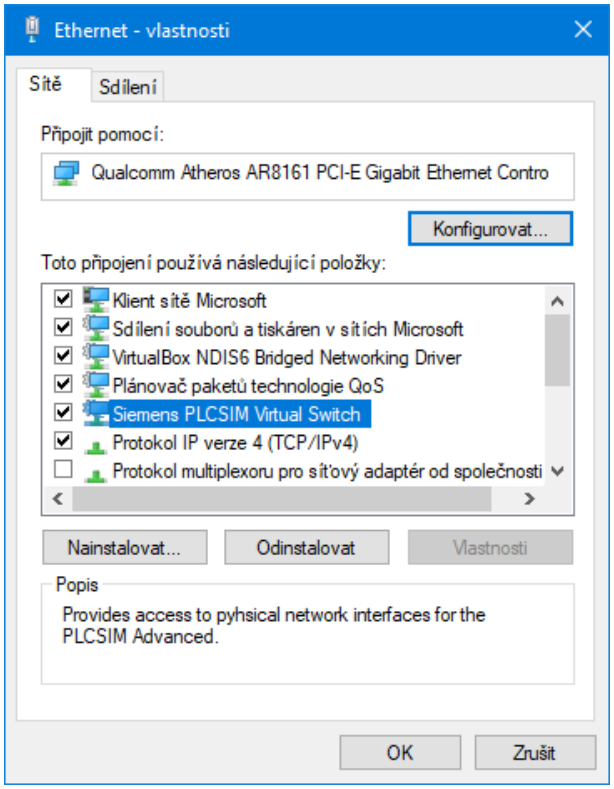


Obr. 2-3: Příklad distribuované komunikace PC <-> PC



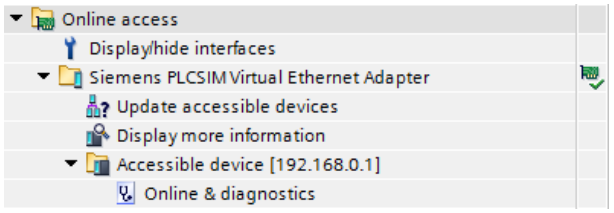
Obr. 2-4: Příklad distribuované komunikace PC <-> VM

Aby mohl PLCSIM Virtual Switch komunikovat s jinými zařízeními a aplikacemi na jiném PC nebo VM, je potřeba povolit „Siemens PLCSIM Virtual Switch“ v nastavení fyzického síťového adaptéru (Obr. 2-5) počítače, na kterém je PLCSIM Advanced nainstalován. K nastavení adaptérů se lze ve Windows dostat přes volbu Ovládací panely->Centrum síťových připojení a sdílení->Změnit nastavení adaptéru. Virtual Switch lze povolit na kabelovém (Ethernet) i na bezdrátovém (Wi-Fi) adaptéru. Nastavíme ty adaptéry, přes které plánujeme komunikovat.



Obr. 2-5: Povolení PLCSIM Virtual Switch

Jakmile je PLCSIM Virtual Switch aktivován, TIA Portal jej zobrazí v záložce „Online access“ (Obr. 2-6) včetně všech zařízení, které přes něj komunikují.



Obr. 2-6: Záložka „Online access“ ve stromu projektu v TIA Portal

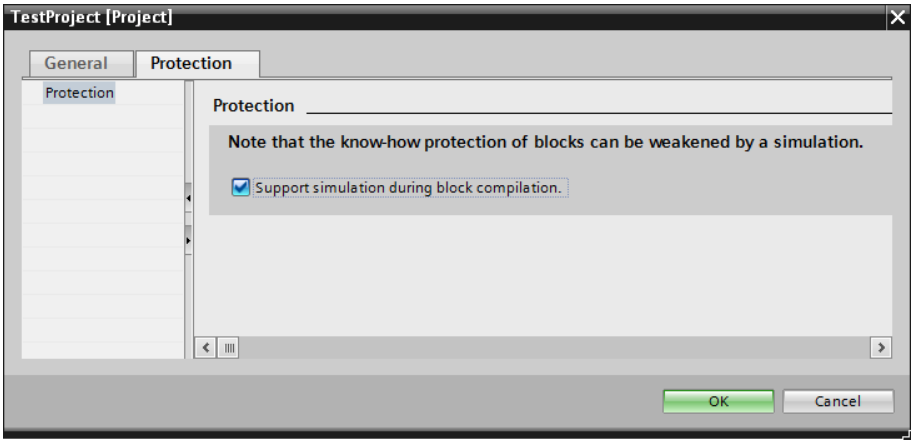
2.3 Nahrání programu do PLCSIM instance

Jsou dva způsoby, jak nahrát připravenou HW a SW konfiguraci do samotného virtuálního PLC. Buď lze nahrát program z TIA Portalu napřímo do instance, nebo přes virtuální paměťovou kartu.

2.3.1 Přímo do virtuálního PLC z TIA Portalu

V tomto případě se program nahrává stejným způsobem jako při nahrávání do fyzického PLC, přes funkci „Download“ v TIA Portalu. Je však potřeba nejprve zaškrtnout volbu "Support simulation during block compilation" v nastavení TIA

Portal projektu (Edit -> Properties) v záložce „Protection“ (Obr. 2-7). Cílová PLCSIM Advanced instance musí být spuštěna a musí být dosažitelná přes zvolené komunikační rozhraní.

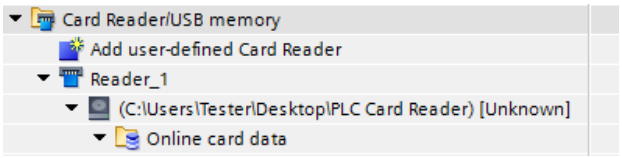


Obr. 2-7: Nastavení projektu v TIA Portal

2.3.2 Virtuální paměťová karta

Pokud není z jakéhokoliv důvodu možné nahrát program napřímo (například pokud nejsou TIA Portal a PLCSIM Advanced na stejném PC ani na stejné síti), lze HW a SW konfiguraci „nahrát“ přes virtuální paměťovou kartu. Postup je následující:

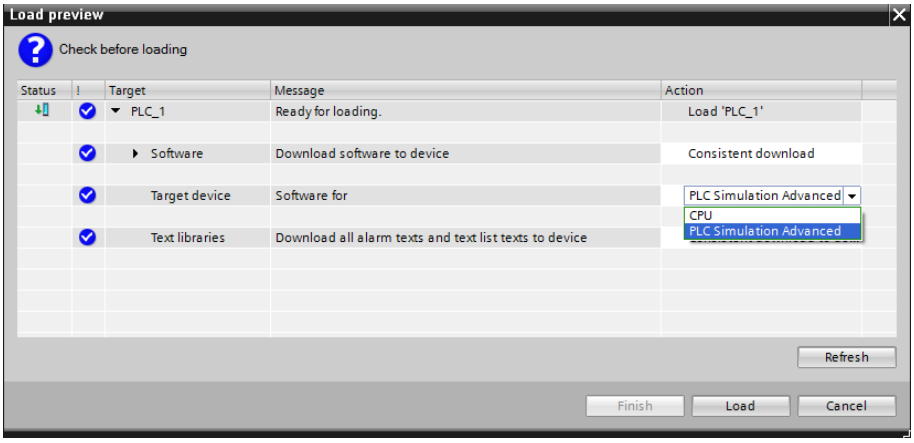
1. Přidat uživatelskou čtečku karet pomocí Project -> Card Reader/USB memory -> Add user-defined Card Reader a definovat cestu ke složce (virtuální kartě), do které bude program nahrán.



Obr. 2-8: Přidání uživatelské čtečky karet

2. Ve stromové struktuře TIA Portal projektu označit PLC, jehož program chceme nahrát na kartu, aktivovat příkaz Project -> Card Reader/USB memory -> Write to memory card… a vybrat vytvořenou čtečku.
3. V dialogovém okně „Load preview“ zvolit jako cílové zařízení (Target device) „PLC simulation Advanced“ a program nahrát (Obr. 2-9).
4. Přenést složku virtuální karty s nahraným programem do PC, na kterém se nachází PLCSIM Advanced.

5. V Control panelu PLCSIM Advanced kliknout na „Virtual SIMATIC Memory Card“, otevřít složku požadované instance a do složky \SIMATIC_MC přesunout z virtuální karty složku \SIMATIC.S7S.
6. Po spuštění instance naběhne s nahraným programem.



Obr. 2-9: Volba cílového zařízení v dialogovém okně „Load preview“

2.4 Virtuální čas

Virtuální PLC pro simulaci interně používá dva typy hodin, reálné a virtuální. Základ pro simulaci jsou vždy virtuální hodiny. Běží podle nich PLC program, cyklické OB bloky, systémový čas PLC, čas cyklu, výpočty atd. Je možné je zpomalovat nebo zrychlovat. Reálné hodiny běží vždy nezměněny a jsou využívány pro věci, které nejsou součástí řízení procesu (technologie), například pro komunikaci s TIA Portalem.

2.4.1 Systémový čas virtuálního PLC

Při startu PLCSIM Advanced je virtuální systémový čas virtuálního PLC stejný jako systémový čas operačního systému Windows. Jeho rychlost odpovídá rychlosti (časovému měřítku) virtuálních hodin, na kterých je založen. Veškeré události, které PLCSIM instance poskytuje přes API, obsahují časovou značku založenou na tomto času.

2.4.2 Časové měřítko a rychlost simulace

Pomocí časového měřítka můžeme zrychlovat nebo zpomalovat běh virtuálních hodin a tím i čas celé simulace.

Výchozí hodnota časového měřítka je 1, při které průběh virtuálního času odpovídá průběhu reálného času. Při hodnotě časového měřítka větší než 1 jsou virtuální hodiny zrychleny (např. při hodnotě 2,5 je simulace 2,5krát rychlejší oproti

reálnému času). Pokud je hodnota časového měřítka menší než 1, je virtuální čas zpomalen (např. při hodnotě 0,5 je rychlost simulace poloviční oproti reálnému času).

2.4.2.1 Omezení

Změna časového měřítka nemůže ovlivnit rychlost vykonání jednotlivých instrukcí PLC programu. Ta je závislá na rychlosti procesoru PC, na kterém běží PLCSIM Advanced.

Při zrychlení virtuálního času se budou (z hlediska reálného času) cyklické objekty (např. cyklické OB) vykovávat odpovídajícím způsobem častěji, ale samotné vykonání posloupnosti instrukcí bude trvat pořád stejnou dobu. Tím reálně hrozí, že pokud bude rychlost simulace příliš velká, doba vykonání bloku OB1 překročí maximální nastavený čas (který je závislý na virtuálních hodinách) a PLC se přepne do režimu STOP.

Maximální rychlost simulace závisí na více faktorech (velikost a skladba projektu, nastavení maximálního času vykonání OB1, výkon procesoru). Doporučení je začít s malými hodnotami časového měřítka a postupně po malých krokách zvyšovat.

2.5 Synchronizace simulace

2.5.1 Pozastavení simulace PLC - stav „Freeze“

Pro pozastavení simulace a synchronizace s dalšími kosimulačními programy může být virtuální PLC přepnuto do stavu „Freeze“. Tento vnitřní stav neodpovídá žádnému operačnímu stavu reálného PLC (RUN/STOP) a po jeho ukončení je PLC přepnuto zpět do posledního operačního stavu.

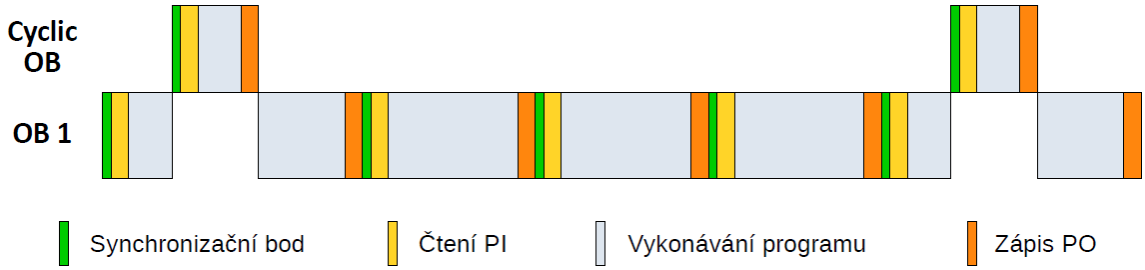
Ve stavu „Freeze“ se děje následující:

- Virtuální čas je pozastaven.
- Neběží žádné OB ani časovače.
- Uživatelský PLC program není nadále vykonáván.
- K virtuálnímu PLC je nadále možné se připojit z TIA Portalu.
- I/O data virtuálního PLC jsou v konzistentním stavu.

PLC může být přepnuto do stavu „Freeze“ pouze v okamžiku dosažení synchronizačního bodu.

2.5.2 Synchronizační bod

Synchronizační bod se nachází před každým okamžikem čtení vstupů (PI – Process Input), např. na konci cyklu PLC (blok OB1) nebo před začátkem každého vykonání cyklického OB. Časový diagram výskytu synchronizačního bodu je znázorněn na následujícím obrázku:



Obr. 2-10: Časový diagram výskytu synchronizačního bodu

Událost „OnSyncPointReached“

Tato událost je vyvolána v okamžiku dosažení synchronizačního bodu a je přes API poskytována simulačním partnerům, kteří na ni mohou programově reagovat (např. načíst výstupy PLC, vykonat vlastní algoritmus a zapsat na vstupy PLC). Událost však nemusí být vyvolána při každém dosažení synchronizačního bodu, dokonce nemusí být vyvolána vůbec. Závisí to na zvoleném operačním módu.

2.5.3 Operační módy

Operační mód virtuálního PLC rozhoduje o tom, při kterých synchronizačních bodech je vyvolána událost „OnSyncPointReached“ a přepne do stavu „Freeze“.

Výchozí operační mód je „Default“. V tomto módu není virtuální PLC nikdy přepnuto do stavu „Freeze“ a událost „OnSyncPointReached“ je vyvolána při dosažení synchronizačního bodu pouze v případě, že je vlastnost PLC instance „IsSendSyncEventInDefaultModeEnabled“ nastavena (přes API) na hodnotu „true“ (více v části Cyklus virtuálního PLC na str. 41).

Operační mód virtuálního PLC je možno změnit na některý z dalších, které spadají do kategorie buď cyklového řízení, nebo časového řízení.

2.5.3.1 Cyklově řízená synchronizace

Při těchto operačních módech je vždy při dosažení příslušného synchronizačního bodu vyvolána událost „OnSyncPointReached“ a PLC přepnuto do stavu „Freeze“.

Volbou konkrétního módu je určeno při jakých synchronizačních bodech je vyvolána událost a pozastaveno PLC (jestli při všech, nebo jen před začátkem OB1,

nebo jen před začátkem cyklického OB) a jestli má být přepsána minimální doba cyklu PLC

Tabulka 2: Operační módy - cyklové řízení

Operační mód	Synchronizační bod		Minimální čas cyklu „T“
	Cyklus PLC (OB 1) „C“	Cyklické OB „P“	
SingleStep_C	ANO	NE	NE
SingleStep_P	NE	ANO	NE
SingleStep_CP	ANO	ANO	NE
SingleStep_CT	ANO	NE	ANO
SingleStep_CPT	ANO	ANO	ANO

Stav „Freeze“ je po dokončení synchronizace vždy nutno zrušit přes API zavoláním funkce RunToNextSyncPoint().

2.5.3.2 Časově řízená synchronizace

Při tomto řízení synchronizace je stanoven minimální čas, po který musí PLC běžet nepozastaven. Než tento čas uplyne, jsou všechny synchronizační body ignorovány. Po uplynutí času je po dosažení prvního synchronizačního bodu (podle konkrétního operačního módu) vyvolána událost „OnSyncPointReached“ a PLC je přepnut do stavu „Freeze“.

Minimální čas se předává jako parametr funkce StartProcessing(t) a je nutné ji zavolat vždy po dokončení synchronizace pro zrušení stavu „Freeze“.

Tabulka 3: Operační módy - časová synchronizace

Operační mód	Synchronizační bod	
	Cyklus PLC (OB 1) „C“	Cyklické OB „P“
TimespanSynchronized_C	ANO	NE
TimespanSynchronized _P	NE	ANO
TimespanSynchronized _CP	ANO	ANO

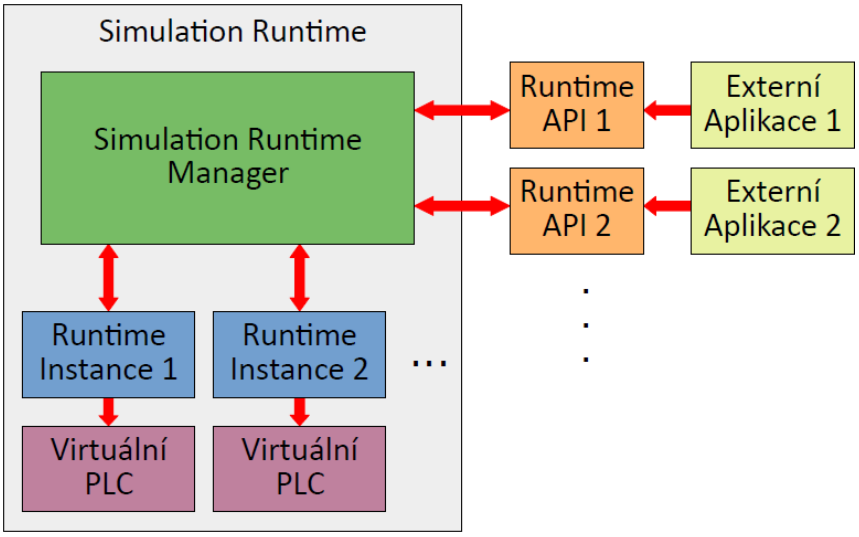
2.6 Aplikační rozhraní (API)

PLCSIM Advanced poskytuje otevřené aplikační rozhraní pro interakci s dalšími simulačními nástroji a aplikacemi psanými nativně v jazyce C++ nebo C#. Takovéto spolupráci více simulačních nástrojů se říká „kosimulace“.

Díky tomuto API mají kosimulační nástroje přístup k datům virtuálních PLC (I/O, merkery, datové bloky, časovače, operační stav, …) a mohou nahradit funkce control panelu PLCSIM Advanced a tedy vytvářet/mazat instance PLC, měnit jejich operační stav (RUN/STOP), nastavovat komunikaci atd.

2.6.1 Runtime procesy a knihovny

Na Obr. 2-11 je znázorněno schéma runtime PLCSIM Advanced včetně přístupu externích aplikací. Proces Simulation Runtime řídí procesy Runtime Instance, a ty potom načítají knihovny virtuálních kontrolérů.



Obr. 2-11: Schéma runtime komponent PLCSIM Advanced

Detailní schéma přístupu externí aplikace k API je zobrazeno na Obr. 2-12. Externími aplikacemi mohou být například další simulační software nebo jiné grafické uživatelské rozhraní (GUI).

2.6.1.1 Simulation Runtime Manager

Simulation Runtime Manager (SRM) je Windows proces běžící na pozadí. Je to hlavní runtime komponenta, která řídí ostatní komponenty. Spouští se automaticky, jakmile se aplikace pokusí inicializovat runtime API. Ukončí se automaticky, pokud nadále neběží žádná aplikace, která runtime API inicializovala.

Název procesu:

- Siemens.Simatic.Simulation.Runtime.Manager.exe

2.6.1.2 Runtime Instance

Runtime Instance je proces PLC instance, který načítá knihovnu virtuálního PLC. Každá PLC instance má svůj vlastní proces.

Název procesu:

- Siemens.Simatic.Simulation.Runtime.Instance.exe

2.6.1.3 API knihovny

Tyto knihovny obsahují rozhraní a funkce pro aplikace psané v nativním kódu (C++) i pro aplikace psané v řízeném kódu (C#). Externí aplikace musejí tyto knihovny načíst (respektive alespoň jednu z nich – podle toho jestli bude aplikace pro 32-bit nebo 64-bit systémy), aby mohly komunikovat s procesy PLCSIM Advanced.

Názvy knihoven:

- Siemens.Simatic.Simulation.Runtime.Api.x86.dll – pro 32-bit aplikace
- Siemens.Simatic.Simulation.Runtime.Api.x64.dll – pro 64-bit aplikace

Hlavičkový soubor pro C++ aplikace:

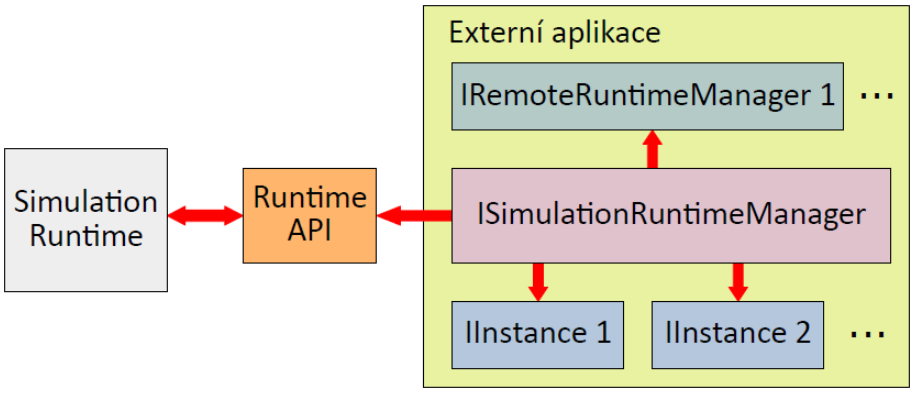
- SimulationRuntimeApi.h

2.6.2 Poskytovaná rozhraní

Poskytované knihovny pro externí aplikace, napsané v jazyce C++ nebo C#, obsahují následující rozhraní:

- ISimulationRuntimeManager – rozhraní SRM, slouží ke správě instancí a jejich rozhraní.
- IInstance – rozhraní Runtime Instance, slouží k řízení virtuálního PLC a výměně I/O dat.
- IRemoteRuntimeManager – rozhraní pro přístup ke vzdálenému SRM.

Následující obrázek (Obr. 2-12) ilustruje vzájemný vztah těchto rozhraní v externí aplikaci a přístup k Simulation Runtime přes API. Tento obrázek souvisí s Obr. 2-11, na kterém je zobrazeno schéma Simulation Runtime.

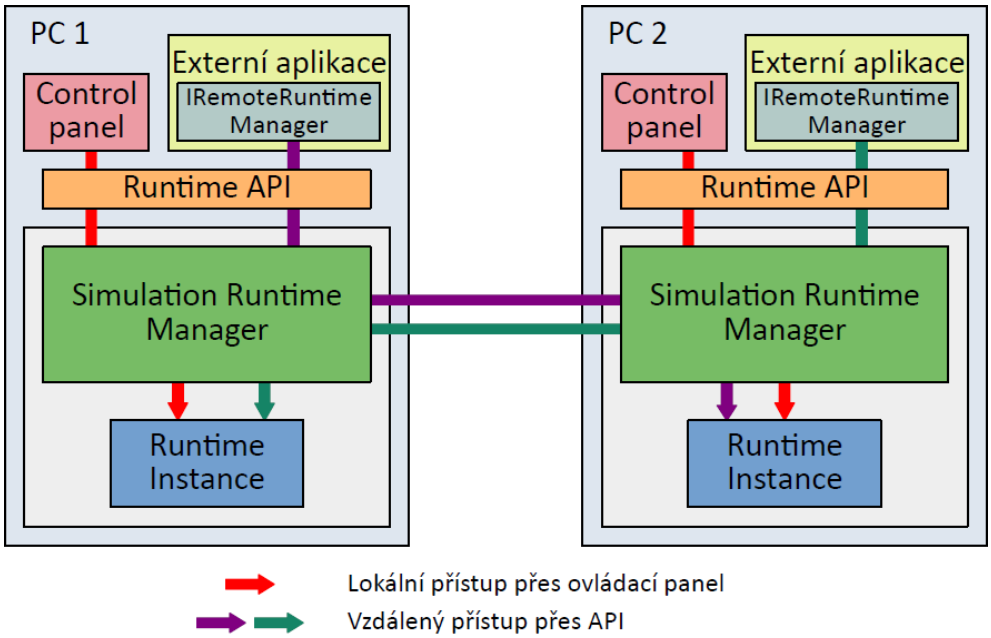


Obr. 2-12: Schéma přístupu externí aplikace k API

Jsou zde malé rozdíly pro aplikace v C++ a C#. V případě C# je rozhraní ISimulationRuntimeManager skryto (nelze ho implementovat), je však poskytnuta statická třída SimulationRuntimeManager, která toto rozhraní implementuje.

2.6.3 Přístup k PLC instancím

Přes Control panel lze přistupovat pouze k instancím, které jsou dostupné lokálně na stejném PC. Přes API lze však (při distribuované komunikaci), kromě místních instancí, přistupovat i k instancím, které běží na jiném PC. Schéma lokálního a vzdáleného přístupu k instancím je zobrazeno na Obr. 2-13.



Obr. 2-13: Lokální a vzdálený přístup k PLC instancím

Na Obr. 2-13 je naznačeno, jak může externí aplikace přistupovat k instancím, které jsou na jiném PC. Je nutné nejdříve získat přístup ke vzdálenému SRM z lokálního SRM, přes rozhraní IRemoteRuntimeManager (str. 42). Toto rozhraní lze získat pomocí funkce RemoteConnect (viz. část Vzdálený přístup na str. 39). Přes toto rozhraní lze potom ovládat vzdálený SRM a mimo jiné získat rozhraní IInstance (str. 39) k požadované vzdálené instanci pomocí funkce CreateInterface (viz. část Správa instancí na str. 42).

2.6.4 Rozhraní ISimulationRuntimeManager

Toto rozhraní poskytuje externí aplikaci přístup k SRM. Nabízí funkce pro správu instancí a jejich rozhraní a pro nastavení vzdáleného přístupu.

Samotné rozhraní je pro C# aplikace skryto. K jeho funkcím je nutno přistupovat přes statickou třídu SimulationRuntimeManager, která toto rozhraní implementuje. Nachází se ve jmenném prostoru Siemens.Simatic.Simulation.Runtime.

2.6.4.1 Důležité funkce

Správa instancí

- RegisteredInstanceInfo { get; } - vrací informace (jméno a ID) o všech aktuálně registrovaných instancích. Tyto informace jsou potřeba pro vytvoření rozhraní k již registrovaným instancím pomocí funkce CreateInterface().
- RegisterInstance() - vytvoření nové instance v SRM. Umožňuje definovat jméno instance a typ CPU a vrací vytvořené rozhraní k této instanci.
- CreateInterface() - vytvoří a vrátí rozhraní k již existující instanci. Požadovaná instance je definována pomocí jména nebo ID.

Vzdálený přístup

- OpenPort() - otevře port, přes který může k tomuto SRM vzdáleně přistupovat jiný SRM. Číslo portu musí být větší než 1024.
- RemoteConnect() - vytvoří nové spojení ke vzdálenému SRM a vrátí rozhraní IRemoteRuntimeManager.

2.6.4.2 Události

Rozhraní ISimulationRuntimeManager poskytuje informace o těchto událostech:

- OnConfigurationChanged - tato událost dává informaci o změně konfigurace SRM, např. když je registrována nová instance, odebrána instance nebo vytvořeno nově spojení se vzdáleným SRM.
- OnRuntimeManagerLost - informace o ztrátě spojení se vzdáleným SRM.

2.6.5 Rozhraní IInstance

Toto rozhraní nabízí funkce pro získání informací o instanci, konfiguraci a řízení instance a výměnu I/O dat s instancí virtuálního PLC.

Pro přístup k těmto funkcím je nutné nejprve získat rozhraní k požadované instanci pomocí funkce CreateInterface() nebo RegisterInstance(), které patří statické třídě SimulationRuntimeManager.

2.6.5.1 Důležité funkce

Informace o instanci a její nastavení

- Name { get; } – získání jména instance.
- CPUType { get; set; } – získání nebo nastavení typu CPU virtuálního PLC.
- CommunicationInterface { get; set; } – získání nebo nastavení komunikačního rozhraní.

- ControllerIPSuite4 { get; } – získání IP adresy, masky podsítě a výchozí brány.
- SetIPSuite() – nastavení IP adresy, masky podsítě a výchozí brány.
- OperatingState { get; } – operační stav virtuálního PLC.
- Dispose() – odstraní rozhraní k instanci (v SRM instance stále zůstane a je možné k ní znovu rozhraní vytvořit).
- UnregisterInstance() – odebere instanci ze SRM (pokud mají k této instanci vytvořené rozhraní jiné aplikace, tak toto rozhraní ztratí).

Ovládání PLC

- PowerOn() – vytvoří a spustí proces Runtime Instance a spustí firmware virtuálního PLC.
- PowerOff() – ukončí proces Runtime Instance.
- MemoryReset() – vypne virtuální PLC, ukončí jeho proces a provede restart.
- Run() – přepne virtuální PLC do stavu RUN.
- Stop() – přepne virtuální PLC do stavu STOP.

Tag list

- UpdateTagList() – načte tagy z virtuálního PLC a zapíše je do sdílené paměti. Maximální počet tagů je omezen na 500000 položek. Můžeme definovat (pomocí parametru funkce), jaký typ tagů se má načíst (např. pouze I/O).
- TagInfos { get; } – vrátí pole všech tagů v tag listu. Každá položka v poli obsahuje jméno tagu, typ tagu (I/O), datový typ, velikost v paměti, index atd. Pro aktuální informace je vhodné nejprve zavolat funkci UpdateTagList().

Synchronizování I/O (výměna dat)

Adresový přístup:

- InputArea { get; }, MarkerArea { get; }, OutputArea { get; } – vrátí rozhraní IIOArea, na kterém lze volat následující funkce pro adresní přístup k příslušné oblasti.
- AreaSize { get; } – vrátí velikost příslušné oblasti v bytech.
- ReadBit(), ReadByte(), ReadBytes(), ReadSignals() – čtení dat
- WriteBit(), WriteByte(), WriteBytes(), WriteSignals() – zápis dat

Symbolický přístup podle jména tagu:

- Read() – vrátí strukturu obsahující datový typ a hodnotu tagu
- ReadBool(), ReadInt16(), ReadInt32(), ReadFloat(), ReadDouble(), … – vrátí hodnotu tagu v požadovaném typu

- ReadSignals() – naplní a vrátí pole struktur, obsahující datový typ a hodnotu tagu, které bylo předtím předáno jako parametr funkce.
- Write() – zapíše strukturu obsahující datový typ a hodnotu tagu
- WriteBool(), WriteInt16(), WriteInt32(), WriteFloat(), WriteDouble(), … – zapíše hodnotu tagu v požadovaném typu
- WriteSignals() – zapíše více tagů najednou pomocí pole struktur, obsahující datový typ a hodnotu tagu

Nastavení virtuálního času

- SystemTime { get; set; } – vrátí nebo nastaví systémový čas virtuálního PLC.
- ScaleFactor { get; set; } - vrátí nebo nastaví časové měřítko virtuálního času. Je možné nastavit hodnotu v rozmezí 0,01 až 100.

Cyklus virtuálního PLC

- OperatingMode { get; set; } – vrátí nebo nastaví operační mód virtuálního PLC.
- IsSendSyncEventInDefaultModeEnabled { get; set; } – vrátí nebo nastaví příznak, který určuje, zda bude v operačním módu Default vyvolávána (při dosažení synchronizačního bodu) událost OnSyncPointReached.
- OverwrittenMinimalCycleTime_ns { get; set; } – vrátí nebo nastaví minimální čas cyklu, používaný v operačních módech SingleStep_CT and SingleStep_CPT.
- RunToNextSyncPoint() – virtuální PLC opustí stav „Freeze“ a pokračuje, dokud nedosáhne dalšího synchronizačního bodu. Používá se v „SingleStep“ operačních módech.
- StartProcessing() - virtuální PLC opustí stav „Freeze“ a pokračuje po nastavený čas, a poté pokračuje dál, dokud nedosáhne dalšího synchronizačního bodu. Používá se v „Timespan“ operačních módech.

2.6.5.2 Události

Rozhraní IInstance dále poskytuje informace o následujících událostech:

- OnOperatingStateChanged – změna operačního stavu virtuálního PLC. Obsahuje mimo jiné informaci předchozím a aktuálním operačním stavu.
- OnSyncPointReached – dosažení synchronizačního bodu.
- OnConfigurationChanging – probíhá změna konfigurace virtuálního PLC (náběh PLC po zapnutí napájení nebo začátek nahrávání do PLC).
- OnConfigurationChanged – dokončení změny konfigurace virtuálního PLC (po náběhu, nahrání nebo po změně IP adresy).

- OnLedChanged – změna stavu některé z indikačních LED diod. Obsahuje informace o tom, která LED byla změněna a do jakého stavu.

2.6.6 Rozhraní IRemoteRuntimeManager

Toto rozhraní poskytuje některé funkce vzdáleného SRM.

2.6.6.1 Důležité funkce

Vzdálené spojení

- IP { get; } – vrátí IP adresu PC, na kterém vzdálený SRM běží.
- Port { get; } – otevřený port vzdáleného SRM.
- RemoteComputerName { get; } – vrátí jméno PC, na kterém vzdálený SRM běží.
- Dispose() – odstraní rozhraní ke spojení se vzdáleným SRM. Spojení zůstane otevřeno a je možné k němu znovu vytvořit rozhraní.
- Disconnect() – uzavře spojení se vzdáleným SRM. Všechny aplikace, které jsou připojeny k tomuto vzdálenému SRM, ztratí toto spojení.

Správa instancí

- RegisteredInstanceInfo { get; } – vrátí informace o aktuálně registrovaných instancích vzdáleného SRM.
- RegisterInstance() - vytvoření nové instance ve vzdáleném SRM. Umožňuje definovat jméno instance a typ CPU a vrací vytvořené rozhraní k této instanci.
- CreateInterface() - vytvoří a vrátí rozhraní k již existující instanci vzdáleného SRM.

2.6.6.2 Události

Rozhraní IRemoteRuntimeManager poskytuje informace o následujících událostech:

- OnConnectionLost – aktivuje se, když je spojení se vzdáleným SRM ukončeno.

3 NÁVRH SIMULÁTORU

Tato kapitola bude věnována návrhu a vytvoření vlastní kosimulační aplikace pro simulaci výrobních linek s dopravníkovými systémy. Budou zde popsány možné přístupy, zdůvodněno vybrané řešení, rozebrán koncept a použité nástroje a podrobně popsán postup realizace simulátoru.

Pro simulaci nebude (mimo PLCSIM Advanced) použit žádný z hotových komplexních simulačních nástrojů, které byly představeny v první kapitole. To jednak z důvodu poměrně drahých licencí na tyto nástroje, ale hlavně proto, že jedním z cílů této diplomové práce je představení možností PLCSIM Advanced a jeho otevřeného API pro spolupráci s uživatelskými aplikacemi, psanými v jazyce C++ nebo C#.

3.1 Požadavky

Nejprve se zamysleme nad tím, co by měl simulátor mít a umět, aby byl prakticky použitelný. Samozřejmostí je schopnost výměny dat s virtuálním PLC (v tomto případě s instancemi PLCSIM Advanced). Simulátor musí reagovat na řídicí signály (výstupy) PLC, simulovat nějaké chování linky a dávat zpětnou vazbu na vstupy PLC. Určitě by měl mít uživatelské rozhraní (GUI) přes které bude možné ovládat simulaci, vytvářet objekty a navozovat poruchové stavy a nestandardní situace, atd. Dále by bylo vhodné mít nějakou vizualizaci pohybu materiálu po dopravnících, sepnutí čidel, aktivních pohonů atd. Také by bylo užitečné, aby simulátor nebyl jen na jedno použití (jedna linka), ale daly se v něm sestavovat vlastní dopravníkové linky, generovat další objekty (dopravníky, čidla, testovací materiál atd.) a upravovat jejich parametry a pozice. Zároveň by bylo dobré mít možnost vytvořenou linku uložit a znovu načíst po vypnutí a opětovném zapnutí simulátoru.

Shrnutí základních požadavků by tedy vypadalo takto:

- Komunikace a výměna dat s PLCSIM Advanced.
- Simulace chování linky – reakce na výstupy PLC a zpětná vazba na vstupy.
- Uživatelské rozhraní (GUI) pro ovládání simulace.
- Vizualizace simulované linky.
- Generování objektů - možnost sestavení a konfigurace vlastní linky.
- Save/Load systém.

3.2 Volba nástrojů

3.2.1 2D vs. 3D

Jedna ze základních otázek před návrhem simulátoru byla, jestli udělat jednoduchý simulátor ve dvojrozměrném prostředí (pomocí vykreslování jednoduchých geometrických tvarů), nebo ve trojrozměrném prostředí.

Volba 2D přístupu by znamenala použití nějakého vykreslovacího nástroje pro vizualizaci dopravníků, materiálu, čidel atd., který by byl řízen z vlastní aplikace (simulace chování linky), komunikující s PLCSIM Advanced. Vykreslovacím nástrojem by molo být např. WPF (Windows Presentation Foundation), což je framework pro tvorbu formulářových aplikací, který je součástí .NET frameworku a využívá značkovací jazyk XML. Další možností by bylo vykreslování 2D objektů ve formátu SVG (Scalable Vector Graphics) v aplikaci Inkscape, která rovněž nabízí interakci s objekty (např. kliknutí) a jejich přesouvání by zde bylo jednodušší.

Pro 3D simulaci by bylo možné využít Blender, což je open-source software pro modelování a vykreslování 3D počítačové grafiky a animací, nebo některý z dostupných herních enginů, jako je Unity3D nebo Unreal engine.

Simulace ve 2D by byla samozřejmě jednodušší, ale pak bych musel zvolit zobrazení buď z boku, kde by bylo omezení pouze na přímé výrobní linky s dopravníky seřazenými za sebou, nebo ze shora, kde by se zase těžko simuloval vertikální pohyb materiálu. Proto jsem nakonec zvolil 3D přístup s využitím herního enginu Unity3D, který obsahuje vykreslovací i fyzikální engine, což mi umožní snadno simulovat kolize, tření a gravitaci.

3.2.2 Unity3D 2018.3 Personal

Unity je multiplatformní herní engine vyvinutý společností Unity Technologies. Byl napsán v jazyce C++. Umožňuje vytvářet hry a interaktivní světy ve 3D i ve 2D. Obsahuje vlastní editor, vlastní fyzikální engine, audio engine a pro vykreslování podporuje knihovny (API) DirectX, OpenGL a WebGL. Samotné skriptování (programování chování hry) probíhá v C# a podporuje integraci vývojového prostředí MS Visual Studio. Umožňuje import 3D objektů z jiných nástrojů (např. z Blenderu). Ve verzi Personal je pro nekomerční účely je zdarma.

3.2.3 MS Visual Studio 2019 Community

Visual Studio je vývojové prostředí (IDE) od firmy Microsoft pro vývoj konzolových aplikací, formulářových aplikací, webových stránek, webových aplikací, Windows služeb atd. Obsahuje velké množství vývojových nástrojů a podporuje široké

spektrum programovacích jazyků. Lze ho použít pro psaní C# skriptů v Unity3D. Základní verze Community je dostupná zdarma.

3.2.4 C#

C# je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý a udržovaný firmou Microsoft jako součást .NET frameworku. Byl založen na jazycích Java a C++. Má dynamickou správu paměti, takže není nutno paměť ručně přidělovat a uvolňovat, nepodporuje vícenásobnou dědičnost, neexistují v něm globální proměnné a metody, vše musí patřit nějaké třídě (nahrazuje se statikou), je typově bezpečný a „case sensitive“.

3.3 Úvod do Unity3D

Znalost vývoje v tomto herního enginu není pro náš obor nijak přínosná ani zajímavá a předpokládám, že většina čtenářů této práce se s ním nesetkala, je však vhodné se zmínit o některých základních principech jeho fungování, aby čtenář lépe pochopil některé věci v části této diplomové práce, věnované realizaci samotné kosimulační aplikace.

3.3.1 Základní pojmy

3.3.1.1 Scéna

Scéna je vlastně prostředí (herní svět), kde se vše odehrává. Skládá se z jednotlivých herních objektů (GameObjects). V našem simulátoru bude pouze jedna scéna, která bude reprezentovat továrnu.

3.3.1.2 Herní objekt (GameObject)

Herní objekty jsou nejdůležitější součást projektu. Vlastně všechno, co se nachází ve scéně, jsou herní objekty (včetně kamery a světla). Dokonce i objekty uživatelského rozhraní (UI) jsou herní objekty. Tyto objekty jsou uspořádány do hierarchie, kde každý objekt může mít jeden nadřazený objekt (Parent) a neomezené množství podřízených objektů (Child). Pokud je změněna pozice, rotace nebo rozměr objektu, je stejně tak změněna u všech jeho podřízených objektů. Herní objekty se skládají z komponent.

3.3.1.3 Komponenty

Všechny herní objekty se skládají z komponent. Nejčastějšími komponentami jsou:

- Transform – toto je jediná komponenta, kterou musí obsahovat všechny herní objekty. Uchovává informace pozici, rotaci a rozměrech objektu.

- Mesh filter a Mesh Renderer – slouží pro renderování objektu. Pokud má být objekt ve scéně vidět, musí mít tyto komponenty.
- Collider – tato komponenta je potřeba, aby mohl objekt kolidovat s ostatními objekty, které mají Collider.
- Rigidbody – zajišťuje fyzikální chování objektu. Pouze na objekt, který obsahuje Rigidbody mohou působit fyzikální síly (setrvačnost, gravitace, …).
- Script – C# skript, přiřazený danému objektu, ve kterém je naprogramováno jeho chování. Musí obsahovat třídu, která dědí z třídy MonoBehaviour.

3.3.1.4 Inspector

Okno editoru, ve kterém lze přidávat a odebírat komponenty zvoleného herního objektu a upravovat jejich hodnoty.

3.3.1.5 Assets (aktiva)

Složka objektů, které se dají použít v projektu. Patří mezi ně například:

- Scény – všechny scény, které budou v projektu použity.
- Skripty – všechny (C#) skripty, které budou v projektu použity.
- Prefaby – předpřipravené 3D objekty (modely), které nejsou ve scéně hned při staru hry, ale mají být generovány v jejím průběhu (dalo by se říci, že jsou to něco jako třídy a během hry se tvoří jejich instance).
- Obrázky (textury) - např. soubory BMP, TIF, JPG atd.
- Zvukové (audio) soubory – např. MP3, OGG, WAV atd.
- Materiály – předpřipravené soubory vlastností jako jsou: barva, lesklost, koeficient tření, koeficient pružnosti atd., které lze přiřadit objektům.
- Pluginy – například DLL knihovny, které chceme použít v projektu.

3.3.2 Fyzikální engine

Unity3D obsahuje vlastní fyzikální engine, který poskytuje komponenty pro fyzikální simulaci v reálném čase. Umožňuje vytvářet pasivní objekty realistickým způsobem tak, že mohou být uvedeny do pohybu kolizemi s jinými objekty a jinými externími silami, jako je např. gravitace. Výpočty kolizí a reakcí na síly počítají se zadanými vlastnosti materiálů objektů, jako hmotnost, koeficient tření a tvrdost.

3.3.3 Skriptování

Skriptování je základní složkou všech Unity projektů. I nejjednodušší hra potřebuje skripty, aby mohla reagovat na vstupy od uživatele. Kromě toho mohou být skripty

použity pro vytváření grafických efektů, ovládání fyzického chování objektů nebo dokonce implementaci vlastního systému AI (umělé inteligence) pro objekty ve hře.

Pokud chceme naprogramovat chování nějakého herního objektu, musíme mu přidat komponentu Script, což je C# skript obsahující třídu, která dědí ze třídy MonoBehaviour.

3.3.3.1 Třída MonoBehaviour

Toto je základní třída pro všechny Unity skripty. Obsahuje všechny metody a události, které jsou dostupné standardním skriptům, připojeným k hernímu objektu. Poskytuje následující důležité události:

- Awake() –je zavolána pouze jednou před samotným načtením instance skriptu. Používá se pro inicializaci.
- Start() – je zavolána pouze jednou před prvním vykreslení objektu (pře prvním zavolání metody Update).
- Update() – je volána před každým vykreslením snímku objektu. Takže její frekvence je závislá na aktuálním FPS (Frame per Second).
- FixedUpdate() – volá se před každým výpočtem fyzikálního enginu, nezávisle na FPS, s konstantní frekvencí (defaultně jednou za 0,02 s).
- OnCollisionEnter() – zavolá se v okamžiku, kdy se objekt dotkne jiného (oba musí mít Collider).
- OnCollisionExit() – volá se v okamžiku, kdy se objekt přestane dotýkat jiného.
- OnCollisionStay() – volá se každý snímek, kdy se objekt dotýká jiného.
- OnMouseDown() – volá se při stisknutí tlačítka myši v okamžiku, kdy je ukazatel myši na herním objektu (z pohledu kamery).
- OnDestroy() – při zničení herního objektu (odstranění ze scény).

Třída obsahuje mnohem více událostí, metod a vlastností, všechny jsou popsány v oficiálním manuálu k Unity3D ([11]).

3.3.3.2 Třída Transform

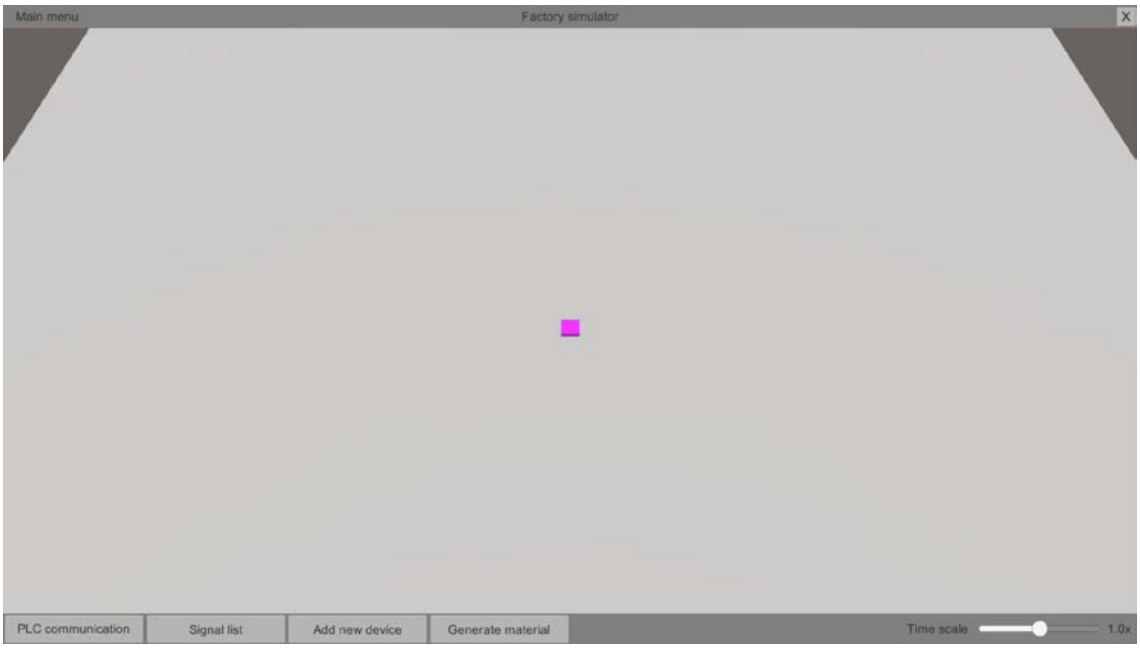
Každý herní objekt má určitou pozici, rotaci a rozměry, které jsou reprezentovány komponentou Transform. Tato komponenta má spoustu užitečných funkcí pro manipulaci s objektem, ke kterým lze přistupovat ze skriptu přes třídu Transform, kterou získáme jako vlastnost daného herního objektu.

3.4 Realizace simulačního prostředí

3.4.1 Vzhled aplikace po spuštění

3.4.1.1 Prázdná scéna (továrna)

Po spuštění kosimulační aplikace je zobrazeno prostředí, které reprezentuje prázdnou továrnu (halu). Ve scéně je tedy pouze rovná podlaha, která je osvětlena svrchu rovnoměrně po celé ploše. Vzhled simulátoru po spuštění ukazuje Obr. 3-1.



Obr. 3-1: Vzhled kosimulační aplikace po spuštění

Světlá (světle šedá) plocha je podlaha továrny, na které budeme stavět linku, tmavá (tmavě šedá) plocha okolo je prázdný prostor. Růžový kvádr značí místo, kde se bude generovat materiál. Může být pouze jeden.

3.4.1.2 Uživatelské rozhraní (UI)

Uživatelské rozhraní je součástí scény a obsahuje panely, ve kterých jsou seskupeny tlačítka pro ovládání funkcí simulátoru, textová pole, I/O textová pole a posuvníky.

Po spuštění aplikace jsou zobrazeny dva panely v horní a spodní části okna po celé jeho šířce.

Horní panel

Horní panel obsahuje název aplikace a tlačítka pro zobrazení hlavního menu a ukončení aplikace.

Obr. 3-2: UI - horní panel

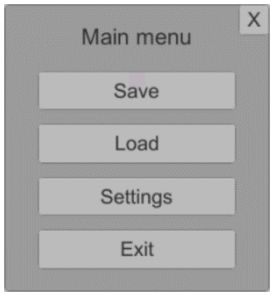
Dolní panel

Spodní panel obsahuje tlačítka pro zobrazení dalších panelů (PLC komunikace, seznamu signálů a přidání nového zařízení), tlačítko pro generování materiálu a posuvník pro změnu časového měřítka simulace, včetně popisu a aktuální hodnoty.

Obr. 3-3: UI – dolní panel

Hlavní menu

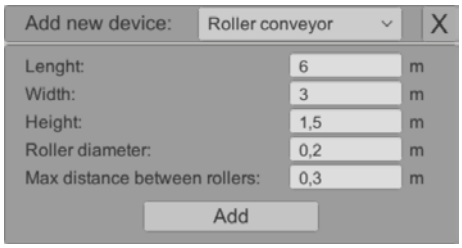
Po stisku tlačítka „Main menu“ na horním panelu, nebo po stisknutí klávesy Escape, se zobrazí hlavní menu (Obr. 3-4), ve kterém můžeme otevřít okno pro uložení aktuální konfigurace linky, otevřít okno pro načtení uložené konfigurace linky, otevřít nastavení a ukončit aplikaci.



Obr. 3-4: Vzhled hlavního menu

3.4.2 Přidání nového zařízení

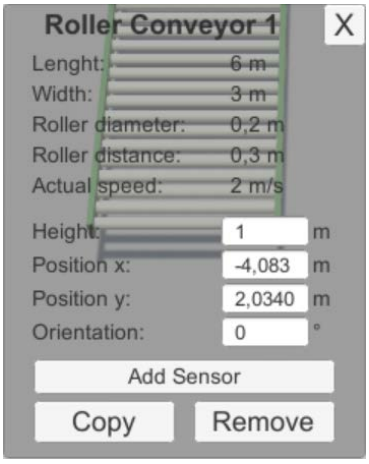
Nové zařízení lze přidat po stisku tlačítka „Add new device“ na dolním panelu. Otevře se nové okno (Obr. 3-5) pro výběr a konfiguraci zařízení, které chceme přidat.



Obr. 3-5: Okno pro přidání nového zařízení

V horní části tohoto okna je rozbalovací menu, kde vybereme, jaké zařízení chceme přidat (zatím je k dispozici pouze válečkový dopravník). Spodní část okna se přizpůsobí podle toho, jaké zařízení je vybráno, a můžeme zde zadat jeho rozměry. Stiskem tlačítka „Add“ se zařízení vygeneruje podle zadaných parametrů. Přidanému zařízení je automaticky přiřazeno jméno podle typu zařízení a jeho pořadí (např. první válečkový dopravník bude mít jméno „Roller conveyor 1“ atd.).

Nové zařízení se vygeneruje ve středu továrny a můžeme ho posunout na požadované místo táhnutím myši nebo po nebo po kliknutí na něj pravým tlačítkem myši se otevře panel, ve kterém můžeme upravit přímo jeho souřadnice a úhel natočení podle svislé osy (Obr. 3-6). Na tomto panelu jsou zároveň vidět vlastnosti zařízení (pro dopravník jsou to rozměry a aktuální rychlost) a obsahuje tlačítka pro vytvoření kopie zařízení, jeho odstranění a případně další (v případě dopravníku přidání senzoru).



Obr. 3-6: Panel válečkového dopravníku

Toto okno patří přímo k zařízení a posouvá se spolu s ním. Lze ho zavřít stisknutím křížku v pravém horním rohu panelu, nebo stejným způsobem jakým bylo otevřeno, kliknutím pravým tlačítkem myši na zařízení.

3.4.2.1 Válečkový dopravník

Pokud chceme přidat nový válečkový dopravník, musíme v okně „Add new device“ zadat jeho požadovanou délku, šířku, výšku, průměr válců a maximální rozestup mezi nimi.

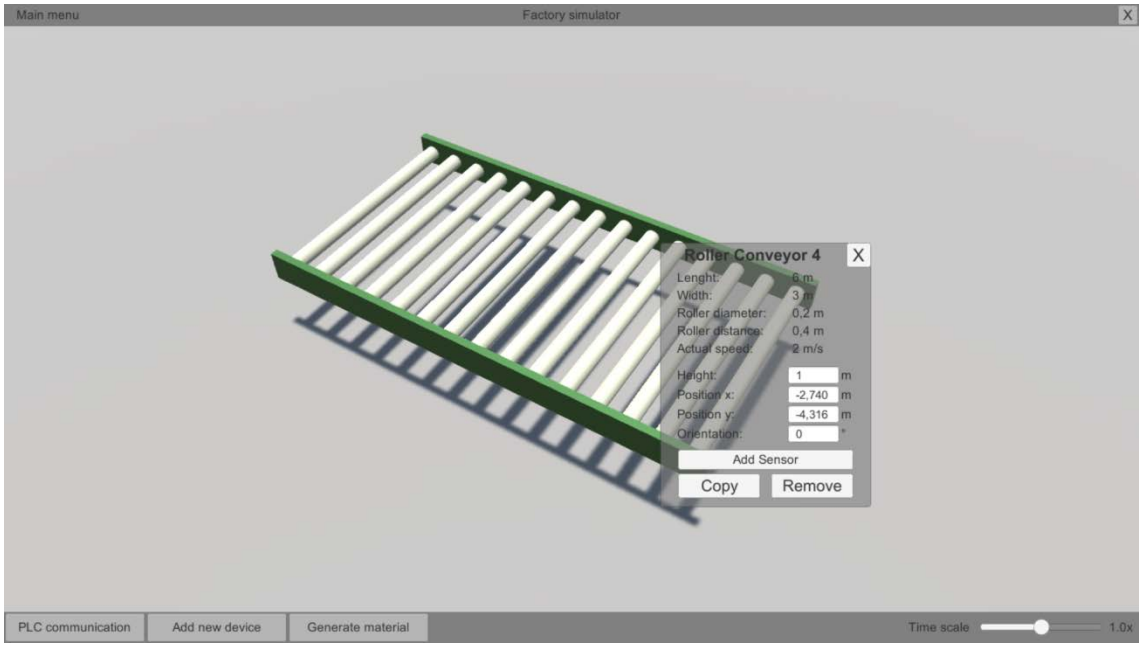
Dopravník se skládá z válců a konstrukce dopravníku (kryty uchycení válců). Tyto objekty jsou podřazeny prázdnému hernímu objektu „Roller conveyor“, který obsahuje pouze komponenty: Transform, skript „RollerConveyor“ (pro generování a ovládání dopravníku) a Collider, který se vytvoří podle velikosti dopravníku a

slouží pouze pro funkci Drag&Drop (nekoliduje s ostatními Collidery). Vygenerování dopravníku je prováděno pomocí následujícího úseku kódu:

```
public float Lenght { get; private set; } // délka dopravníku
public float Width { get; private set; } // šířka válců
public float Height { get; private set; } // výška dopravníků
public float MaxDistance { get; private set; } // maximální vzdálenost mezi osami válců
public float RollerDiameter { get; private set; } // průměr válců
List<GameObject> rollers = new List<GameObject>(); // list všech válců dopravníku

public void CreateConveyor()
{
    GenerateRollers(); // vygenerování válců
    CreateConstruction(); // vytvoření konstrukce dopravníku
    CreateDragCollider(); // vytvoření collideru pro funkcu Drag&Drop
    gameObject.SetActive(true); // aktivování dopravníku ve scéně
}

// vygenerování válců
void GenerateRollers()
{
    int numberOfRollers = Mathf.RoundToInt(Lenght / MaxDistance);  // počet válců dopravníku
    float distance = Lenght / numberOfRollers;   // vzdálenost mezi osami válců
    Vector3 rollerSize = new Vector3(RollerDiameter, Width / 2, RollerDiameter); //rozměry válců
    for (int i = 1; i <= numberOfRollers; i++)
    {
        Vector3 pos = transform.position + transform.forward * (i * distance - distance / 2); // pozice válce
        GenerateRoller(pos, rollerSize); // vytvoření válce
    }
}

// vytvoření válce
void GenerateRoller(Vector3 pos, Vector3 size)
{
    GameObject roller = Instantiate(rollerPrefab, pos, Quaternion.Euler(0, 0, 90), transform); // vytvoření válce
    roller.transform.localScale = size; // rozměry válce
    rollers.Add(roller); // přidání do seznamu válců
}

// vytvoření konstrukce dopravníku
void CreateConstruction()
{
    GameObject side1 = Instantiate(conveyorConstruct, transform); // nový GameObject podle předlohy
    side1.transform.localScale = new Vector3(0.1f, RollerDiameter + 0.2f, Lenght); // rozměry
    side1.transform.localPosition = new Vector3(Width / 2, 0, Lenght / 2); // pozice
    GameObject side2 = Instantiate(conveyorConstruct, transform); // nový GameObject podle předlohy
    side2.transform.localScale = new Vector3(0.1f, RollerDiameter + 0.2f, Lenght); // rozměry
    side2.transform.localPosition = new Vector3(-Width / 2, 0, Lenght / 2); // pozice
}

// vytvoření collideru pro funkci Drag&Drop
void CreateDragCollider()
{
    BoxCollider col = gameObject.AddComponent<BoxCollider>(); // přidání komponenty BoxCollider
    col.size = new Vector3(Width, RollerDiameter, Lenght); // rozměry Collideru
    col.center = new Vector3(0, 0, Lenght / 2); // pozice Collideru
    col.isTrigger = true; // nebude kolidovat s ostatními Collidery
}
```

V kódu výše uvedeném stojí za zmínku funkce Instantiate, která vytvoří (a vrátí na něj referenci) nový herní objekt podle předlohy (Prefab), která je předána jako první parametr funkce. Posledním (volitelným) parametrem funkce je reference na komponentu Transform objektu, který má být nadřazen nově vygenerovanému objektu. Prostředními (volitelnými) parametry jsou pozice a rotace objektu.

Na Obr. 3-7 je zobrazen vygenerovaný dopravník s rozměry: délka 6 m, šířka 3 m, výška 1,5 m, průměr válců 0,2 m a maximálními rozestupy mezi válci 0,4 m.



Obr. 3-7: Vygenerovaný dopravník

Vytvořený dopravník nemá nohy ani jiné podpěry a jen „visí“ v prostoru, nepůsobí na něj totiž gravitace (lze nastavit ve vlastnostech komponenty Rigidbody). Generování noh může být samozřejmě doděláno, ale přišlo mi to nyní bezpředmětné a při změně výšky dopravníku by se musela měnit (přepočítávat) délka a pozice nohou.

Každý válec má přiřazen skript „Roller“, který obsahuje metodu „Rotate“, která zajišťuje otáčení válce zadanou rychlostí (viz. následující kód).

```
Rigidbody rigid; // proměnná typu Rigidbody
Quaternion deltaRot; // přírůstek rotace
Vector3 angleVel; // úhlová rychlost

void Awake()
{
    rigid = GetComponent<Rigidbody>(); // získání reference na komponentu Rigidbody
}

void RotateRoller(float speed)
{
    angleVel = new Vector3(0, -speed, 0); // úhlová rychlost
    deltaRot = Quaternion.Euler(angleVel * Time.deltaTime); // přírůstek rotace
    rigid.MoveRotation(transform.rotation * deltaRot); // rotace o úhel
}
```

Tato metoda je volána ze skriptu „RollerConveyor“, který je přiřazen dopravníku, pro všechny válce dopravníku. Metodu nějakého objektu můžeme zavolat z jiného objektu tak, že mu pošleme zprávu s názvem požadované metody, pomocí funkce SendMessage (viz. následující kód). Další možností by bylo získat referenci na skript, ve kterém je metoda obsažena, a zavolat metodu přímo (metoda musí být veřejná, tedy s modifikátorem public).

```
void FixedUpdate()
{
   if (rollers.Count > 0 && rollers[0] != null) // pokud není seznam válců dopravníku prázdný…
   {
      foreach (GameObject roller in rollers) // … tak pro každy válec v seznamu…
      {
         roller.SendMessage("RotateRoller", speed); // … zavolej funkci “RotateRoller”
      }
   }
}
```

Pohyb materiálu po dopravníku zde funguje (stejně jako v reálném světě) na principu tření mezi ním a válci. Na materiál zároveň působí gravitace. Není zde tedy psán žádný kód pro pohyb materiálu (pouze pro otáčení válců), o všechno se stará fyzikální engine Unity.

3.4.2.2 Přidání optického senzoru

Optický senzor nemůže existovat samostatně, ale je vytvořen vždy pro konkrétní dopravník. Přidáme ho kliknutím na tlačítko „Add sensor“ na panelu dopravníku. Senzor se vytvoří tak, že vysílač a přijímač jsou umístěny na konstrukci dopravníku naproti sobě, ve vzdálenosti podle šířky dopravníku. Můžeme s ním hýbat po celé délce dopravníku táhnutím myši, nebo na panelu senzoru, který se zobrazí po kliknutí pravým tlačítkem myši na vysílač senzoru. Počet senzorů na dopravník není omezen.

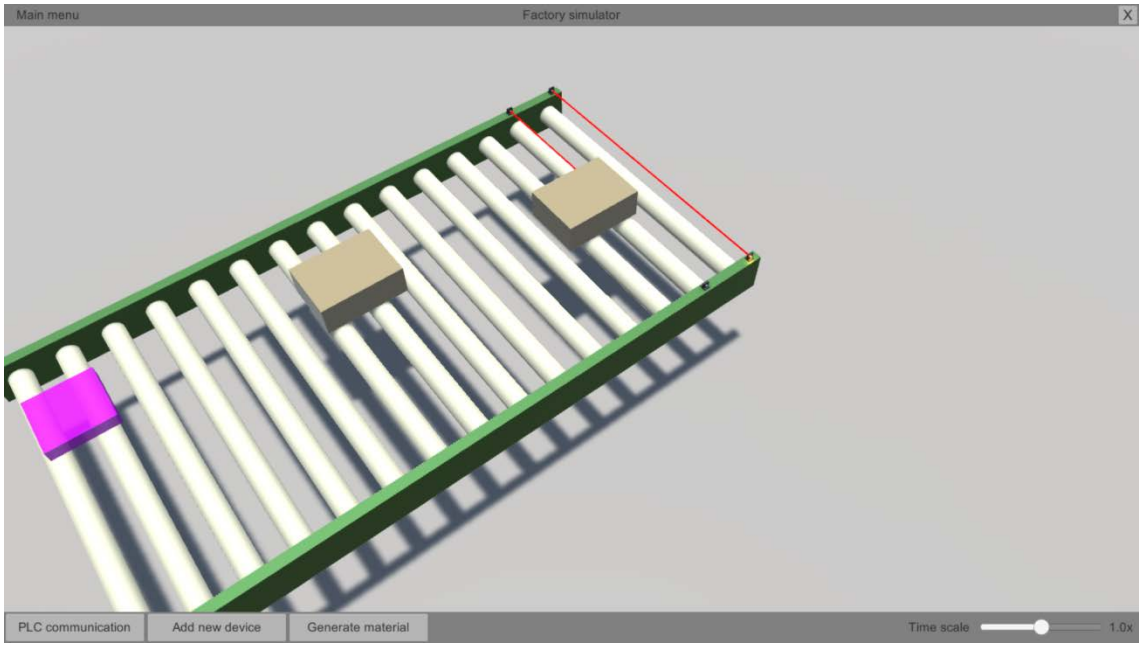
Senzor obsahuje objekty vysílač (Transmitter) a přijímač (Reciever). Z vysílače je směrem k přijímači vyslán paprsek (Ray) a vyhodnocuje se, jestli zasažený objekt je přijímač, nebo jiný objekt (materiál).

Vyhodnocení kolize paprsku a vykreslení čáry se provádí v následujícím úseku kódu ve třídě Laser:

```
LineRenderer line; // vykreslování čáry
Vector3 endPoint; // koncový bod, ke kterému bude čára vykreslena
public float Range { get; set; } // max vzdálenost, do jaké vysílat paprsek
public bool notInterrupted; // optická závora není narušena

void Awake()
{
   line = GetComponent<LineRenderer>(); // získání reference na komponentu pro vykreslování čáry
   endPoint = line.GetPosition(1); // koncový bod = bod čáry 1 (počáteční bod čáry by byl bod 0)
   Range = endPoint.x;
}

void Update()
{
   endPoint = transform.right * Range; // souřadnice koncového bodu podle dosahu
   if (Physics.Raycast(transform.position, endPoint, out RaycastHit hit, Range))
   {
      endPoint = transform.InverseTransformPoint(hit.point); // koncový bod = bod kolize
      if (hit.collider.gameObject.CompareTag("Reciever")) // pokud byl trefen přijímač...
      {
         notInterrupted = true; // závora nepřerušena
      }
      else // jinak...
         notInterrupted = false; // závora přerušena
   }
   line.SetPosition(1, endPoint); // vykreslení čáry ke koncovému bodu
}
```

Na kódu ukázaném výše stojí za zmínku funkce Raycast, která je součástí statické třídy UnityEngine.Physics. Tato funkce vyšle neviditelný paprsek z počátečního bodu (transform.position – aktuální pozice objektu) směrem ke koncovému bodu (endPoint) a v parametru typu RaycastHit vrací informace o tom, co paprsek trefil (jaký objekt, souřadnice bodu atd.).

Na Obr. 3-8 jsou dva vygenerované optické senzory na dopravníku, z nichž jeden je přerušen jiným objektem (materiálem) a druhý je nepřerušen.



Obr. 3-8: Ukázka přerušené a nepřerušené optické závory

Na obrázku výše můžeme vidět, že u přerušené závory je čára vykreslena pouze k místu kolize s objektem. Zároveň si můžeme všimnout, že u nepřerušené závory svítí na přijímači oranžové světlo.

3.4.3 Generování materiálu

Výrobní materiál je zde vygenerován (jeden kus) po kliknutí na tlačítko „Generate material“ na dolním panelu, nebo po stisknutí mezerníku na klávesnici. Je vytvořen na místě zvaném „Spawn place“, což je herní objekt, který je ve scéně reprezentován průhledným růžovým kvádrem. S tímto objektem lze hýbat táhnutím myši, nebo po kliknutí na něj pravým tlačítkem myši se otevře okno, ve kterém můžeme upravit přímo jeho souřadnice (včetně výšky). Ve scéně je vždy jen jeden „Spawn place“. Vygenerované kusy materiálu jsou jediné objekty ve scéně, na které působí gravitace. Můžeme s nimi také posouvat pomocí myši.

3.4.4 Kamera

Kamera je zde také herní objekt, takže jí lze přidat skript, ve kterém bude naprogramován její pohyb. Pomocí šipek (nebo kláves WSAD) na klávesnici pohybujeme kamerou ve vodorovné rovině (v konstantní výšce). Dále při stisku klávesy CTRL nebo pravého tlačítka myši natáčíme pohled kamery pohybem myši. Kolečkem myši poté posouváme kameru dopředu a dozadu ve směru pohledu.

3.4.5 Save/Load systém

Ukládání a načítání stavu scény je velice komplikovaná a složitá procedura. Nelze pouze jednoduchou funkcí „otisknout“ aktuální stav někam do paměti (souboru) a poté otisk jednoduše znovu přetvořit na objekty. Nejlepší způsob ukládání stavu scény do souboru je serializace (převedení do proudu jedniček a nul). Serializovat lze však pouze elementární datové typy, nikoliv herní objekty nebo jejich komponenty a ani třídy. Pro uložení stavu scény je tedy potřeba nalézt všechny potřebné objekty, každý z nich rozložit na komponenty a ty potom rozložit na jejich vlastnosti v podobě elementárních datových typů.

Ukládat se budou všechny objekty, které byly přidány do scény od startu aplikace. Všechny tyto objekty byly vytvořeny podle své předlohy (Prefab). Každá z těchto předloh obsahuje skript „Object Identifier“. Po startu aplikace jsou všechny předlohy (ze složky Assets/Prefabs) načteny do slovníku (Prefab Dictionary). Postup pro uložení stavu scény je následující:

1. Nalézt všechny objekty, které mají komponentu „Object Identifier“.
2. Každému z nich přiřadit unikátní ID a ID objektu, který je mu nadřazen.
3. Zavolat metodu „OnSerialize“ (provedení úkonů, které by měly být vykonány před serializací objektu) na každém objektu, který ji obsahuje.
4. Pro každý objekt vytvořit instanci třídy „SceneObject“, která bude uchovávat serializovatelná data objektu, která jsou nutná pro opětovné vytvoření objektu.
5. Všechny komponenty objektů rozebrat na data (elementární datové typy), která lze serializovat (např. u komponenty Transform získat jednotlivé XYZ souřadnice pozice, hodnoty rotace v jednotlivých osách a rozměry). V případě skriptů uložit hodnoty všech třídních proměnných a vlastností, které jsou serializovatelné a nejsou označeny atributem [DontSaveField]. Poté pro každou komponentu vytvořit třídu „ObjectComponent“, a do ní všechna tato data uložit, včetně typu komponenty.
6. Do „SceneObject“ uložit tyto informace o objektu:
    - Jméno objektu.
    - Jméno předlohy (Prefab) podle které byl vytvořen.

- ID objektu.
- ID nadřazeného objektu (nebo null, pokud nemá nadřazený objekt).
- Informace o tom, zda je objekt ve scéně aktivní.
- Seznam komponent a jejich data ve formě „ObjectComponent“.

7. Vytvořit instanci třídy „SceneData“ a uložit do ní serializovatelná data o všech objektech ve formě třídy „SceneObject“.
8. Serializovat „SceneData“ do binárního souboru.

Postup opětovného vytvoření uložené scény:

1. Vyčistit scénu – odstranit všechny objekty, které byly přidány až po startu aplikace.
2. Deserializovat „SceneData“ z binárního souboru.
3. Pro každý „SceneObject“:
    - je podle uloženého jména předlohy vybrána odpovídající předloha ze slovníku (Prefab Dictionary) a podle ní je vytvořen nový herní objekt pomocí funkce Instantiate.
    - Novému objektu jsou přiděleny uložená data: jméno objektu, ID, ID nadřazeného objektu, hodnoty komponenty Transform a informace, zda je ve scéně aktivní.
    - Následně jsou komponentám nového objektu přiřazeny uložené hodnoty.
4. Projít všechny nově vytvořené herní objekty: Každý, který má nenulové ID nadřazeného objektu, je přiřazen pod příslušný objekt.
5. Zavolat metodu „OnDeserialize“ (provedení úkonů, které by měly být vykonány po deserializaci objektu) na každém novém objektu, který ji obsahuje.

3.4.6 Komunikace s PLCSIM Advanced

Toto je pro automatizačního inženýra nejzajímavější část - samotná komunikace kosimulační aplikace s PLCSIM Advanced a jeho ovládání přes API. Zároveň to byla také největší výzva této práce, protože jsem zde narazil na nepříjemný problém.

3.4.6.1 Použití API funkcí v C# aplikacích

Pokud chceme využít funkce API PLCSIM Advanced ve vlastní C# aplikaci, je to jednoduché. Stačí ve Visual Studiu přidat odkaz (referenci) na danou API knihovnu skrz příkaz „Projekt -> Přidat odkaz…“. Pokud je naše aplikace cílena pro 32-bit systémy, přidáme odkaz na knihovnu Siemens.Simatic.Simulation.Runtime.Api.x86.dll, pro 64-bit systémy odkážeme na Siemens.Simatic.Simulation.Runtime.Api.x64.dll. Poté

už jen stačí na začátek každého projektového C# souboru, ve kterém chceme používat API funkce, přidat řádek:

```
using Siemens.Simatic.Simulation.Runtime;
```

a máme k dispozici všechna rozhraní a datové typy API knihovny.

3.4.6.2 Problém

Skripty pro Unity jsou psány v C# a ve Visual Studiu je pro tento účel automaticky vytvořen projekt s názvem „Assemly-CSharp“. Původně jsem proto předpokládal, že bude možné pouze přidat referenci na Siemens DLL knihovnu a budu moci používat funkce API, stejně jako u klasických C# aplikací. To však bohužel není možné a příkaz „Projekt -> Přidat odkaz…“ zde ani není vůbec dostupný. Je to z důvodu, že tento C# projekt není vlastně klasická C# aplikace, ale spíš soubor C# skriptů, které si poté Unity zkompiluje vlastním způsobem.

Unity však nabízí způsob, jak použít externí DLL knihovny v projektu, a to pomocí zásuvných modulů (pluginů). Rozděluje pluginy na dva typy: „Managed“ (psané v řízeném kódu – C#) a „Native“ (psané v nativním kódu – C++). Nejběžnějšími pluginy jsou DLL knihovny, ale můžou být i jiného formátu. Pluginy je nutno nakopírovat do Unity projektu do složky Assets/Plugins. Použití řízených (Managed) pluginů je velmi jednoduché, po nakopírování do složky se na něj automaticky vytvoří reference v C# projektu a můžeme hned používat jeho funkce. U nativních pluginů je to složitější. Reference se na ně přidat nelze a je nutné všechny požadované funkce nejprve zvlášť definovat ve formátu:

```
[DLLImport („jméno pluginu“)]
private static extern float NázevFunkce();
```

a až poté je možné funkci použít. Je to však možné pouze pokud byla tato funkce před kompilací knihovny v C++ kódu označena jako extern „C“.

Po nakopírování Siemens knihovny do Unity projektu mě ovšem čekalo další nepříjemné překvapení. Unity „vidí“ tuto knihovnu jako nativní plugin a nevytvoří na ni referenci do C# projektu. Bohužel je to asi proto, že tato knihovna obsahuje jak rozhraní pro C# aplikace, tak pro C++ aplikace, a byl tedy použit C++ kompilátor. Použití DLLImport funkce však nepřipadá v úvahu, protože některé funkce vracejí datové typy, které jsou rovněž definované v té knihovně a potřebujeme využívat i rozhraní a události knihovny.

Tento problém jsem se pokusil obejít tím, že jsem napsal vlastní C# DLL knihovnu, ve které jsem odkazoval na původní Siemens knihovnu. Všechna rozhraní a datové typy jsem napodobil ve své knihovně a pro všechny API funkce jsem vytvořil metody ve své knihovně, ve kterých je zavolána odpovídající metoda Siemens knihovny. Tudíž by mělo být možné využívat funkce Siemens knihovny z Unity nepřímo přes moji knihovnu, která je již čistě C# knihovna, a mělo by být možné ji přidat do Unity jako řízený plugin. Po nakopírování mojí DLL knihovny

do Unity projektu na ni byla automaticky vytvořena reference do C# projektu a mohl jsem potřebné funkce v programu využívat. Vše se tedy zdálo v pořádku, dokud jsem Unity projekt nezkusil zkompilovat. Kompilace byla ukončena s chybou, že součást Siemens.Simatic.Simulation.Runtime.Api.x64.dll, odkazovaná mnou přidaným řízeným pluginem, nelze načíst. Ani toto řešení tedy nebylo úspěšné.

Řešení

Po několika neúspěšných pokusech jsem došel k závěru, že nejrozumnější řešení bude naprogramovat C# aplikaci (službu), která bude přes API komunikovat s PLCSIM Advanced a bude si vyměňovat potřebná data s kosimulační aplikací přes paměť počítače. Tato aplikace se bude jmenovat PLCSIMInterface a bude spouštěná jako proces z kosimulační aplikace.

3.4.7 PLCSIMInterface

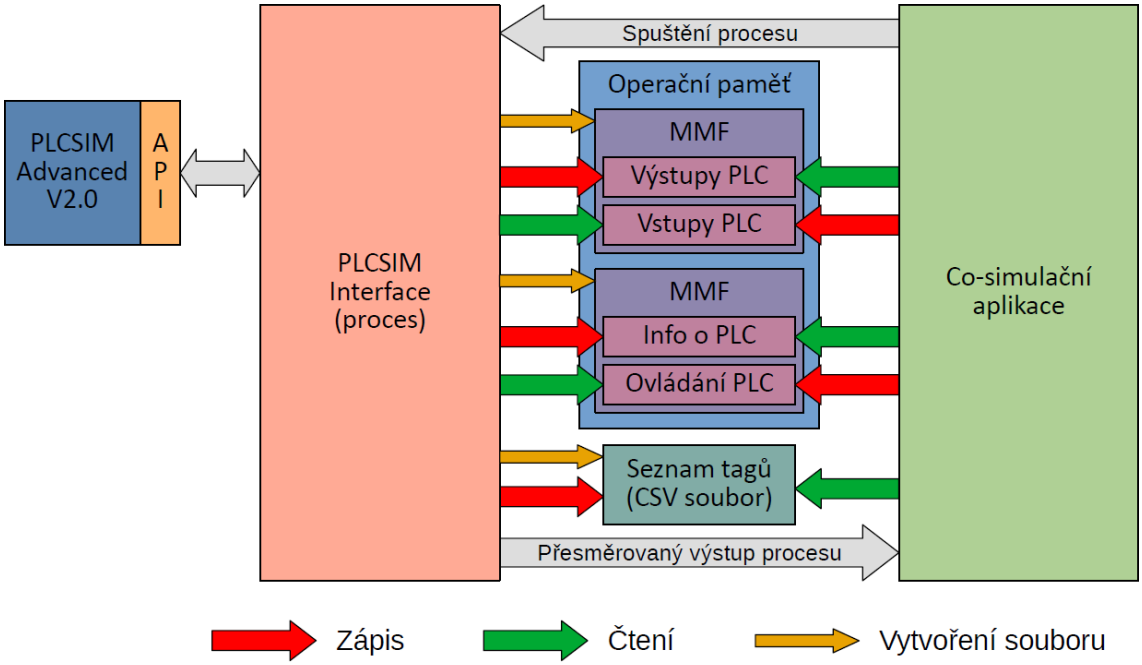
PLCSIMInterface je konzolová C# aplikace (proces), která bude zajišťovat výměnu dat mezi kosimulační aplikací a PLCSIM Advanced.

3.4.7.1 Princip výměny dat

Pro výměnu dat mezi dvěma aplikacemi je nutné využít nějaké místo v paměti počítače. V našem případě bude poměrně vysoký nárok na výkon, protože bude potřeba zapisovat a číst data každý cyklus virtuálního PLC (řádově desítky ms). Proto jsem se rozhodl, že využiji techniku mapování souborů do paměti (MMF – Memory-mapped file). Tyto MMF soubory budou dva, jeden pro výměnu informací o stavu PLC a jeho ovládání, a druhý pro výměnu hodnot vstupů a výstupů PLC (tagů). Pro zabránění situaci, kdy obě aplikace pracují se souborem současně, jsem použil funkci Mutex (mutual exclusion), což je synchronizační mechanizmus pro zajištění výhradního přístupu k systémovým prostředkům.

Samotný seznam tagů (Tag list) je uložen do CSV souboru na počítač do složky aplikačních dat (…\AppData\Roaming\PLCSIM Cosimulation). Tento soubor obsahuje jméno, datový typ, typ vstup/výstup a velikost datového typu v bytech každého tagu. Tyto informace se mění pouze při přehrání PLC a bylo by zbytečné je posílat každý cyklus přes operační paměť PC. Kosimulační aplikace si tento seznam tagů načte ze souboru pouze tehdy, když dostane od PLCSIMInterface informaci, že byl změněn.

Na Obr. 3-9 je znázorněno schéma výměny dat mezi kosimulační aplikací a procesem PLCSIMInterface.



Obr. 3-9: Schéma výměny dat mezi PLCSIMInterface a kosim aplikací

Ve zkratce: kosimulační aplikace spustí proces PLCSIMInterface a ten vytvoří dva MMF soubory a jeden CSV soubor, přes které bude probíhat výměna dat. Některé informace, na které by měla kosimulační aplikace reagovat asynchronně, jsou předávány přes přesměrovaný výstup procesu.

3.4.7.2 Spuštění procesu

Proces je spuštěn z kosimulační aplikace po vytvoření nového PLC. Zároveň je procesu předáno jméno PLC instance, nastaveno skrytí konzole a přesměrován jeho výstup (bude ho zachytávat kosimulační aplikace místo toho, aby byl vypsán do konzole). Pro každé přidané PLC je vytvořen vlastní proces. Spustitelný EXE soubor procesu je vhodné umístit spolu s používanou Siemens API knihovnou do Unity projektu do složky Assets/StreamingAssets. Spuštění procesu z kosimulační aplikace je provedeno pomocí následujícího kódu:

```
private void StartDataExchangeService()
{
    ProcessStartInfo startInfo = new ProcessStartInfo(Path.Combine(Application.streamingAssetsPath,
"PLCSIMInterface.exe"))
    {
        CreateNoWindow = true, // nespouštět proces v novém okně
        UseShellExecute = false, // vytvořit proces přímo z EXE souboru
        RedirectStandardInput = true, // přesměrovat vstup procesu
        RedirectStandardOutput = true, // přesměrovat výstup procesu
        RedirectStandardError = true, // přesměrovat chybové hlášení procesu
        Arguments = Name + " " + minCycleTime_ms // předání argumentů procesu
    };

    try
    {
        process = Process.Start(startInfo); // spuštění procesu s vytvořeným nastavením
        UnityEngine.Debug.Log("PLCSIM Advanced Interface started");
```

```
    }
    catch (Exception ex)
    {
        UnityEngine.Debug.LogWarning(ex.Message); // vypsání hlášky dané výjimky
        UnityEngine.Debug.LogWarning("PLCSIM Advanced Interface can not be started");
    }

    process.OutputDataReceived += Process_OutputDataReceived; // metoda volaná při zachycení výstupu procesu
    process.BeginOutputReadLine(); // začni zachytávat výstup procesu
    process.ErrorDataReceived += Process_ErrorDataRecieved; // metoda volaná při zachycení chyby procesu
    process.BeginErrorReadLine(); // začni zachytávat chybové hlášky procesu
}
```

3.4.7.3 Vytvoření rozhraní k PLC instanci

Abychom mohli ovládat instanci PLCSIM Advanced z naší aplikace, musíme k této instanci získat rozhraní. Toto rozhraní vytvoříme pomocí statické třídy SimulationRuntimeManager. Kód funkce pro získání rozhraní:

```
private IInstance CreateInstanceInterface(string instanceName)
{
    IInstance instance; // rozhraní instance
    try
    {
        instance = SimulationRuntimeManager.CreateInterface(instanceName); // vytvoření rozhraní k instanci
        Console.WriteLine("Interface to PLCSIM instance '" + instanceName + "' was created");
    }
    catch (SimulationRuntimeException ex)
    {
        if (ex.RuntimeErrorCode == ERuntimeErrorCode.DoesNotExist) // pokud daná instance neexistuje…
        {
            instance = SimulationRuntimeManager.RegisterInstance(instanceName); // …vytvoření nové instance
            Console.WriteLine("PLCSIM instance '" + instanceName + "' was registered");
        }
        else
        {
            instance = null;
            Console.WriteLine(ex);
        }
    }
    return instance;
}
```

Funkce nejprve zkusí vytvořit rozhraní k instanci. Pokud program zachytí výjimku SimulationRuntimeException s kódem DoesNotExist, znamená to, že žádná instance se zadaným jménem není registrována a zaregistruje novou instanci pomocí funkce RegisterInstance, která k ní zároveň vytvoří a vrátí rozhraní.

3.4.7.4 Vytvoření MMF souborů

Mapování souborů do paměti je alternativní práce se soubory, kdy je jejich obsah promítnut jádrem operačního systému do virtuální paměti, která je namapována do operační paměti počítače. Obsah souboru je tak možné zapisovat a číst z operační paměti běžnými strojovými instrukcemi pro práci s pamětí, což je velice rychlé.

MMF pro výměnu informací o stavu PLC a jeho ovládání

Proces po spuštění vytvoří tento soubor, namapuje ho do operační paměti a pošle na výstup zprávu o vytvoření souboru. Kosimulační aplikace zachytí zprávu a vytvoří si k souboru přístup, nyní může instanci ovládat a číst její stav.

Velikost MMF souboru, přes který jsou předávány data pro ovládání a informace o PLC, je 100 bytů. Čtení/zápis do tohoto souboru probíhá cyklicky každých 100 ms z obou stran. Strukturu souboru ukazuje Tabulka 4.

Tabulka 4: Struktura MMF souboru pro ovládání a informace o PLC

Byte	Datový typ	Informace/příkaz
0	int	rezerva
4	byte	operační stav PLC
5	byte	změna konfigurace PLC
6	byte	rezerva
7	byte	rezerva
8	int	min. čas cyklu PLC (ms)
12	int	počítadlo cyklů PLC
16	int	doba chodu PLC (s)
20	4x byte	IP adresa PLC
24	4x byte	maska podsítě
28	4x byte	výchozí brána
32	byte	stav LED diody STOP
33	byte	stav LED diody RUN
34	byte	stav LED diody ERROR
35	byte	stav LED diody BF (Bus Fault)
36-49		rezervy
50	float	časové měřítko simulace
54	bool	zapnutí napájení PLC
55	bool	vypnutí PLC
56	bool	přepnout PLC do RUN
57	bool	přepnout PLC do STOP
58	bool	RESET paměti PLC
59	bool	rezerva
60	bool	rezerva
61	bool	nastavit IP adresu
62	4x byte	IP adresa PLC
66	4x byte	maska podsítě
70	4x byte	výchozí brána
71-99		rezervy

Přístup do tohoto souboru je řešen adresově po bytech. Hodnota datového typu bool zabírá v paměti jeden byte. Na adresách 0 až 49 jsou informace o stavu PLC, proces tam zapisuje, kosimulační aplikace čte. Na adresy 50 až 99 naopak zapisuje aplikace, ale v případě příkazů i proces. Např. pro zapnutí PLC zapíše kosimulační aplikace na adresu 54 hodnotu „true“ (bool), proces po přečtení této adresy vykoná zapnutí PLC a zapíše na tuto adresu hodnotu „false“.

MMF pro výměnu hodnot PLC vstupů a výstupů

Tento MMF soubor je vytvořen odpovídající velikosti seznamu tagů a pošle na výstup zprávu o vytvoření. Kosimulační aplikace si načte seznam tagů a vytvoří přístup k MMF souboru.

Velikost tohoto MMF souboru závisí na počtu tagů a velikosti jejich datových typů. Pokud je seznam tagů změněn, je aktuální MMF zahozen, je vytvořen nový podle aktuálního seznamu tagů a na výstup je poslána zpráva, že byl vytvořen nový MMF soubor. Kosimulační aplikace zachytí zprávu a vytvoří přístup k novému MMF souboru. Kód pro vytvoření MMF:

```
Mutex tagValuesMutex; // mutex pro přístup k MMF souboru
MemoryMappedFile tagValuesMemory; // MMF soubor
MemoryMappedViewAccessor tagValuesAccessor; // přístup k MMF souboru

public void CreateTagValuesSharedMemory()
{
    if (tagList == null || tagList.Length == 0) // pokud je seznam tagů prázdný...
        return;

    tagValuesMutex.WaitOne(); // počká na uvolnění zdroje a pak ho zablokuje pro sebe

    int capacity = 0;
    // výpočet velikosti MMF souboru podle počtu a velikostí tagů
    foreach (STagInfo tag in tagList)
    {
        capacity += tag.Size;
    }

    if (tagValuesMemory != null) // pokud je již soubor vytvořen...
        tagValuesMemory.Dispose(); // uvolni veškeré jeho zdroje

    tagValuesMemory = MemoryMappedFile.CreateNew("tagValues-" + instance.Name, capacity);
    tagValuesAccessor = tagValuesMemory.CreateViewAccessor(); // vytvoření přístupu k MMF souboru

    tagValuesMutex.ReleaseMutex(); // uvolnění zdroje

    Console.WriteLine("Shared memory created - tagValues"); // poslání zprávy cosimulační aplikaci
}
```

3.4.7.5 Vytvoření a aktualizace seznamu tagů

Poté, co je do PLC nahrán program, proces získá z instance seznam tagů, podle něj vytvoří (nebo přepíše) CSV soubor. Kód funkce pro aktualizaci seznamu tagů:

```
STagInfo[] tagList;

private void UpdateTagList()
{
    if (instance == null) // pokud není interface vytvořen...
        return;

    instance.UpdateTagList(ETagListDetails.IO); // aktualizace seznamu tagů (PLC vstupů a výstupů)
    tagList = instance.TagInfos; // získání seznamu tagů

    if (tagList.Length == 0) // pokud je seznam tagů prázdný...
    {
        Console.WriteLine("Tag list is empty");
        return;
    }

    try
    {
        appDataPath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "PLCSIM
CoSimulation");
        if (!Directory.Exists(appDataPath)) // pokud složka neexistuje...
        {
            Directory.CreateDirectory(appDataPath); // ...vytvoř složku
        }
    }
    catch (Exception ex)
    {
```

```
        Console.WriteLine(ex.Message);
    }

    try
    {
        using (StreamWriter sw = new StreamWriter(Path.Combine(appDataPath, "TagList.csv")))
        {
            foreach (STagInfo tag in tagList)
            {
                string[] fields = {
                    tag.Name, // jméno tagu
                    ((ushort)tag.Area).ToString(), // typ (vstup/výstup)
                    ((ushort)tag.DataType).ToString(), // datový typ v PLC
                    ((ushort)tag.PrimitiveDataType).ToString(), // PC datový typ
                    tag.Size.ToString() // velikost datového typu v paměti (v bytech)
                };
                string line = string.Join(";", fields); // oddělení vlastností tagu středníkem
                sw.WriteLine(line); // zápis řádku s informacemi o tagu do CSV souboru
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    Console.WriteLine("Tag list was updated"); // poslání zprávy cosimulační aplikaci
}
```

Volbou parametru ETagListDetails.IO funkce UpdateTagList je zajištěno, že seznam tagů bude obsahovat pouze PLC vstupy a výstupy. Výčtový typ ETagListDetails nabízí i jiné volby. Může obsahovat ještě merkery (M), datové bloky (DB), čítače (C) nebo časovače (T). Možné jsou všechny jejich kombinace.

3.4.7.6 Reakce na události PLC

Po získání rozhraní k instanci je třeba zaregistrovat PLC události, na které se bude reagovat, a přiřadit jim metody.

```
private void RegisterInstanceEvents(int cycleTime_ms)
{
    instance.RegisterOnSyncPointReachedEvent(); // registrace události OnSyncPointReached
    instance.RegisterOnOperatingStateChangedEvent(); // registrace události OnOperatingStateChanged
    instance.RegisterOnConfigurationChangedEvent(); // registrace události OnConfigurationChanged
    instance.RegisterOnLedChangedEvent(); // registrace události OnLedChanged

    instance.OperatingMode = EOperatingMode.SingleStep_CT; // volba operačního módu
    instance.OverwrittenMinimalCycleTime_ns = cycleTime_ms * 1000000; // nastavení minimální doby cyklu

    instance.OnSyncPointReached += OnOnSyncPointReached; // reakce na dosažení synchronizačního bodu
    instance.OnConfigurationChanged += OnOnConfigurationChanged; // reakce na změnu konfigurace
    instance.OnOperatingStateChanged += OnOperatingStateChanged; // reakce na změnu operačního stavu
    instance.OnLedChanged += OnLedChanged; // reakce na změnu stavu LED diod
}
```

Dále je ukázána metoda OnOnConfigurationChanged, která se volá v reakci na změnu konfigurace PLC. Jedním z parametrů je výčtový typ EInstanceConfigChanged, který nese informaci o tom, zda byla změněna HW/SW konfigurace, nebo IP adresa.

```
private void OnOnConfigurationChanged(IInstance sender, ERuntimeErrorCode errorCode, DateTime dateTime,
EInstanceConfigChanged instanceConfigChanged, uint param1, uint param2, uint param3, uint param4)
{
    if (instanceConfigChanged == EInstanceConfigChanged.HardwareSoftwareChanged) // pokud byla změněna HW/SW
konfigurace...
    {
        UpdateTagList(); // ...aktualizuj seznam tagů,...
        CreateTagValuesSharedMemory(); // ...vytvoř nový MMF soubor...
        Console.WriteLine("Configuration changed"); // …a pošli zprávu cosimulační aplikaci
```

```
    }
    else if (instanceConfigChanged == EInstanceConfigChanged.IPChanged) // pokud byla změněna IP adresa...
    {
        Console.WriteLine("IP changed"); // ...pošli zprávu cosimulační aplikaci
    }
}
```

Metoda OnOnSyncPointReached je volána v reakci na dosažení synchronizačního bodu. Zde je zavolána metoda UpdateTagValues pro aktualizaci hodnot tagů, tedy přečtení hodnot vstupů a zapsání výstupů PLC:

```
private void OnOnSyncPointReached(IInstance sender, ERuntimeErrorCode errorCode, DateTime dateTime, uint pipID,
long timeSinceSameSyncPoint_ns, long timeSinceAnySyncPoint_ns, uint syncPointCount)
{
    cycleCounter++; // počítadlo počtu cyklů
    plcLifeTime_s = (float)(dateTime - startTime).TotalSeconds; // doba běhu PLC
    UpdateTagValues(); // přečtení hodnot vstupů a zapsání výstupů PLC
    instance.RunToNextSyncPoint(); // pokračování k dalšímu synchronizačnímu bodu
}
```

3.4.7.7 Ovládání PLC instance

Proces cyklicky čte hodnoty adres v MMF souboru, které reprezentují příkazy z kosimulační aplikace pro ovládání a nastavení PLC. Zároveň do toho samého MMF souboru zapíše hodnoty reprezentující informace o stavu PLC na příslušné adresy. To je prováděno v cyklicky volané metodě UpdateStatus.

```
public void UpdateStatus(object state)
{
    if (plcInfoAccessor == null) // pokud není přístup k MMF vytvořen...
        return;

    plcInfoMutex.WaitOne(); // počká na uvolnění zdroje a pak ho zablokuje pro sebe
    // čtení z MMF
    instance.ScaleFactor = plcInfoAccessor.ReadSingle(50); // nastavení časového měřítka instance
    bool powerOnPLC = plcInfoAccessor.ReadBoolean(54);
    bool powerOffPLC = plcInfoAccessor.ReadBoolean(55);
    bool runPLC = plcInfoAccessor.ReadBoolean(56);
    bool stopPLC = plcInfoAccessor.ReadBoolean(57);
    bool memoryReset = plcInfoAccessor.ReadBoolean(58);
    bool setIP = plcInfoAccessor.ReadBoolean(61);
    // zápis zo MMF
    plcInfoAccessor.Write(4, (byte)instance.OperatingState);
    plcInfoAccessor.Write(5, plcConfigChanged);
    plcInfoAccessor.Write(8, (int)(instance.OverwrittenMinimalCycleTime_ns / 1000000));
    plcInfoAccessor.Write(12, cycleCounter);
    plcInfoAccessor.Write(16, plcLifeTime_s);

    plcInfoMutex.ReleaseMutex(); // uvolnění zdroje

    if (powerOnPLC) InstancePowerOn(); // zapnutí PLC
    if (powerOffPLC) InstancePowerOff(); // vypnutí PLC
    if (runPLC) InstanceRun(); // přepnout PLC do RUN
    if (stopPLC) InstanceStop(); // přepnout PLC do STOP
    if (memoryReset) InstanceMRES(); // RESET paměti PLC
    if (setIP) SetIPSuite(); // nastavení IP adresy PLC
}
```

Metoda pro zapnutí PLC:

```
public void InstancePowerOn()
{
    if (instance == null) // pokud není interface vytvořen...
        return;

    if (instance.OperatingState == EOperatingState.Off) // pokud je PLC vypnuté...
    {
        Console.WriteLine("The instance " + instance.Name + " is powering on...");
```

```
        try
        {
            instance.PowerOn(60000); // zapnutí PLC
        }
        catch (SimulationRuntimeException ex)
        {
            Console.WriteLine(ex.Message);
        }
    }

    if (plcInfoAccessor != null) // pokud je přístup k MMF vytvořen...
    {
        plcInfoMutex.WaitOne();
        plcInfoAccessor.Write(54, false);
        plcInfoMutex.ReleaseMutex();
    }
}
```

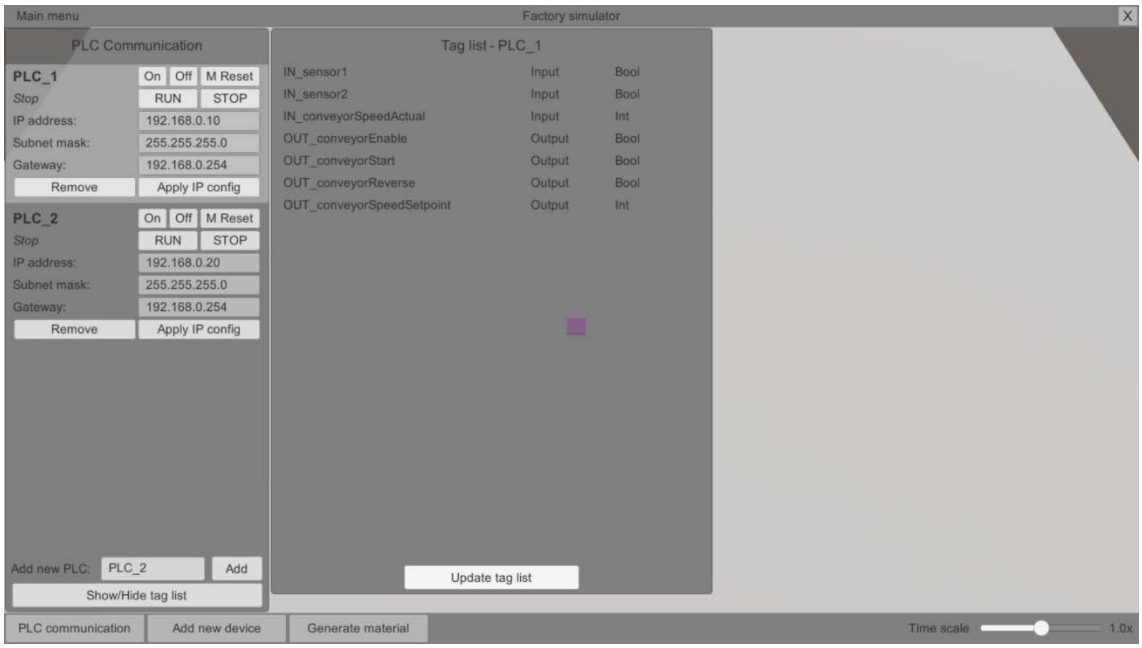
Nastavení IP konfigurace PLC instance:

```
public void SetIPSuite()
{
    if (plcInfoAccessor == null) // pokud není přístup k MMF vytvořen...
        return;
    // nastavení komunikačního rozhraní PLC instance na TCP/IP
    instance.CommunicationInterface = ECommunicationInterface.TCPIP;

    byte[] ipAddress = new byte[4]; // nové pole pro IP adresu
    byte[] subnetMask = new byte[4]; // nové pole pro masku podsítě
    byte[] defaultGateway = new byte[4]; // nové pole pro výchozí bránu

    plcInfoMutex.WaitOne();
    ipAddress[0] = plcInfoAccessor.ReadByte(62);
    ipAddress[1] = plcInfoAccessor.ReadByte(63);
    ipAddress[2] = plcInfoAccessor.ReadByte(64);
    ipAddress[3] = plcInfoAccessor.ReadByte(65);
    subnetMask[0] = plcInfoAccessor.ReadByte(66);
    subnetMask[1] = plcInfoAccessor.ReadByte(67);
    subnetMask[2] = plcInfoAccessor.ReadByte(68);
    subnetMask[3] = plcInfoAccessor.ReadByte(69);
    defaultGateway[0] = plcInfoAccessor.ReadByte(70);
    defaultGateway[1] = plcInfoAccessor.ReadByte(71);
    defaultGateway[2] = plcInfoAccessor.ReadByte(72);
    defaultGateway[3] = plcInfoAccessor.ReadByte(73);
    plcInfoAccessor.Write(61, false);
    plcInfoMutex.ReleaseMutex();
    // nová IP konfigurace
    SIPSuite4 IPSuite = new SIPSuite4
    {
        IPAddress = new SIP { IPArray = ipAddress },
        SubnetMask = new SIP { IPArray = subnetMask },
        DefaultGateway = new SIP { IPArray = defaultGateway }
    };

    try
    {
        instance.SetIPSuite(0, IPSuite, true); // nastavení IP konfigurace PLC instance
    }
    catch (SimulationRuntimeException ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

3.4.8 Přidání PLC

Po kliknutí na tlačítko „PLC communication“ se otevře okno, které slouží pro správu a ovládání všech PLC v simulaci. Přidání PLC se provede tak, že ve spodní části okna je potřeba napsat jméno PLC a kliknout na tlačítko „Add“. Pokud je v PLCSIM

Advanced již zaregistrována instance s tímto jménem, je s ní vytvořeno spojení, pokud není, je vytvořena nová instance. Každé přidané PLC má na panelu PLC komunikace vlastní ovládací panel, přes který je možné PLC ovládat a nastavovat jeho IP konfiguraci.



Obr. 3-10: Panel PLC komunikace

Na obrázku výše je zobrazen panel PLC komunikace, se dvěma přidanýma PLC, a seznam tagů, který patří prvnímu PLC (PLC_1). Každé PLC má vlastní seznam tagů, který lze zobrazit nebo skrýt stiskem tlačítka „Show/Hide tag list“ na panelu PLC komunikace. Je zobrazen vždy pouze jeden seznam tagů, patřící tomu PLC, který je aktuálně vybrán (na jehož ovládací panel bylo kliknuto naposled).

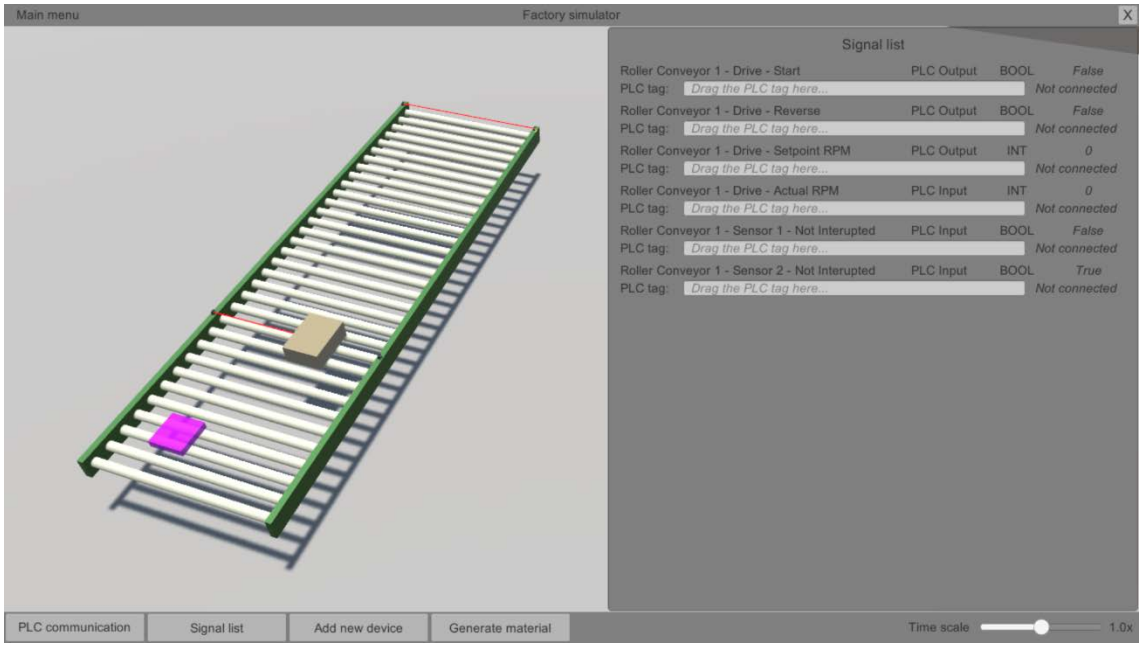
Obr. 3-11: Ovládací panel PLC

Ovládací panel PLC (na Obr. 3-11) tedy obsahuje jméno PLC (tučně), aktuální operační stav (kurzívou), tlačítka pro zapnutí/vypnutí PLC, přepnutí do stavu RUN/STOP, reset paměti PLC, odebrání PLC a nastavení IP konfigurace. Pro nastavení IP konfigurace musí být PLC v zapnutém stavu, tedy nesmí být ve stavu OFF. Po kliknutí na tlačítko „Remove“ je zrušeno spojení s PLC a odebrán jeho

ovládací panel a seznam tagů, v PLCSIM Advanced však instance zůstane a je k ní možno opět vytvořit spojení.

3.4.9 Mapování signálů

Mapováním signálů je myšleno přiřazení PLC vstupů a výstupů I/O signálům linky. Po vytvoření zařízení jsou automaticky vytvořeny signály elektrických komponent, které dané zařízení obsahuje. Signály všech zařízení v lince mohou být zobrazeny v seznamu signálů (Obr. 3-12), který lze zobrazit po kliknutí na tlačítko „Signal list“ na dolním panelu.



Obr. 3-12: Seznam signálů

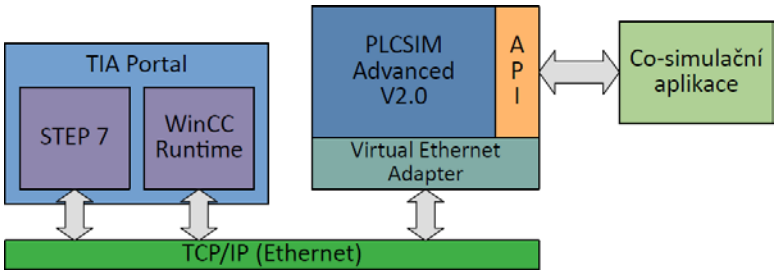
V seznamu signálů je každý signál reprezentován grafickým objektem, rozděleným na dvě poloviny (řádky). Na horním řádku vidíme jeho název, typ (vstup/výstup), datový typ a aktuální hodnotu. Název signálu se skládá z názvu zařízení, jeho elektrické části a samotného jména signálu. Ve spodní části vidíme pole pro přiřazení PLC tagu a informaci o tom, jestli je signál s PLC tagem propojený. PLC tag můžeme přiřadit tak, že otevřeme seznam tagů příslušného PLC a z něj přetáhneme požadovaný tag na příslušný signál v seznamu tagů stylem Drag&Drop. Jméno PLC tagu se potom automaticky objeví v příslušném poli pro přiřazení tagu. PLC tag musí být stejného typu (vstup/výstup, datový typ), jako signál, kterému ho chceme přiřadit, jinak se přiřazení neprovede.

Obr. 3-13: Ukázka propojeného a nepropojeného signálu

Na Obr. 3-13 vidíme, jak vypadá propojený a nepropojený signál v seznamu signálů. U propojeného signálu je v poli pro přiřazení PLC tagu před názvem samotného tagu napsáno také jméno PLC, kterému tag patří. To je z toho důvodu, že více PLC může obsahovat tag se stejným názvem, a také pro rychlejší orientaci v případě více PLC.

4 SIMULAČNÍ ÚLOHA

V této kapitole bude v simulátoru vytvořena a otestována jednoduchá dopravníková linka, v TIA Portalu bude napsán PLC program (STEP 7), který bude linku řídit, a vytvořena vizualizace (WinCC), přes kterou bude možné linku ovládat.

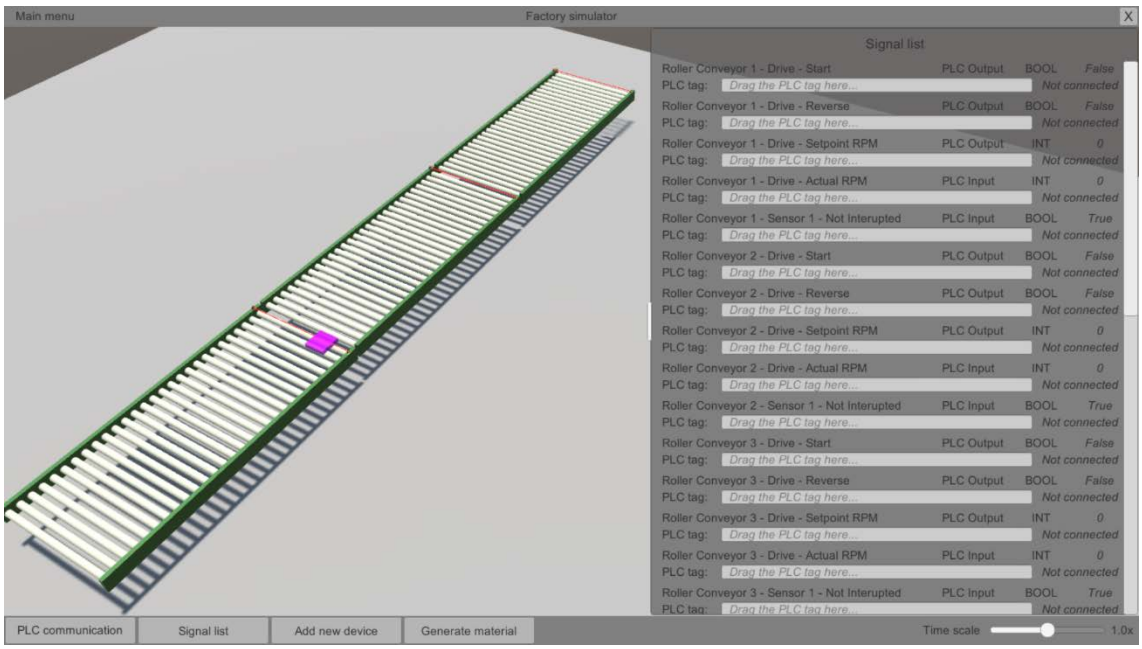


Obr. 4-1: Schéma komunikace nástrojů simulační úlohy

Na Obr. 4-1 je schéma komunikace mezi nástroji v simulační úloze. STEP 7 a vizualizační runtime komunikují s PLCSIM Advanced přes TCP/IP (lokálně) a kosimulační aplikace komunikuje s PLCSIM Advanced přes jeho API.

4.1 Vytvoření dopravníkové linky

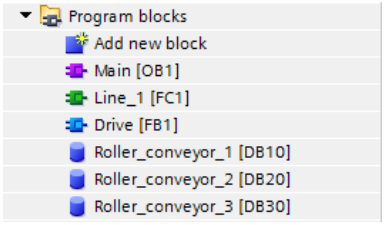
Pro demonstraci a otestování simulátoru jsem vytvořil jednoduchou přímou dopravníkovou linku se třemi stejnými dopravníky uspořádanými za sebou. Každému dopravníku jsem přidal jeden optický senzor. Na Obr. 4-2 můžeme vidět, jak linka vypadá a seznam všech signálů z/do linky.



Obr. 4-2: Vytvořená dopravníková linka a její signály

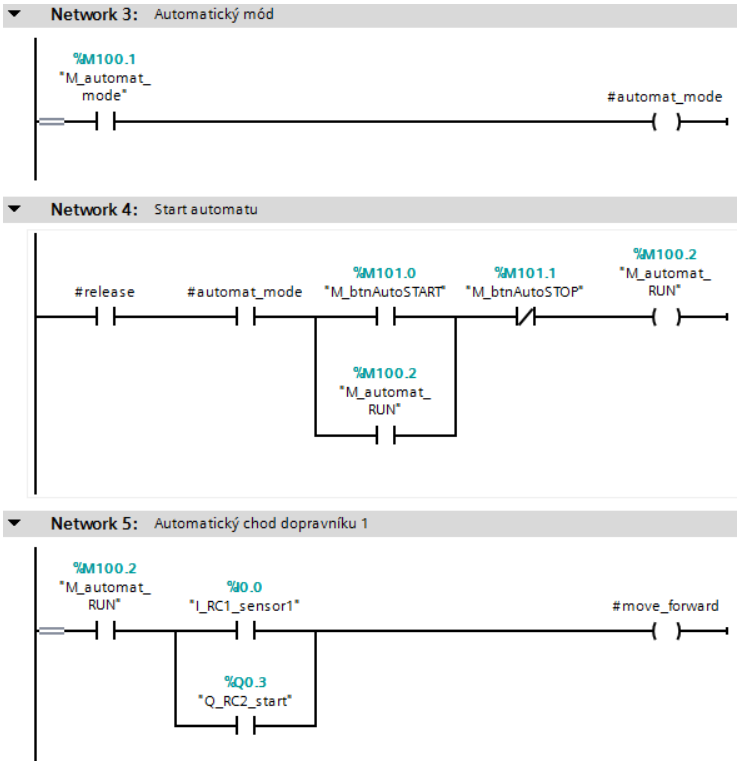
4.2 Vytvoření PLC programu ve STEP 7 (TIA Portal)

V TIA Portalu jsem vytvořil jednoduchý program pro manuální i automatické řízení. Řízení linky probíhá v bloku (funkci) „Line_1“ (FC1), který je volán z hlavního cyklického bloku „Main“ (OB1). Pohony dopravníků jsou ovládány ve funkčním bloku „Drive“ (FB1), přičemž pro každý dopravník (pohon) je vytvořena jedna instance tohoto bloku (DB10, DB20, DB30).



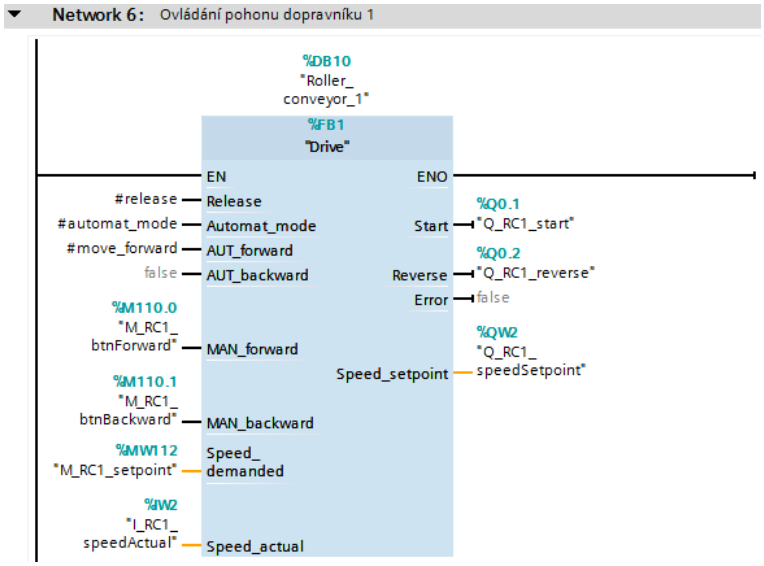
Obr. 4-3: Programové bloky TIA projektu

Samotnou logiku programu jsem tvořil v PLC programovacím jazyku LAD (Ladder Diagram). Tento jednoduchý jazyk je podobný reléovému schématu a díky tomu je čitelný i pro elektro údržbáře zařízení. Je vhodný pro většinu logického PLC řízení, pro sekvenční řízení a složitější výpočty jsou vhodnější jiné jazyky.



Obr. 4-4: Část programu funkce FC1 v jazyu LAD

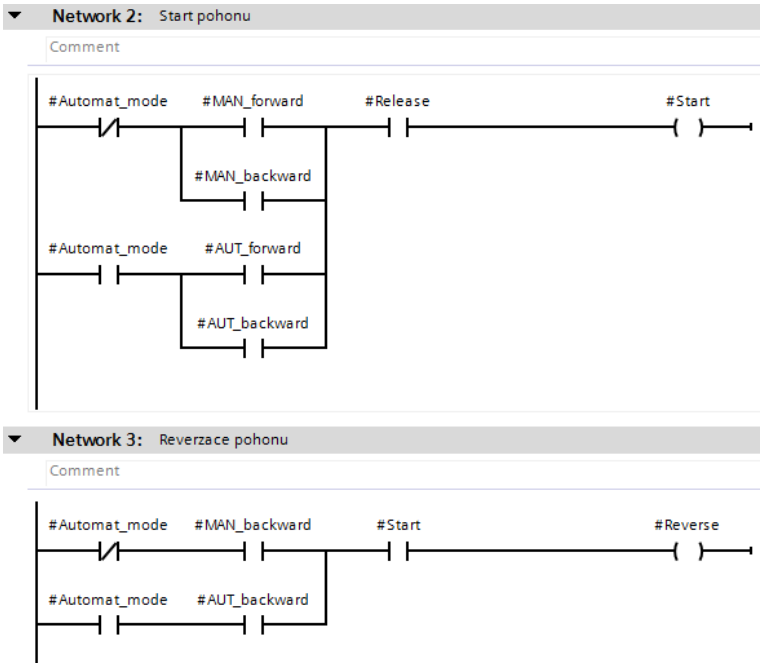
Na Obr. 4-4 je část funkce „Line_1“ pro řízení linky. Merker „M_automat_mode“ je ovládán tlačítky na vizualizaci a nese informaci o tom, zda je aktivní automatický nebo manuální mód. Pokud je automatický mód spuštěn, můžeme pomocí dalších tlačítek na vizualizaci („M_btnAutoSTART“ a „M_btnAutoSTOP“) odstartovat nebo zastavit automatické řízení linky. V automatickém režimu jsou dopravníky řízeny pomocí proměnné „#move_forward“, která je v logické „1“, pokud není senzor na konci dopravníku přerušen („log 1“ – funguje jako optická závora) nebo další dopravník v pořadí (po směru toku materiálu) běží.



Obr. 4-5: Funkční blok FB1 pro ovládání pohonu dopravníku

Na Obr. 4-5 je pokračování funkce FC1, konkrétně volání funkčního bloku „Drive“ pro ovládání pohonu prvního dopravníku. Merkery „M_RC1_btnForward“ a „M_RC1_btnBackward“ jsou nastaveny tlačítky z vizualizace a slouží pro pohyb dopravníku v manuálním režimu. Merker „M_RC1_setpoint“ je požadovaná rychlost pohonu, nastavovaná opět z vizualizace.

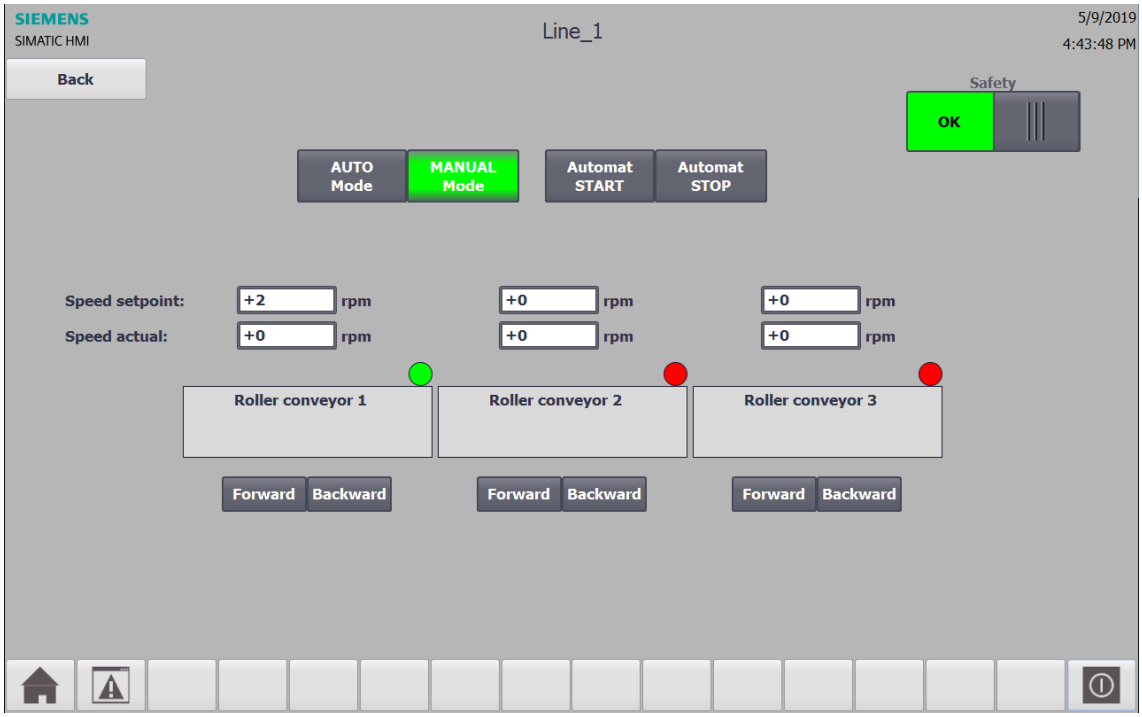
Logika uvnitř tohoto bloku, konkrétně pro výstupní signály pro start a reverzaci pohonu, je ukázána na Obr. 4-6. Pro pohyb dopravníku dopředu musí být aktivní pouze signál „Start“, pro zpětný chod musí být aktivní oba signály.



Obr. 4-6: Část bloku FB1 pro start a reverzaci pohonu

4.3 Vytvoření vizualizace ve WinCC (TIA Portal)

Ve WinCC jsem vytvořil vizualizaci pro HMI panel TP1500 Comfort, který ovšem nebudu potřebovat, protože vytvořený runtime lze spustit i na klasickém PC.

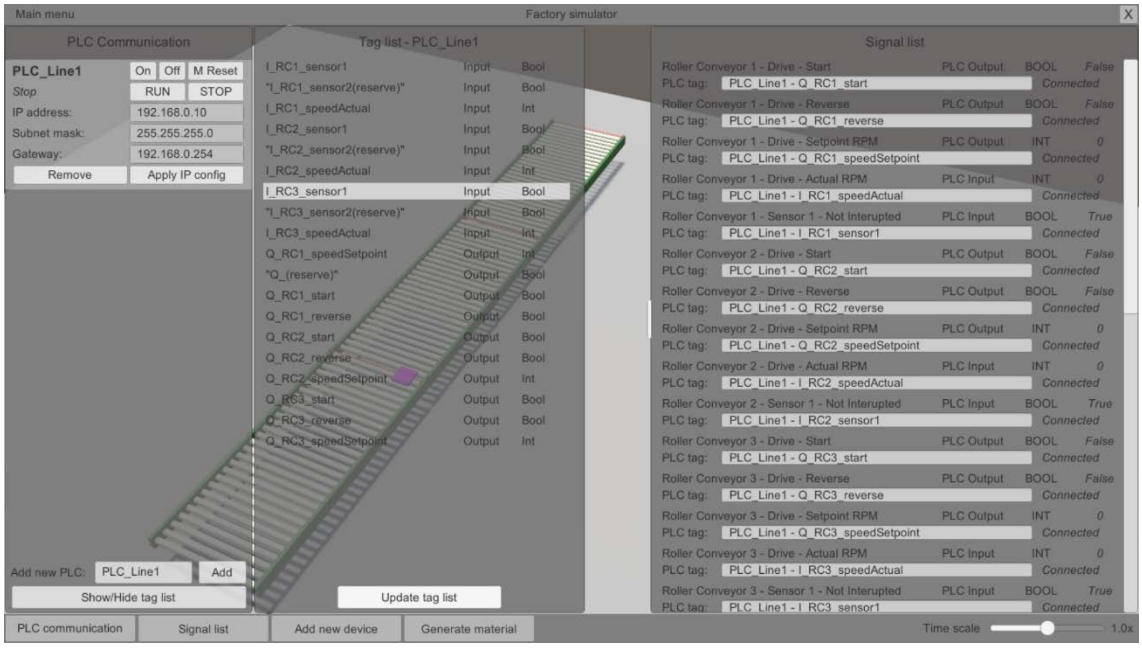


Obr. 4-7: Obrazovka vizualizace pro ovládání linky

Spuštěný runtime vizualizace (konkrétně obrazovku pro ovládání linky) můžeme vidět na Obr. 4-7. Obrazovka obsahuje v horní části přepínač pro zajištění/přerušení bezpečnostního okruhu a tlačítka pro volbu režimu linky (automatický/manuální) a pro spuštění/zastavení automatického chodu linky. Ve spodní části je poté vizualizace a ovládání jednotlivých dopravníků. Každý dopravník má svoje pole pro zadání požadované rychlosti a zobrazení aktuální rychlosti, tlačítka pro manuální ovládání pohybu a indikátor stavu optického senzoru (zelená = závora nepřerušena, červená = závora přerušena).

4.4 Otestování simulace

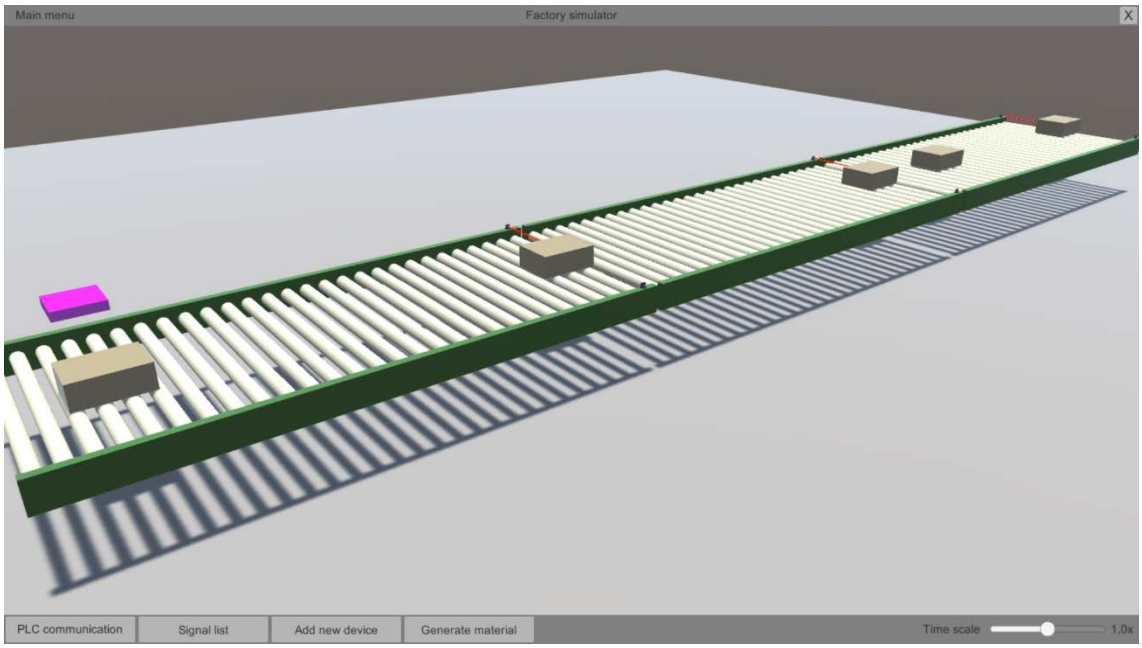
V simulátoru jsem vytvořil nové PLC s názvem „PLC_Line1“, nastavil mu IP adresu 192.168.0.10 a nahrál do něj připravený program. Poté jsem zobrazil jeho seznam tagů a ty přiřadil příslušným signálům linky, jak lze vidět na Obr. 4-8.



Obr. 4-8: Propojení signálů a PLC tagů

Nyní stačí přepnout PLC do stavu RUN, spustit vizualizaci a můžeme linku ovládat. Po startu automatického řízení linky a postupném vygenerování několika kusů materiálu se linka chová podle očekávání. Nejprve všechny dopravníky běží a poté, co první kus materiálu dorazí na konec linky, se poslední dopravník zastaví. Dále poté, co dorazí kus na konec druhého dopravníku, je zastaven i tento dopravník. Následně je stejně tak po přerušení prvního senzoru zastaven i první dopravník. Tento stav, kdy jsou dopravníky zastaveny, je zachycen na Obr. 4-9. Pokud první kus odstraníme z konce linky, poslední dopravník se opět rozjede a následně i druhý a

první dopravník. Video ukázka vytvoření a otestování této simulační úlohy je v Příloze 4.



Obr. 4-9: Automatické řízení linky

Závěrem je třeba dodat, že pro simulaci je potřeba mít dostatečně výkonný hardware, protože je spuštěn simulátor (včetně procesu PLCSIMInterface), procesy instancí PLCSIM Advanced, TIA Portal a WinCC runtime pro vizualizaci. Je pro to potřeba výkonný procesor a hlavně dostatek operační paměti. Při spuštěném simulátoru s jednou běžící PLC instancí a spuštěném TIA Portalu a WinCC runtime je potřeba zhruba 9 GB RAM. Každá další běžící PLC instance vyžaduje navíc něco přes 1 GB operační paměti. Dále je vhodné mít k PC připojené dvě obrazovky, jednu pro simulátor, druhou pro WinCC runtime. Teoreticky je možné místo WinCC runtime, běžícího na PC, použít reálný HMI panel, připojený přes Ethernet. Prakticky jsem to však nezkoušel, protože nemám žádný panel k dispozici.

5 ZÁVĚR

Hlavním cílem této diplomové práce bylo vytvořit simulátor dopravníkových výrobních linek, na kterém by bylo možné testovat a ověřit řídicí PLC program, ve spolupráci se simulačním nástrojem PLCSIM Advanced.

V první kapitole jsem provedl rešerši aktuálních trendů a možností v oblasti virtuálního zprovoznění a popsal nejpoužívanější simulační nástroje firmy Siemens, která je největším hráčem v oboru. Informace jsem ve velké míře čerpal z nedávných webových prezentací od firmy Siemens, a jsou tak velmi aktuální.

Ve druhé kapitole jsem detailně rozebral nástroj Siemens PLCSIM Advanced, který umožňuje vytvořit digitální dvojče PLC SIMATIC S7-1500. Tento nástroj je naprostý základ pro plně softwarové simulace, tedy bez použití reálného hardware, a obsahuje rozhraní pro všechny simulační nástroje, popsané v první kapitole. Zároveň má otevřené rozhraní, pro komunikaci s externími aplikacemi, které jsem zde detailně popsal a dále využil pro programování vlastní aplikace.

Ve třetí kapitole jsem popsal návrh a realizaci vlastního simulátoru. Pro jeho vytvoření jsem využil vývojový nástroj Unity3D, který obsahuje vlastní fyzikální a vykreslovací engine, a umožňuje tak vytvořit 3D simulátor na vysoké úrovni. Tento nástroj slouží hlavně pro vývoj 3D her a pro náš obor není nijak zajímavý nebo přínosný, proto jsem mu v této práci věnoval pouze malou část, nutnou pro pochopení některých souvislostí. Nakonec se však volba nástroje neukázala jako nejšťastnější, neboť jsem narazil na problém, že Unity3D nedokáže zkompilovat knihovnu, nutnou pro komunikaci s PLCSIM Advanced. Tento problém jsem musel řešit naprogramováním další aplikace, která zajišťuje výměnu dat mezi PLCSIM Advanced a vlastním simulátorem, což mě stálo hodně času. Na druhou stranu mi tento nástroj poskytl možnosti pro vytvoření velmi dobře vypadajícího simulátoru a nevím, zda by pro vytvoření 3D simulace byl jiný nástroj, například Blender nebo Unreal Engine, lepší volbou.

Ve čtvrté kapitole jsem vytvořil jednoduchou simulační úlohu, na které jsem demonstroval reálné možnosti použití mnou vyvinutého simulátoru.

Vytvořený simulátor umožňuje zatím vytvářet linky pouze z jednoho typu dopravníků, na kterých mohou být umístěny pouze optická čidla (závory). V tomto stavu je tak vhodný pouze pro jednoduché a čistě dopravníkové linky. Další zařízení, jako otočné, zvedací či vícesměrové dopravníky, mohou být doprogramovány a simulátor má tak potenciál stát se užitečným nástrojem např. pro virtuální oživování automatizovaných skladů. Také má potenciál stát se pomocníkem pro výuku PLC programování, kdy studenti si mohou vytvořit dopravníkovou linku a napsat pro ni řídicí PLC program a otestovat jeho funkčnost ve virtuálním prostředí.

Mezi další přínosy této práce bych zařadil souhrn aktuálních trendů v oblasti virtuálního zprovoznění včetně rozdělení a popisů nejpoužívanějších nástrojů od firmy Siemens, což může někomu pomoci se zorientovat v nepřeberném množství simulačních nástrojů, které Siemens nabízí. Dalším přínosem je detailní popis nástroje PLCSIM Advanced a zejména jeho otevřeného aplikačního rozhraní, což může sloužit jako návod pro vývojáře, kteří by chtěli napsat vlastní kosimulační aplikaci.

Simulátor je sice plně funkční, ale je zde také velký potenciál pro další vývoj a přidávání dalších funkcí. Kromě naprogramování dalších zařízení, které by bylo možné přidat do simulace, je možné například využít volně dostupnou neoficiální knihovnu Sharp7, která by umožnila integrovat do simulátoru S7 protokol, a mohlo by se tak vytvořit rozhraní pro komunikaci s reálnými PLC SIMATIC. Další možnost rozšíření simulátoru by byla ve využití TIA Openess, což je aplikační rozhraní do TIA Portal. Mohly by se tak například, při vytvoření nového zařízení v simulátoru, automaticky generovat v TIA Portalu standardizované bloky kódu, tagy nebo části vizualizace.

Literatura

[1] SIEMENS. *S7-PLCSIM Advanced*, Function manual. Siemens, 12/2017. Dostupné z: https://support.industry.siemens.com/cs/document/109760835/simatic-s7-1500-s7-plcsim-advanced-?dti=0&pnid=24442&lc=en-WW

[2] PÁSEK, Jan a Vlastimil BRAUN. *Automatizace procesů II*, učební text VUT, FEKT, Brno 2017.

[3] *Digitální továrna Tecnomatix* [online]. [cit. 2018-12-12]. Dostupné z: https://www.axiomtech.cz/24751-digitalni-tovarna.

[4] MAŘÍK, Vladimír. *Průmysl 4.0 – aktuální výzvy pro energetiku* [online]. Praha, 2018 [cit. 2018-12-10]. Dostupné z: https://s-ic.cz/wp-content/uploads/2018/03/SIC-Prof.-Marik-Energetika.pdf

[5] SIEMENS. *Průmysl 4.0* [online]. [cit. 2019-02-20]. Dostupné z: https://www.siemens.cz/prumysl40/

[6] Industry 4.0. In: *Wikipedia: the free encyclopedia* [online]. Last modified on 19. 4. 2018 [cit. 2018-12-15]. Dostupné z: https://en.wikipedia.org/wiki/Industry_4.0

[7] SIEMENS. *Virtuální zprovoznění šetří čas i náklady* [online]. Webová prezentace. [cit. 2019-04-10]. Dostupné z: https://zoom.us/recording/play/MlqZbMT2OLfxQnrQJ29asZ0IvSTVvoFCz2SHbsbFWj6PlLsCSNd-LxKChQrqu6As?continueMode=true

[8] *Virtuální ověřování složitých výrobků před zprovozněním* [online]. [cit. 2018-12-15]. Dostupné z: https://www.konstrukter.cz/virtualni-overovani-slozitych-vyrobku-pred-zprovoznenim/

[9] SIEMENS. *Simcenter Amesim, Driving innovation without compromise by optimizing the performance of mechatronic systems*. Siemens, 2018. Dostupné z: https://www.plm.automation.siemens.com/media/global/cz/Siemens-PLM-Simcenter-Amesim-Driving-innovation-fs-68692-A1_tcm84-38412.pdf

[10] Oficiální webové stránky společnosti Siemens PLM [online]. [cit. 2019-04-17]. Dostupné z: https://www.plm.automation.siemens.com/global/cz/index.html

[11] UNITY TECHNOLOGIES. *Unity Manual* [online]. Manuál k hernímu enginu Unity3D. [cit. 2019-04-15]. Dostupné z: https://docs.unity3d.com/Manual/index.html

Seznam příloh

Příloha 1 - Zdrojové kódy simulátoru 79
Příloha 2 - Simulátor 79
Příloha 3 - TIA Portal projekt 79
Příloha 4 - Video simulační úlohy 79

Příloha 1 - Zdrojové kódy simulátoru

Zdrojové kódy mé simulační aplikace jsou uloženy na přiloženém CD ve složce \Kosimulační aplikace\, která obsahuje:

- soubor Cosimulation.zip – komprimovaný archiv s projektem pro Unity3D 2018.3.12f1, který lze otevřít (překonvertovat) i ve všech novějších verzích Unity3D. Pro otevírání C# skriptů je potřeba MS Visual Studio.
- složku PLCSIMInterface – obsahuje C# projekt aplikace PLCSIMInterface pro MS Visual Studio.

Příloha 2 - Simulátor

Spustitelný soubor kosimulační aplikace má název CoSimulation.exe a je umístěn ve složce \Kosimulační aplikace\CoSimulation\ na přiloženém CD. Spustitelný EXE soubor je třeba přemisťovat s celou složkou \CoSimulation\, pokud by se nacházel jinde než v této složce, aplikaci nebude možné spustit. Aplikaci je možné spustit pouze na 64-bitovém systému.

Příloha 3 - TIA Portal projekt

TIA Portal projekt pro řízení a ovládání vytvořené simulační úlohy obsahuje PLC program a WinCC projekt. Je uložen na přiloženém CD ve složce \TIA Portal projekt\ v archivu Simulacni_uloha.zap15_1. Projekt je možné otevřít v TIA Portal V15.1 a novějším.

Příloha 4 - Video simulační úlohy

Video s vytvořením a demonstrací simulační úlohy. Je uloženo na přiloženém CD ve složce \Video simulační úlohy\ a má název Simulacni_uloha.flv.