VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020 Bc. Pavel Janeček

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

APLIKACE PRO GENEROVÁNÍ PLC PROGRAMŮ POMOCÍ TIA OPENNESS
THE APPLICATION FOR THE PLC PROGRAM GENERATION USING TIA OPENNESS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Pavel Janeček
AUTHOR

VEDOUCÍ PRÁCE — Ing. Jakub Arm
SUPERVISOR

BRNO 2020

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. Pavel Janeček ***ID:*** 186098

Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Aplikace pro generování PLC programů pomocí TIA Openness

POKYNY PRO VYPRACOVÁNÍ:

Vytvořte uživatelskou aplikaci v jazyce C#, která bude umět vytvořit projekt pro PLC a HMI dle zadaného předpisu. Pro samotné generování projektu využijte knihoven TIA Openness. Zvolte vhodný formát souboru s předpisem a vhodnou formu uživatelského rozhraní aplikace. Vytvořte referenční bloky pro PLC a HMI, které použijete pro vygenerování demonstračního příkladu (řízení buněk v projektu Barman) ze vzorového předpisu.

1) Navrhněte uživatelskou aplikaci průvodce generování programu pomocí TIA Openness.
2) Realizujte vlastní aplikaci průvodce.
3) Vytvořte referenční sadu modulů zařízení pro PLC a HMI.
4) Vytvořte vzorový předpis pro generování demonstrační PLC a HMI aplikace.
5) Ověřte funkčnost generované demonstrační aplikace.
6) Zdokumentujte aplikaci.

DOPORUČENÁ LITERATURA:

Jon Skeet. C# in Depth. 4th Edition. 528 s. Manning, 2019. ISBN 9781617294532.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: Ing. Jakub Arm

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

ABSTRAKT

Tato diplomová práce se zabývá vytvořením aplikace umožňující generování PLC a HMI programů pomocí TIA Openness. Aplikace je naprogramována tak, aby byla separována logika od grafiky, což je výhodné při případné obměně vizuální stránky aplikace. Generování programů je možné pomocí XML předpisu či vytvořením konfigurace v rámci průvodce aplikace. Generování probíhá přepisováním zdrojových XML souborů obsažených v aplikační knihovně dle navolené konfigurace. Aplikace byla vytvořena pomocí frameworku Windows Forms v jazyku C#. Byla vytvořena referenční sada modulů zařízení pro PLC a HMI ve vývojovém prostředí TIA Portal V15.1. Zdrojové XML soubory, na základě kterých generování probíhá, byly následně získány exportováním referenční sady a umístěny do knihovny aplikace. Pomocí simulačního nástroje Factory I/O a S-7 PLCSIM bylo ověřeno, že generovaný program je díky namapování proměnných na piny bloků plně funkční v manuálním režimu. Zároveň bylo dokázáno, že lze generovat programy pro různé stroje, což svědčí o univerzálnosti použití aplikace. Přínosem této práce je podstatné zkrácení doby při vytváření nových PLC a HMI programů.

KLÍČOVÁ SLOVA

TIA Openness, TIA Portal, C#, PLC, HMI

ABSTRACT

This thesis focuses on creating an application that allows generation of PLC and HMI programs using TIA Openness. The application is programmed to separate logic from graphics, which is beneficial in case of a change of the visuals of the application. Generating programs is possible thanks to the XML transcription or by creating a configuration within the application guide. The generating is done by overwriting source XML files contained in the application library according to the selected configuration. The application was created with Framework Windows Forms in C# language. A reference set of device modules has been created for PLC and HMI in the development environment of TIA Portal V15.1. The source XML files, on the basis of which the generating is carried out, were then obtained by exporting a reference set and placed into the application library. With a simulation tool Factory I/O and S-7 PLCSIM it was verified that the generated program is, thanks to the mapping of variables on pins of blocks, fully functional in manual mode. At the same time it was proved that it is possible to generate programs for different devices which signifies the versatile use of this application. The contribution of this work is the significant reduction of time when creating new PLC and HMI programs.

KEYWORDS

TIA Openness, TIA Portal, C#, PLC, HMI

JANEČEK, Pavel. *Aplikace pro generování PLC programů pomocí TIA Openness*. Brno, 2020, 102 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce: Ing. Jakub Arm

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Aplikace pro generování PLC programů pomocí TIA Openness“ 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/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno 1. 6. 2020 .....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Jakubu Armovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno 1. 6. 2020 .....

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Programování řídících algoritmů pro nové stroje ovládaných PLC předchází vytvoření základní kostry programu a odzkoušení všech komponent v manuálním režimu. K tomuto je většinou používáno již dříve definovaných struktur nebo hotových komponent pro moduly zařízení, které je programátor nucen kopírovat manuálně ze starších projektů. Při velkém počtu modulů zařízení se tento proces stává velmi nekomfortním a hlavně časově náročným, proto je snaha zefektivnit čas programátora tak, aby nebyl nucen vykonávat velké množství rutinních úkonů. Při programování PLC od firmy Siemens má pomoci rozhraní TIA Openness, díky kterému by zakládání a tvorba nových projektů měla být efektivnější.

Cílem práce je vývoj uživatelské aplikace, která za použití knihoven TIA Openness zefektivní vytváření nových projektů. V ideálním případě by měl být vygenerovaný projekt plně funkční v manuálním režimu a programátor by řešil pouze vytvoření algoritmu pro automatické řízení. Vygenerování projektu by mělo být umožněno jednak pomocí XML předpisu, tak i v uživatelském rozhraní této aplikace. Pro generování se využívá zdrojových XML souborů z vytvořené referenční sady modulů zařízení. Hlavními požadavky na aplikaci je možnost konfigurovat PLC i HMI zařízení. U PLC se jedná o importování tagů z CSV souboru a vkládání uživatelských datových typů. Stěžejní funkcí by však měl být import bloků do projektu s možností přivedení proměnných na vstupy/výstupy těchto bloků a jejich následné programové volání. V rámci HMI byl kladen důraz především pro konfigurování obrazovek. Měly by být přítomny jednak funkce pro změnu rozměrů a pozice komponent obrazovek pro HMI s různým rozlišením, tak i funkce pro co nejefektivnější přiřazení PLC proměnných k událostem obrazovek.

Teoretická část práce se nejdříve zabývá rešerší základních pojmů z oblasti programovacího jazyka C#, které jsou nezbytné pro vytvoření uživatelské aplikace. Dále je představeno vývojové prostředí TIA Portal a rozhraní TIA Openness, kde jsou popsány základní funkce z knihoven TIA Openness.

V praktické části práce je ve třetí kapitole popsán přístup k problému a navržení struktury programu. Dále je představena vlastní aplikace se všemi jejími funkcemi. První polovina čtvrté kapitoly se zabývá vytvořenou referenční sadou modulů zařízení, které se používají pro generování demonstračních příkladů a navrženou strukturou předpisu pro generování. Ověření funkčnosti vygenerovaných projektů je zprostředkováno pomocí simulačního nástroje Factory I/O a věnuje se mu druhá polovina čtvrté kapitoly. V páté kapitole jsou zdokumentovány jednotlivé vytvořené objekty umožňující konfigurování a následné generování projektů, přičemž proces generování je popsán samostatně.

1 Použité technologie

V této kapitole jsou popsány technologie potřebné při vytváření aplikace, která bude zprostředkovávat automatické generování programu pomoci TIA Openness.

1.1 Programovací jazyk C#

1.1.1 Objektově orientované programování

Pojem objektově orientované programování, dále jen OOP, není jen technikou nebo strukturou jak psát program, ale také i určitý způsob myšlení jakým se nahlíží na program, jak ho strukturovat do jednotlivých objektů často tak, aby se co nejvíce blížily realitě. Základním kamenem OOP jsou tedy objekty, které se přirovnávají k reálným objektům. Přidružují se jim data či vlastnosti, kterým se říká atributy objektu, a schopnosti, které se nazývají funkcemi či metodami daného objektu. Program se pak skládá dohromady z těchto objektů. OOP je mocným nástrojem, bez kterého se velké projekty téměř již neobejdou. Avšak i malé projekty je výhodné psát pomocí OPP, jelikož to přináší snadnější orientaci v programu, jeho ladění, rozšiřování, upravování či inovování. OOP je postaveno na základních třech pilířích. [1] [2]

- Zapouzdření
- Dědičnost
- Polymorfismus

V následujících části budou představeny vybrané pojmy z oblasti OOP.

Třída

Třída je základní konstrukční prvek objektově orientovaného programování [4]. Třídu si můžeme představit jako vzor, podle kterého se vytvářejí objekty. Třída obsahuje data a operace, které jsou určeny pro práci s těmito daty. *Zároveň slouží jako bezpečnostní prvek, kterým je řízení přístupových práv k datům a metodám [5].* Rozlišují se tři přístupová práva, a to veřejná - *public*, která jsou přístupná všem ostatním třídám, soukromá - *private*, která jsou přístupná pouze dané třídě, kde se vyskytují a poslední možností jsou dědičná - *protected*, která jsou přístupná pouze potomkům dané třídy. Metody jsou funkce dané třídy, sloužící k práci s daty, které daná třída obsahuje. Každý nový objekt vytvořený podle dané třídy se pak nazývá instance. Prvek se vytváří buď konstruktorem, který je téměř vždy typu public a jmenuje se stejně jako název třídy, nebo implicitním konstruktorem, který je definován automaticky překladačem. [1] [3] [4] [5]

Zapouzdření

Spočívá ve vytvoření rozhraní pro objekt, který je možno používat, aniž bychom věděli co se děje uvnitř. Toho je docíleno „schováním“ některých metod a atributů pomocí modifikátoru *private*. Naopak se modifikátorem *public* zveřejní ty metody, které budou udávat používání a chování objektu zvenčí. [1]

Dědičnost

Dědičnost umožňuje vznik nových tříd, které mohou být odvozeny od jiných. Mohou od nich převzít - zdědit některé atributy a metody. Takové třídě se pak říká odvozená nebo potomek. Třída ze které dědí, se nazývá předek nebo rodič. V rodičovské třídě se pomocí modifikátoru *protected* označí ty atributy, vlastnosti nebo metody, které budou zděděny pro potomka, který je bude moci využívat tak, jako by patřily jemu. [6]

Polymorfismus

Posledním ze tří stavebních kamenů je polymorfismus. Jedná se o vlastnost přepsat v potomkovi zděděnou metodu tak, aby odpovídala přímo dané třídě. Aby však v potomkovi bylo možné metodu přepsat, je připsáno za modifikátor přístupu ještě klíčové slovo *override*, které ukazuje, že daná metoda již existuje a bude přepisována. Dále musí být daná metoda označená v předkovi klíčovým slovem *virtual*, čímž se dá najevo, že potomek může danou metodu přepisovat. [7]

Statika

S nadsázkou by se dalo říci, že statika v C# nahrazuje globální proměnné. Jestliže je nějaký objekt statický, připisuje se k modifikátoru klíčové slovo *static*. Pokud se např. zavede statický atribut ve třídě, je tento atribut nezávislý na instanci dané třídy, respektive instanci tento atribut nenáleží, jelikož patří přímo třídě. Jinými slovy, lze k němu přistupovat, aniž by instance třídy existovala. Statika se muže např. využívat pro globální proměnné nebo i pro statické třídy. Z takových tříd nelze vytvořit instance a obsahují pouze statické metody. Většinou slouží jako obslužné, tzn. že neobsahují žádná data, jen logiku. [8]

Abstraktní třída

Z abstraktní třídy se nevytváří instance, neboť se jedná pouze o obecnou třídu, která je označena klíčovým slovem *abstrac*. Takto označená třída je pouze předpisem, jenž obsahuje metody či data společné pro potomky. Třídu lze označit jako abstraktní tehdy, pokud disponuje alespoň jednou abstraktní metodou. Taková

metoda obsahuje pouze hlavičku - nemá vyplněné tělo. Logika metody se doplňuje v jednotlivých potomcích. Zároveň takto označená metoda musí být použita u všech zděděných potomků. [9]

1.1.2 Výjimky

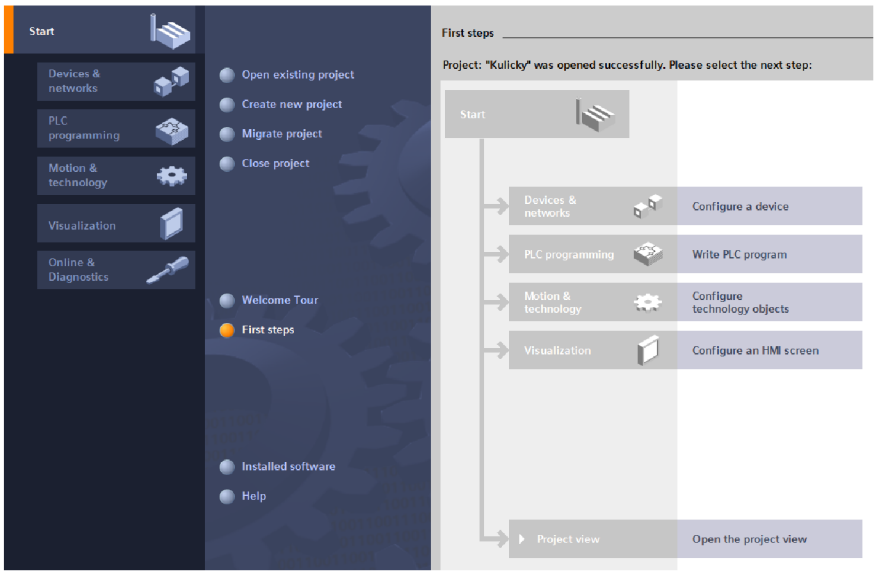
Výjimky (exceptions) slouží jako nový mechanizmus ošetření chyb. *Jedná se o ošetření „předpokládaných" chyb, tedy chyb, o kterých programátor ví, že mohou nastat, a tímto se na ně připraví [5].* V místě, kde vznikne chyba, je vygenerována, tzv. vyhozena, výjimka. Tato výjimka putuje programem až do místa, kde jí je možné „odchytit" a vyřešit bez toho, aniž by uživatel přišel o rozpracovaná data. [5]

1.1.3 Vytvoření DirectoryInfo/FileInfo objektu

Tyto objekty jsou obsaženy v knihovně System.IO. Slouží k zadání absolutní cesty k požadované složce. V rámci používání TIA Openness se uplatní např. pro nalezení projektu, který má být otevřen, nebo určení místa, kam se mají uložit exportované soubory či odkud se mají importovat soubory. Pokud cesta není absolutní nebo pokud odkazuje na místo, které neexistuje, program vygeneruje výjimku. [3] [10] [11]

1.2 TIA Portal

Totally Integrated Automation Portal, dále jen TIA Portal, je software od firmy Siemens pro programování, konfiguraci a diagnostiku řídících systémů. V současné době nemá na trhu téměř konkurenci. Jeden z důvodu je, že jako jediný program obsahuje společnou platformu pro vývoj řídících aplikací a vytváření vizualizací. Jinými slovy TIA Portal sdružuje psaní programu pro PLC systémy a vytváření vizualizací, tedy vytváření operátorských rozhraní pro stroje a zařízení jak s použitím operátorských panelů HMI (Human Machine Interface – dále jen HMI), tak i pro dispečerské systémy kategorie SCADA (Supervisory Control And Data Acquisition – dále SCADA). Když je vše v jednom programu, pracuje se lépe, než když jsou části uloženy zvlášť. Vývojové prostředí navíc působí velmi příjemným dojmem. Vše je přehledně uspořádané, všechny prvky nastavení od konfigurace PLC až po vizualizace mají stejný princip a přidávání je velmi snadné. Na Obr. 1.1 je základní obrazovka projektu, kde lze přes jednotlivé záložky přistupovat ke všem částem projektu okamžitě a jednoduše. [3] [12] [13]



Obr. 1.1: Vývojové prostředí TIA Portal. [3]

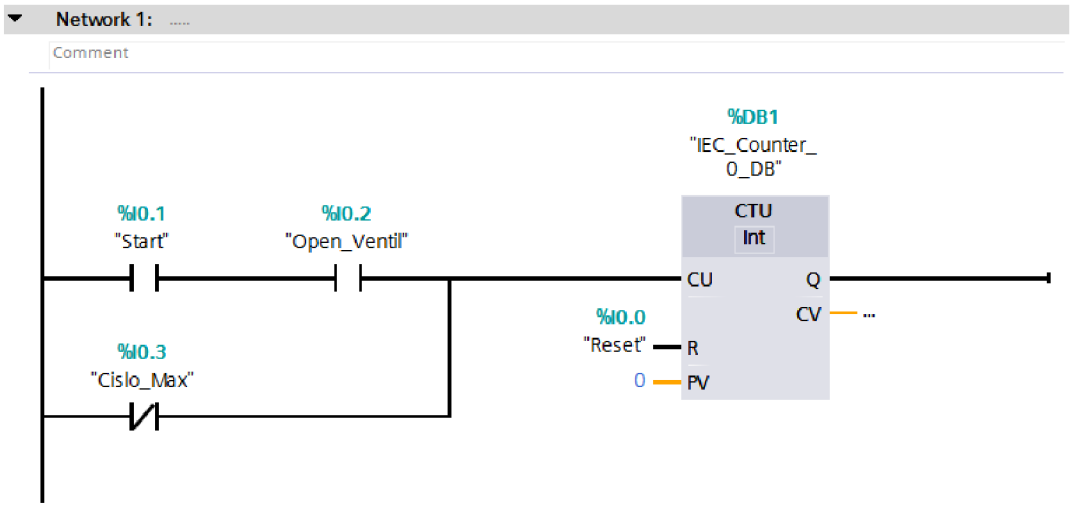
1.2.1 Programovací jazyky

Každý výrobce má pro jejich vývojové prostředí své programovací jazyky určené pro programování PLC, které jsou sice podobné, ale v žádném případě nejsou stejné a tudíž nejsou ani přenositelné mezi jednotlivými vývojovými prostředími od různých výrobců. Existuje však mezinárodní norma IEC 61 131-3, která sjednocuje typy jazyků do čtyř základních typů s přesně definovanou syntaxí. Podle této normy se jazyky tedy dělí na LD - Ladder Diagram, FBD - Function Block Diagram, IL - Instruction List a ST - Structured Text. Jako další lze uvést programovací jazyk SFC - Sequential Function Chart. Další dělení se používá dle formy jazyka a dělí se na grafické a textové. Mezi grafické jazyky patří LD, FBD a SFC. Mezi textové pak patří jazyky ST a IL. [3] [14] [15]

Jazyk LD - Ladder Diagram

Jazyk příčkového diagramu spadá mezi grafické programovací jazyky. LD je založen na principu reléové logiky. Program se píše do jednotlivých příček, které se spíše označují jako network, přidáváním prvků z instrukční sady, jako jsou spínací a rozpínací kontakty, funkce, funkční bloky a mnoho dalších. Tento jazyk je pro svou jednoduchost vhodný pro začátečníky. Při složitějších aplikacích se však stává nevyhovujícím a nepřehledným. [3] [14] [16]

Na Obr. 1.2 je ukázka jazyka LAD reprezentující normovaný jazyk LD, vytvořeného v prostředí TIA Portal od firmy Siemens. Na tomto obrázku jsou zobrazeny spínací a rozpínací kontakty a pak funkční blok pro čítač čítající vpřed. Z obrázku je patrné, že se jedná o velmi přehledný programovací jazyk.



Obr. 1.2: Ukázka jazyka LAD z vývojového prostředí TIA Portal. [3]

Jazyk IL - Instruction List

Tento jazyk patří do skupiny textových jazyků. Jeho syntaxe může být podobná jazyku assembler. Je tvořen seznamem instrukcí, které se vždy píší na samostatný řádek. Instrukce obsahují operátor, v případě potřeby modifikátor a jeden nebo více operandů. Kód se dělí do jednotlivých networků podle jejich funkce. Toto dělení zajišťuje alespoň malou přehlednost programu. Pro složité aplikace je ale opět nevyhovující, protože programátor je nucen znát velké množství instrukcí a program je pak příliš rozsáhlý a nepřehledný. [3] [16] [14]

```
Network: 6

A     T      1
S     Q     40.1
L     S5T#1S
SD    T      2
```

```
Network: 7

A     T      2
R     Q     40.1
R     M     50.1
R     M     50.0
```

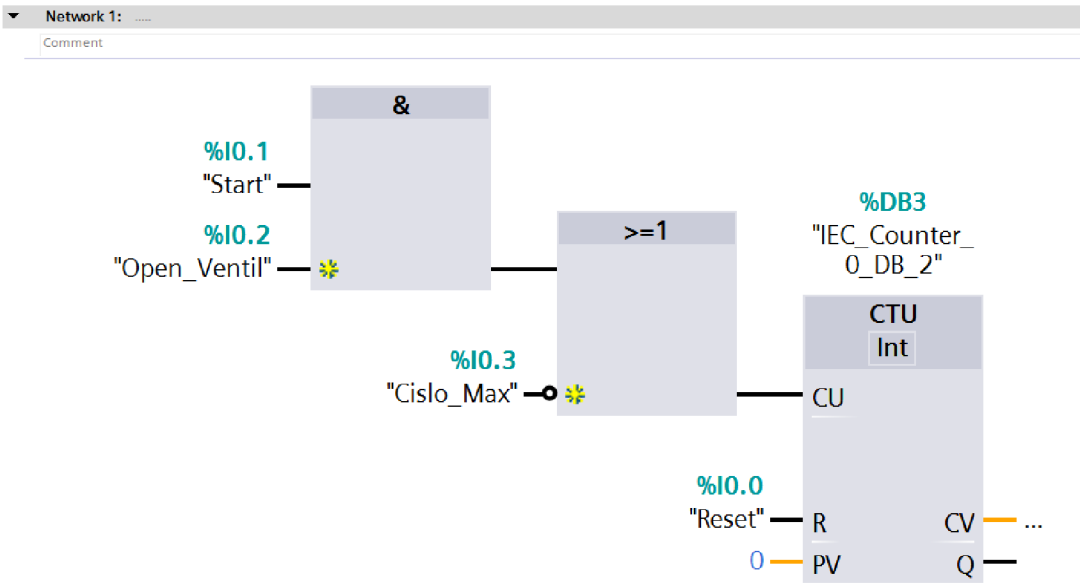
Obr. 1.3: Ukázka jazyka IL z vývojového prostředí Simatic Manager. [3]

Na Obr. 1.3 je ukázka jazyka IL z vývojového prostředí Simatic Manager, z kterého je patrné, že se jednotlivé instrukce píší vždy na samostatný řádek a že kód je dělen do networků. Pro složité funkce je tento jazyk velmi nevhodný a stává se velmi nepřehledným.

Jazyk FBD - Function Block Diagram

V českém překladu funkční blokové schéma. Jak název říká, jedná se o jazyk tvořený pomocí funkčních bloků, kdy jednotlivé operace jsou prováděny zapojováním bloků

v paralelních či sériových kombinacích. Funkční bloky mohou obsahovat logické, aritmetické operace či funkce. Z tohoto popisu vyplývá, že se jedná o grafický typ jazyka a v principu je podobný jazyku LD. [3] [16]



Obr. 1.4: Ukázka jazyka FBD z vývojového prostředí TIA Portal. [3]

Na Obr. 1.4 je ukázka jazyka FBD, jenž znázorňuje stejný program jako v příkladu na Obr. 1.2. Program vypadá podobně a je opět přehledný.

Jazyk ST - Structured Text

Strukturovaný text, jak již název napovídá, patří mezi textové programovací jazyky. Syntaxe je podobně jako např. v jazyce C dána definovanými výrazy a příkazy. Oproti jazyku IL nemusí programátor znát velké množství instrukcí a program je i přehlednější. Jazyk Structured text je pokládán za výkonný programovací jazyk. Z důvodu přehlednosti je využíván pro složitější programy. [3] [16] [14]

Na Obr. 1.5 je ukázka jazyka SCL z vývojového prostředí TIA Portal od firmy Siemens, který je postaven na základě normy jazyka ST. Opět se jedná o stejný program jako na obrázcích 1.2 a 1.4. Tento program sice není vizuálně tak přehledný, ale pro složitější programy, kdy by byl zápis při použití jazyka LD či FBD velmi složitý, např. pro psaní velkého množství výpočtů, je tento jazyk výhodnější.

Jazyk SFC - Sequential Function Chart

V českém překladu sekvenční funkční diagram, častěji nazýván GRAPH. I z tohoto označení plyne, že se jedná o grafický programovací jazyk. Tento program je vhodný pro programování stavových automatů, protože tak se i tento jazyk chová. Program

```
1
2
3  IF "Povoleni" THEN
4      // Statement section IF
5      "IEC_Counter_0_DB_1".CTU(CU := "Start"
6                               AND "Open_Ventil"
7                               OR NOT "Cislo_Max",
8                               R := "Reset",
9                               PV := 0);
10                              //Q=>_bool_out_,
11                              //CV=>_int_out_);
12
13
14 END_IF;
```

Obr. 1.5: Ukázka jazyka SCL z vývojového prostředí TIA Portal. [3]

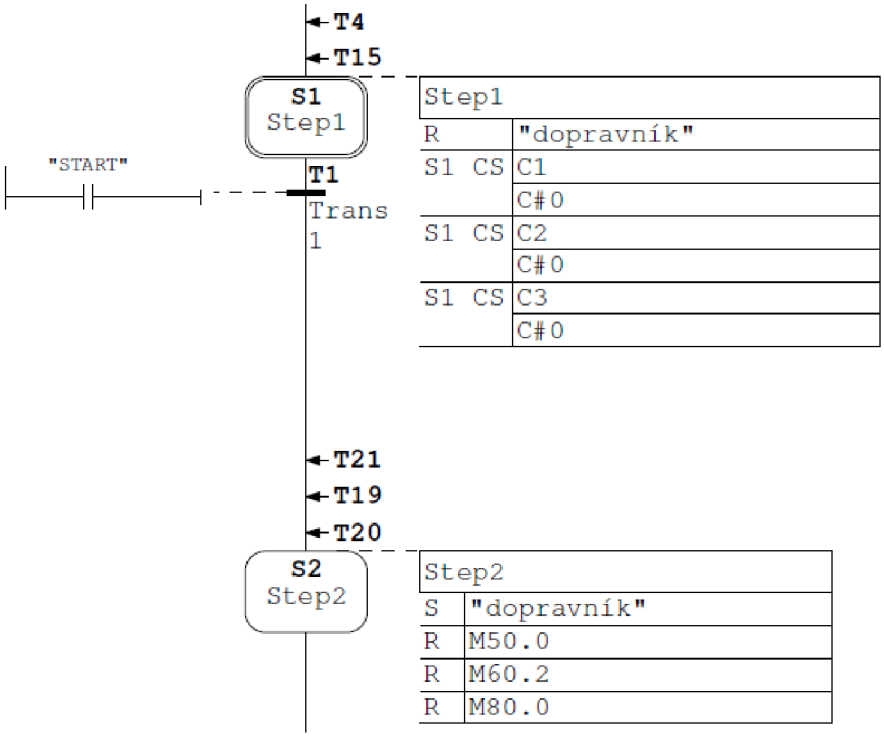
je složen z tzv. startujícího bloku, mezibloků, které se mohou dělit do několika větví, z konečného bloku, kterých může být více, a z přechodů mezi jednotlivými bloky. Bloky pak reprezentují jednotlivé stavy, ve kterých se automat nachází, a ke každému bloku je přiřazena akce, která se vykoná. Přechod do dalšího bloku je podmíněn nadefinovanou podmínkou. Každý prvek, tzn. přechod i blok akcí, může být naprogramován v libovolném jazyce definovaném v normě, včetně vlastního SFC. Velmi často se využívá například spojení se strukturovaným textem. Většinou se využívá většího počtu bloků, kdy každý blok vykonává malou část celého stavového automatu. Ukázka jazyka SFC je na Obr. 1.6. [3] [14]

Programovací jazyky v TIA Portalu

Vývojové prostředí TIA Portal disponuje všemi jazyky dle normy IEC 61 131-3, která je popsána výše, včetně jazyka SFC. Ostatně všechny ukázky kódů pocházejí od firmy Siemens. Na výběr jsou zde jazyky LAD, FBD, STL, SCL a GRAPH, kde jazyk LAD odpovídá normovanému LD, STL jazyku IL, SCL jazyku ST a GRAPH pak jazyku SFC.

1.2.2 Struktura programu

Program se ve vývojovém prostředí TIA Portal konfiguruje pomocí bloků. Bloky se dělí na čtyři základní typy. Základní bloky programu se nazývají organizačními bloky. Pro uchování dat, patřící k nějakému logickému celku slouží datové bloky. Pro opakujících se částí programu jsou zde funkce a funkční bloky.



Obr. 1.6: Ukázka jazyka SFC z vývojového prostředí Simatic Manager. [3]

OB – Organizační bloky

Jsou základními bloky v programu. Při vytvoření projektu se automaticky vytvoří i blok Main (OB1). OB jsou volány cyklicky, odkud se provádí volání dalších bloků a dochází k jejich zpracování. OB je několik druhů. [3] [17]

- OB Startup, který se vykoná jen jednou při uvedení PLC do provozu z režimu STOP.
- OB, které jsou cyklicky volány pro vykonávání programu.
- OB pro zpracování alarmů a chyb.

DB – Datový blok

Datové bloky slouží k uložení a uchování dat patřící k logickému celku. Datové bloky obsahují hodnoty proměnných, které jsou využívány v programu. Datových bloků může být nespočet a vytvářejí se tak, aby v jednotlivých blocích byly obsaženy informace, které spolu souvisejí. Každý funkční blok v programu může do DB zapisovat data nebo z něho číst. [3] [17]

FB – Funkční blok

Funkčními bloky jsou tvořeny podprogramy, které jsou vždy vykonány, když je funkční blok volán. Takovéto volání se nazývá instance funkčního bloku. Pro každé volání funkčního bloku je přidělen datový blok, do kterého se ukládají zpracovaná data a parametry funkčního bloku. Informace obsažené v tomto datovém bloku jsou uchovány i po vykonání FB. Funkční bloky se používají tehdy, když pro vykonání úlohy nestačí funkce, jinými slovy, je potřeba uchovat informace i po skončení volání funkčního bloku, pro další běh programu, které se uchovají v přiděleném datovém bloku. [3] [17]

FC – Funkce

Funkce obsahuje program, který se vykoná po volání z jiného bloku. K funkci se neváže žádná přidělená paměť, po vykonání funkce jsou použitá data ztracena. Funkci je možné volat několikrát z různých částí programu. Proto je funkce vhodná pro napsání jednoduchých částí programu, které se často opakují. [3] [17]

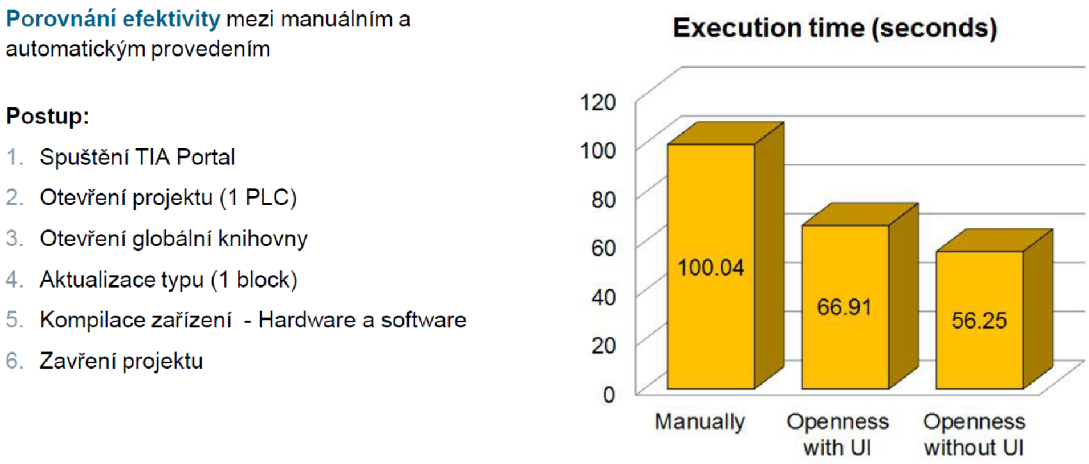
2 TIA Portal Openness

TIA Portal mimo jiné obsahuje i rozhraní TIA Portal Openness. TIA Portal Openness je vylepšením hlavně v řešeních pro digitalizaci. *Vývoj směřuje k tomu, že činnosti jako mechanický a elektrický návrh stroje, programování a testování řídicího systému, vlastní výroba stroje a další, budou probíhat stále více paralelně, čímž se může dosáhnout podstatného zrychlení a větší efektivity. K tomu, aby to fungovalo, je třeba zajistit perfektní koordinaci činností a provázanost všech dat. Digitalizace v pojetí TIA Portal verze V14 znamená pracovat s otevřenými standardy a vizualizovanými a maximálně propojenými systémy [18].* [3]

TIA Portal Openness umožňuje programátorům vytvořit nadstavbu, která je schopná automaticky generovat objekty a části programů jak pro PLC, tak i pro vizualizaci na základě informací uložených v textovém souboru nebo ve vhodně konstruovaném XML souboru. Namísto manuální práce, kde se pomocí myši a klávesnice pracovalo s programem TIA Portal, tak díky Openness, je možnost automatického spouštění požadovaného úkonu, aniž by se programátor nacházel v blízkosti pracoviště. *Díky TIA Portal Openness jsme schopni komunikovat a zadávat příkazy v TIA Portalu prostřednictvím volně programovatelného rozhraní, tzv. API (Application Programming Interface) [18].* TIA Portal Openness tedy zprostředkovává API do TIA Portalu. API, jak vyplývá z anglické zkratky, představuje rozhraní pro programování aplikací, kterými se ovládají jiné aplikace. V tomto případě TIA Portal. Openness přistupuje k funkcím TIA Portalu přes knihovny *Siemens.Engineering.dll* a *Siemens.Engineering.Hmi.dll*. Tyto knihovny obsahují jednotlivé třídy, které jsou rozděleny podle toho, k jakému účelu slouží. Pomocí těchto tříd je umožněno např. spustit TIA Portal na pozadí (WithoutUserInterface) nebo s uživatelským rozhraním (WithoutUserInterface). Spuštění na pozadí je efektivnější, protože jednotlivé úkony se poté provádějí rychleji, jak je zobrazeno na Obr. 2.1. Po spuštění se mohou vykonávat běžné úkony s projekty, jako je jejich vytvoření, otevření, uložení a zavření. Mimo jiné lze i spravovat zařízení, vizualizace, jednotlivé bloky, tagy a využívat mnoho dalších funkcí TIA Portalu. TIA Portal Openness dále nabízí možnost exportovat či importovat jednotlivé položky jako např. tabulky tagů, datové bloky, funkční bloky do/z XML souborů. [3] [18] [19] [20]

Pomocí TIA Portalu Openness lze výrazně automatizovat procesy, realizovat příkazy v TIA Portalu na dálku, díky čemuž je používání efektivnější a rychlejší. Jeho prostřednictvím dochází k minimalizaci chyb, rychlejšímu provádění úloh, vyšší konkurenceschopnosti, s čímž souvisí i účinnější využívání strojů. Největší předností je zkrácení doby uvádění strojů do provozu. Nejčastějším použitím je například automatické generování projektů pro více strojů, protože většina programů bude

mít vždy stejný základ. Důležitým aspektem je možnost pomocí Openness převést knihovny všech projektů na nejnovější verzi, a to u všech najednou a automaticky. [3] [18] [19] [20]



Obr. 2.1: Porovnání efektivity mezi manuálním a automatickým provozem [20]

2.1 Použití

TIA Openness se používá hlavně pro přístup k projektům a k projektovým datům. Převážně se používá pro:

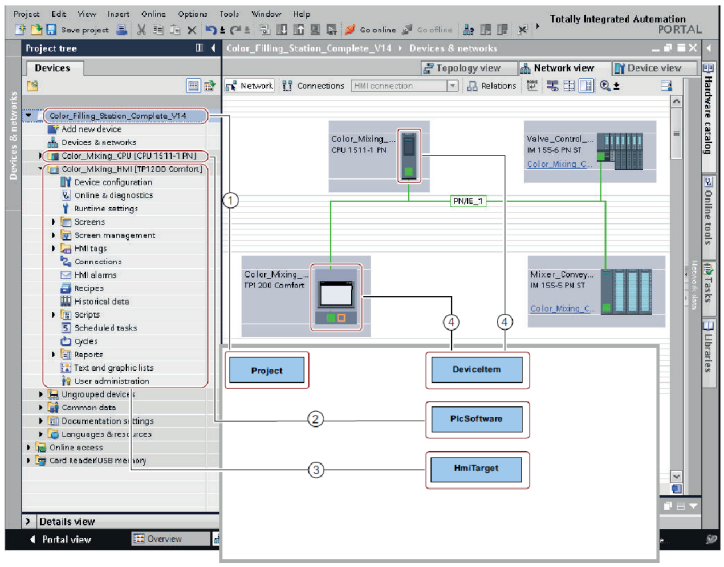
- Otevření, uložení a zavření projektů.
- Vyhledávání a vyjmenovávání objektů v projektu jako např. tabulka tagů, PLC a HMI zařízení aj.
- Vytváření objektů.
- Mazání objektů.

První volání funkce z TIA Openness aplikace trvá déle než volání dalších funkcí. Např. pokud se provádí více exportů konfiguračních dat za sebou, první export může trvat déle než následující. [3] [10]

Aby bylo možné používat TIA Openness, je nutné přidat mezi reference knihovny *Siemens.Engineering*, *Siemens.Engineering.Hmi* a *Siemens.Engineering.AddIn*. Po nainstalování balíčku TIA Portal Openness se tyto knihovny nacházejí v adresáři instalovaného TIA Portalu. Konkrétně se v této složce musí vybrat cesta *Siemens\Automation\PortalV14\PublicAPI\V15_1*. Toto je příklad pro nejnovější verzi TIA Openness, tedy dostupnou s verzí TIA Portal 15_1.

Na Obr. 2.2 je znázorněn vztah mezi objektovým modelem v API Openness a projektem otevřeným ve vývojovém prostředí TIA Portal. Číslem jedna je zobrazen otevřený projekt, který je v API představován třídou *Project*. Čísly jedna a dva je znázorněn software HMI respektive PLC zařízení, tedy to, kde se tvoří softwarový

program do projektu. Fyzický hardware, který je v projektu použit, spadá pod třídu *DeviceItem* a na obrázku je označen číslem čtyři. [10]



Obr. 2.2: Vztah mezi objektovým modelem Openness a projektem v TIA Portalu [10]

2.2 Vybrané funkce pro práci s Openness

V této kapitole budou představeny a popsány základní funkce pro práci s Openness. Pomocí těchto funkcí lze následně z naprogramované API aplikace ovládat jednotlivé segmenty v TIA Portalu.

2.2.1 Spuštění TIA Portalu

Pro spuštění TIA Portalu je nutné vytvořit novou instanci třídy TiaPortal, jež je obsažena v jmenném prostoru, dále jen namespace *Siemens.Engineering*. Z výpisu 2.1 si lze všimnout, že konstruktor této třídy obsahuje jeden parametr, kterým je TIAPortalMode. TiaPortalMode je enumerátor, který definuje dva módy WithUserInterface, dále jen with UI, a WithoutUserInterface, dále without UI. Tento parametr rozhoduje v jakém módu se TIA Portal otevře. Pokud je zvolena možnost with UI, tak se spustí TIA Portal v uživatelském grafickém prostředí tak, jak by se spustil obvykle bez Openness aplikace. Tento režim je vhodný pro ladění Openness API aplikace, jelikož je možné pozorovat změny, které se provádí, popřípadě některé věci měnit manuálně. Při spuštění s módem without UI se program spustí na pozadí, což přináší větší rychlost provádění úloh tak, jak je znázorněno na Obr. 2.1 viz str. 23. Proto při odladěné API aplikaci by se měl spíše používat tento režim. [3] [10]

```
//Spuštění TIA Portalu s módem Without UI
TiaPortal MyTiaPortal =
    new TiaPortal(TiaPortalMode.WithoutUserInterface);

//Spuštění TIA Portalu s módem With UI
TiaPortal MyTiaPortal =
    new TiaPortal(TiaPortalMode.WithUserInterface);
```

Výpis 2.1: Start TIA Portalu

2.2.2 Práce s projektem

Pro práci s projektem slouží třída Project a ProjectComposition opět z namespace *Siemens.Engineering*. Pro otevření již existujícího projektu se využívá metoda *Open()* třídy ProjectComposition, pomocí níž se vytvoří nová instance třídy Project. Použití demonstruje výpis 2.2. K otevření projektu pomocí Openness musí být již spuštěn TIA Portal, tedy musí být vytvořena instance třídy TiaProtal, ve výpisu označená jako MyTiaPortal. Otevřít lze však jen projekty, které byly vytvořeny s aktuální verzí TIA Portalu. Pokud by bylo požadováno otevřít projekt, který byl vytvořen z předchozí verze TIA Portal, tak by metoda *Open()* vygenerovala výjimku. Pro otevření takového projektu se použije metoda *OpenWithUpgrade()*. Tato metoda vytvoří nový upgradovaný projekt a otevře jej. Ovšem pokud by se otevíral projekt z verze starší než je předchozí, opět by se vygenerovala výjimka. Pro otevření projektu je potřeba nejdříve získat cestu k tomuto projektu. To se nejlépe udělá pomocí otevřeného Průzkumníku Windows, kde se jednoduše vybere požadovaný projekt. [3] [10]

Nový projekt se vytvoří pomocí metody *Create* obsažené opět ve třídě ProjectComposition. Tato metoda se volá s parametrem targetDirectory, který představuje cestu k adresáři, respektive místo k uložení projektu, a s parametrem Name, což je jméno nového projektu. Kód pro vytvoření nového projektu je ve druhé polovině výpisu 2.2.

```
//Získání cesty k projektu
OpenFileDialog fileSearch = new OpenFileDialog();
string ProjectPath = fileSearch.FileName.ToString();

//Získání nové instance třídy ProjectComposition
ProjectComposition projects = MyTiaPortal.Projects;

//Otevření existujícího projektu
MyProject = projects.Open(new FileInfo(ProjectPath));
```

```
//Dále kód pro vytvoření nového projektu
//Adresář, kam se uloží nový projekt
DirectoryInfo targetDirectory = new DirectoryInfo("Adresář");

//Vytvoření projektu
MyProject = projects.Create(targetDirectory,NameProject);
```

Výpis 2.2: Otevření a vytvoření projektu TIA Portal

Pokud obsluha nejdříve otevře projekt v TIA Portalu a posléze se chce k tomuto projektu připojit z API aplikace Openness, je situace nepatrně složitější. Jak ukazuje výpis 2.3, musí se získat list spuštěných procesů pomocí metody *GetProcesses()* třídy TiaPortal. Na tomto místě je vhodné ošetřit stav co se má stát, pokud je nalezeno více procesů, neboli když je spuštěno více projektů. Tato práce se zabývá generováním vždy jednoho projektu, tudíž pokud je nalezeno více jak jeden proces, vypíše se chybová hláška. Následně je nutné z procesu získat instanci třídy TiaProtal pomocí metody *Attach()*, která je součástí třídy TiaPortalProcess. V neposlední řadě se z nalezené instance TiaPortal přiřadí obsažený projekt jakožto instance třídy Project.

Pro uložení a zavření projektu se používají metody *Save()* a *Close()* třídy Project. [3] [10]

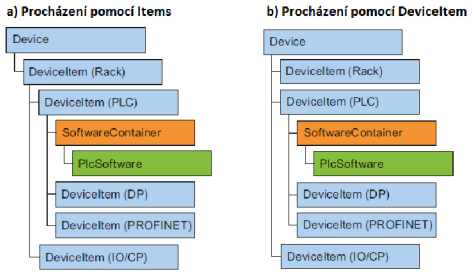
```
Project MyProject;

//Získání běžících procesů TIA Portalu
IList<TiaPortalProcess> processes = TiaPortal.GetProcesses();

//Vybrání prvního procesu
TiaPortalProcess _tiaProcess = processes[0];

//Získání instance TiaPortal
TiaPortal MyTiaPortal = _tiaProcess.Attach();

//Získání projektu
if (MyTiaPortal.Projects.Count <= 0)
    Console.WriteLine("No TIA Portal Project was found!");
else
    MyProject = MyTiaPortal.Projects[0];
```

Výpis 2.3: Připojení k otevřenému projektu

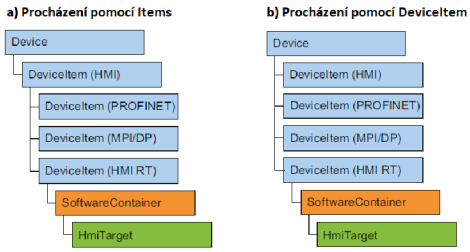
2.2.3 Práce se zařízením

Základními požadavky pro práci se zařízeními jsou: mít Openness aplikaci připojenou k TIA Portalu a mít vytvořenou instanci třídy *Project*.

V objektovém modelu se pracuje převážně se dvěmi třídami z knihovny Siemens. A to třídou *Device* a *DeviceItem*. Vztah mezi nimi je znázorněn např. na Obr. 2.3, kde je zobrazena hierarchie u PLC zařízení. Lze si povšimnout, že *Device* je seznamem konkrétních zařízení označených *DeviceItem*. Dále je zde patrné, že jsou dva způsoby jak procházet tento seznam. Za a) skrze *Items*, kde se jedná spíše o logické uspořádání, jelikož zařízení PLC je potomkem Racku, ve kterém je umístěno. Zatímco iterováním skrze *DeviceItem* je PLC na stejné úrovni jako Rack. Tento druhý způsob je spíše vhodný při vyhledávání zařízení, jelikož je vše na stejné úrovni a jednoduše se vybere požadovaný *DeviceItem*. Na Obr. 2.4 lze pak vidět totéž, jen pro HMI zařízení. [10]



Obr. 2.3: Hierarchie PLC zařízení z pohledu Item a DeviceItem [10]



Obr. 2.4: Hierarchie HMI zařízení z pohledu Item a DeviceItem [10]

Jelikož Openness se využívá především pro generování nových projektů, je nezbytné umět pomocí Openness vytvořit nové zařízení. Jednou z možností je využít metodu *CreateWithItem(string,string,string)* se třemi parametry - OrderNumber požadovaného CPU, jméno stanice např. PLC_1 a jméno zařízení. Tato metoda je součástí třídy DeviceComposition z namespace *Siemens.Engineering.HW*. Ukázka vytvoření nového zařízení v projektu je zobrazena ve výpisu 2.4, kde proměnná *MyProject* je instance aktuálně otevřeného projektu třídy Project. [10]

Dalším velmi potřebným objektem je software zařízení, a to konkrétně pro PLC zařízení PlcSoftware a pro HMI HmiTarget z namespace *Siemens.Engineering.SW*, respektive *Siemens.Engineering.Hmi*. Software je nutný k jakékoli programové práci

se zařízením, ať už se jedná o přidávání proměnných, bloků, obrazovek či jakýkoli import či export objektu do/ze zařízení. Příklad funkce pro získání softwaru zařízení je zobrazen ve druhé části výpisu 2.4. Pro ukázku je zde nastíněno jak lze najít zvlášť PLC či HMI software - nejvíce zanořený if-else. [10]

```
//Přidání zařízení
string OrderNum = "OrderNumber:6ES7 510-1DJ01-0AB0/V2.0";
string name = "PLC_1";
string devName = "Siemens1500";

DeviceComposition devices = MyProject.Devices;
Device MyDevice = devices.CreateWithItem(OrderNum, name, devName);

//Nalezení Softwaru zařízení
private Software GetSoftware(Device device)
{
  DeviceItemComposition deviceItemComposition = device.DeviceItems;
  foreach (DeviceItem deviceItem in deviceItemComposition)
  {
    SoftwareContainer softwareContainer =
      deviceItem.GetService<SoftwareContainer>();
    if (softwareContainer != null)
    {
      Software softwareBase = softwareContainer.Software;
      if (softwareBase is PlcSoftware)
      {
        PlcSoftware plcSoftware = softwareBase as PlcSoftware;
        return plcSoftware;
      }
      else
      {
        HmiTarget hmiTarget = softwareBase as HmiTarget;
        return hmiTarget;
      }
    }
  }
  return null;
}
```

Výpis 2.4: Práce s PLC a HMI zařízením

2.2.4 PLC tagy

Třídy obsažené v namespace *Siemens.Engineering.SW.Tags* slouží pro práci s tag tabulkou a PLC tagy v projektu. Co se týká tag tabulek, tak mezi nejužitečnější

funkce patří vyhledávání dostupných tag tabulek v projektu, vytvoření nové tag tabulky či smazání. Je umožněno spravovat jednotlivé tagy v tabulce jako např. jejich mazání vyhledání či vytvoření nového tagu. Jelikož poslední jmenovaná funkce je obzvláště užitečná při generování programu, je příklad jejího použití demonstrován ve výpisu 2.5. K vytvoření nového tagu slouží metoda *Create(string,string,string)* obsažená ve třídě PlcTagComposition. Parametry této metody jsou jméno tagu, typ a logická adresa. Nejdříve je však zapotřebí najít nebo vytvořit konkrétní tabulku tagů, do které má být daný tag vložen. Tabulka tagů se vytvoří obdobně jako tag, opět metodou *Create(string)*. Tentokrát se jedná o metodu třídy PlcTagTableComposition a jediným parametrem je jméno tabulky tagů. Použití je rovněž součástí výpisu 2.5. [3] [10]

```
//Vytvoření Tag Tabulky s názvem TagTable
PlcTagTableComposition tagTables =
      plcSoftware.TagTableGroup.TagTables;
PlcTagTable table = tagTables.Create("TagTable");

//Vytvoření nového tagu se jménem NewTag, typu Bool a adresou Q4.0
PlcTagComposition tagComposition = table.Tags;
PlcTag tag = tagComposition.Create(NewTag, Bool, Q4.0);
```

Výpis 2.5: Práce s PLC tagy

2.2.5 Export a Import

Vytváření objektů pomocí Openness je otázkou víceméně jen PLC tagů a PLC zařízení. Toto však nestačí pro generování nových projektů. Pro téměř veškeré operace pro přidávání objektů do projektu slouží importování z XML souborů. Stejně tak získávání potřebných objektů z projektu, např. pro přenos s ostatními zařízeními, se používá takřka jen metoda export, která vyexportuje daný objekt do XML souboru. [3] [10]

Funkce import a export se používají převážně pro následující účely: [10]

- Pro výměnu dat
- Kopírování částí projektu
- Externí zpracování konfiguračních dat pro nové projekty založené na stávajících konfiguracích
- Import externě vytvořených konfiguračních dat
- Poskytování dat projektu pro externí aplikace

Podporované bloky pro export a import jsou: [10]

- Funkční bloky (FB)

- Funkce (FC)
- Organizační bloky (OB)
- Datové bloky (DB)

Podporované jazyky pro export jsou: [10]

- STL
- FBD
- LAD
- GRAPH
- SCL

V Tab. 2.1 jsou označena data, která lze z projektu exportovat, respektive importovat.

Tab. 2.1: Tabulka dat, která mohou být exportována a importována [3]

Objekty	**Export**	**Import**
Bloky	**ANO**	**ANO**
Systémové bloky	**ANO**	**NE**
PLC Tag Tabulky	**ANO**	**ANO**
PLC Tagy a Konstanty	**ANO**	**ANO**
Uživatelem definované typy	**ANO**	**ANO**
Obrazovky	**ANO**	**ANO**
HMI Tag Tabulky	**ANO**	**ANO**
HMI Tagy	**ANO**	**ANO**

Export

Pro export souborů slouží metoda Export. Prvním parametrem této metody je FileInfo, jenž představuje cestu, kam má být exportovaný soubor uložen. Druhým parametrem je pak ExportOptions, který udává způsob, jak mají být data exportována, respektive která data má exportovaný soubor obsahovat. Jsou tři nastavení exportu, přičemž druhé a třetí nastavení exportu může být nastaveno současně. [3] [10]

1. ExportOptions.None
   Toto nastavení exportuje pouze upravená data nebo data, která se liší od výchozích. Exportovaný soubor obsahuje také všechny hodnoty, které jsou povinné pro následné importování dat.

2. ExportOptions.WithDefaults
   Exportovány jsou navíc i standartní hodnoty.
3. ExportOptions.WithReadOnly
   Exportovány jsou i hodnoty chráněné proti zápisu.

Import

Konfigurační data jsou importována z dříve exportovaného a upraveného souboru XML nebo z XML souboru. Údaje obsažené v tomto souboru jsou kontrolovány během importu. Data obsažena v XML souboru musí dodržovat stanovená pravidla. Importovaný soubor nesmí obsahovat chyby v syntaxi a struktuře. Pro komplexní změny je vhodné použít XML editor, který tyto kritéria kontroluje. Pokud jsou během importu zjištěny chyby, dojde k vygenerování výjimky. Při importu bloků je nutné brát na vědomí, že instance DB a použité tagy, které se vyskytují v tomto bloku, nejsou vytvořeny automaticky. Pokud v souboru XML není zadáno žádné číslo bloku, tak je přiřazeno automaticky. [3] [10]

K importu souborů slouží metoda Import a způsob je pak velmi podobný, jako tomu je u exportu. Do prvního parametru se udává cesta, odkud má být soubor importován, a jméno souboru. Druhým parametrem je pak ImportOptions, což udává, jakým způsobem mají být soubory importovány. Toto nastavení importu udává, jakým způsobem se mají objekty importovat, pokud již v projektu existují. Jsou dvě nastavení importu. [3] [10]

1. ImportOptions.None
   Při použití tohoto nastavení budou data importována bez přepsání. Pokud je objekt importován ze souboru XML, který již existuje v projektu, import je přerušen a vygeneruje se výjimka.
2. ImportOptions.Override
   Při použití tohoto nastavení budou data konfigurace importována s automatickým přepisováním. Pokud má objekt v projektu stejný název jako ten importovaný, tak je tento objekt přepsán. Respektive takový objekt je před importem smazán a vytvoří se nový s hodnotami obsaženými z importovaného souboru.

3 Realizace aplikace

Hlavním cílem práce bylo vytvořit aplikaci, která umožní nakonfigurovat projekt pro vývojové prostředí TIA Portal. Uživateli by mělo být umožněno kromě základních úkonů, jako je založení nového projektu a jeho následné uložení, zejména vytvoření uceleného projektu. Měly by být poskytnuty vhodné nástroje pro přidání nových zařízení a importování jejich softwarového vybavení.

Velká část práce se věnuje nejen importování dříve vytvořených bloků do nově přidaných zařízení, ale především poskytuje možnost navolit proměnné, které se přivádí na vstupy/výstupy bloků, např. funkčních bloků. Takto namapované bloky se posléze programově volají, a to buď v jazyku LAD nebo SCL. Toto by měla být stěžejní část této práce, společně s možností přiřadit PLC proměnné k událostem HMI obrazovek. Díky těmto možnostem by měl být po vygenerování projektu s nakonfigurovanými komponenty plně funkční manuální režim. Dalším z požadavků na aplikaci je vytváření tabulek tagů a vkládání tagů z CSV souboru do těchto tabulek.

Pro konfiguraci projektu a jeho následné generování do vývojového prostředí TIA Portal je nutné vytvořit aplikaci, která bude daný proces umožňovat. Je nutné nastínit samotnou strukturu projektu a navrhnout jeho členění do objektů tak, aby výsledný program byl co nejpřehlednější. Důležité je následně zvolit vhodné grafické uživatelské rozhraní, dále jen GUI, pro snadnou obsluhu aplikace.

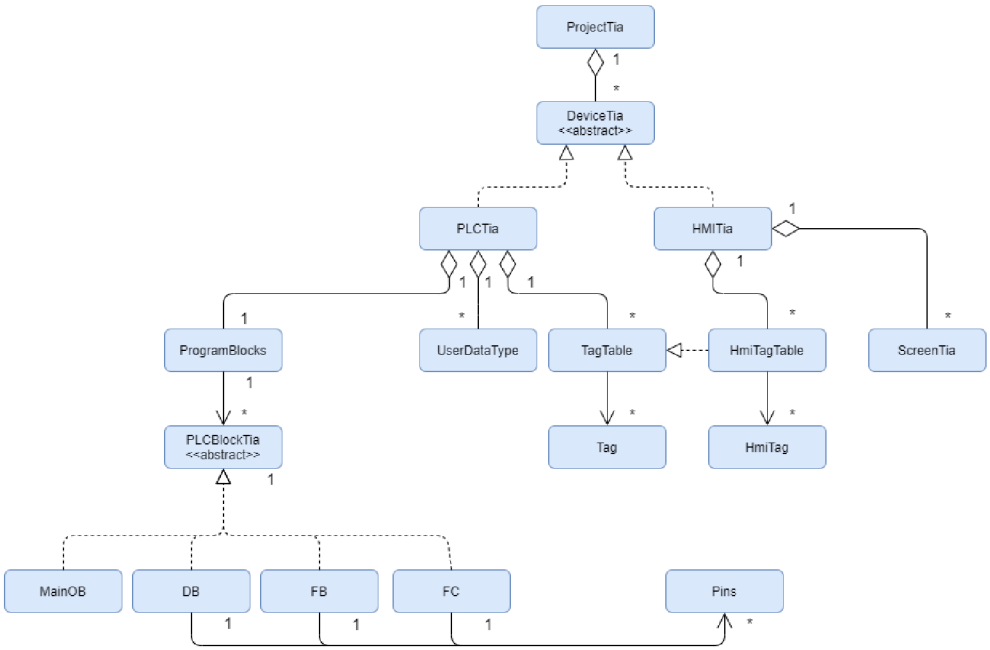
3.1 Návrh aplikace

Aplikace je navržena tak, aby se ovládala z okenního formulářového okna. V této práci je zvoleno Windows Forms App. Veškerá logika je zasazena do odpovídajících objektů tak, aby bylo možné v případě potřeby co nejjednodušeji použít jiné GUI. Grafická stránka aplikace je navržena pomocí komponenty *TabControl*, díky které je umožněno rozdělit funkcionalitu do jednotlivých záložek. Dále je separována struktura objektů pro uchování a konfigurování dat od objektů používající funkce z knihoven TIA Openness pro komunikaci s TIA Portalem.

3.1.1 Struktura programu

Před začátkem samotného programování aplikace bylo nutné nejprve navrhnout základní strukturu programu. Konkrétní navržená základní struktura je zobrazena na Obr. 3.1. Z tohoto diagramu je patrné, že třída *ProjektTia* zastřešuje celý konfigurovatelný projekt a uchovává tak všechna potřebná data pro generování. *ProjektTia* obsahuje jednu abstraktní třídu *DeviceTia*, z níž dále dědí *PLCTia* reprezentující

PLC hardware a *HMITia* pro HMI hardware. Abstraktní třída byla použita z důvodu definování společných atributů a metod obou zařízení. Následně obě zděděné zařízení mají definované objekty určené speciálně pro daný druh hardwaru.



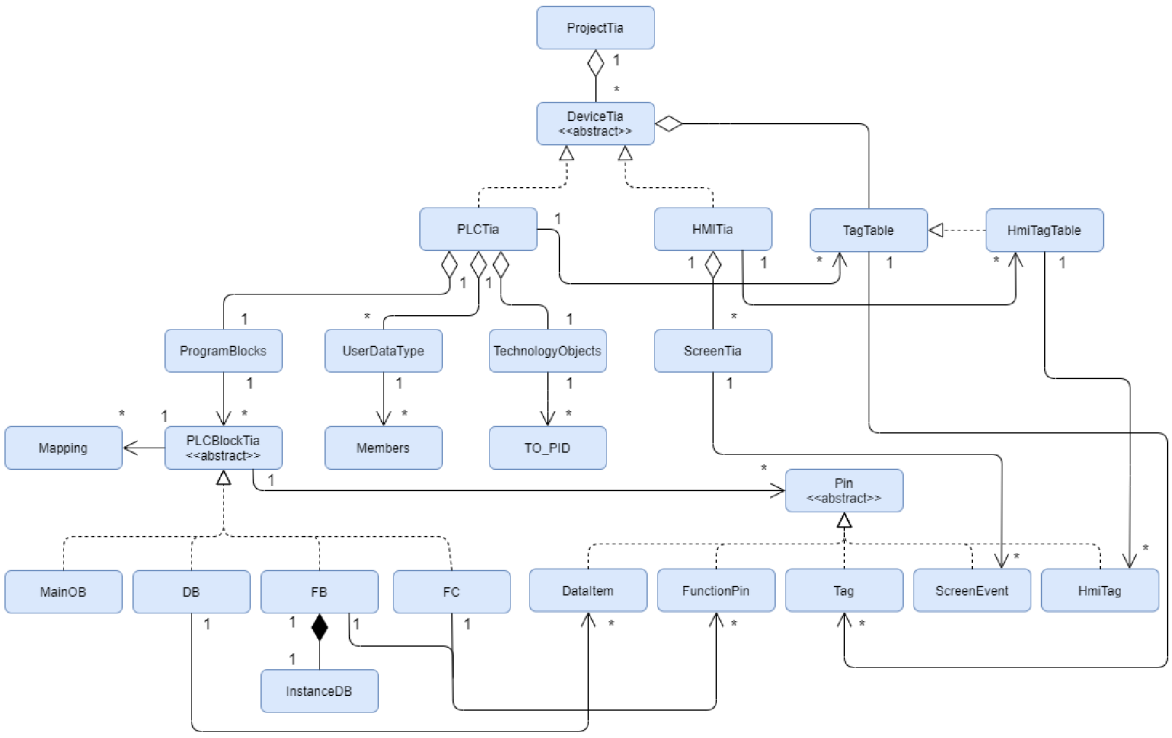
Obr. 3.1: UML diagram původní struktury programu

Levá větev UML diagramu, reprezentuje PLC zařízení, kde třída *PLCTia* spravuje objekty *ProgramBlocks*, *UserDataType* a *TagTable*. *ProgramBlocks* představuje jednotlivé programové bloky PLC programu. Řídí tak jednu abstraktní třídu *PLCBlockTia*, která obsahuje společné prvky pro všechny bloky, které z této třídy dědí. Třídy *FB, FC a DB* mají následně asociační vztah s třídou *Pins*, která reprezentuje vstupy/výstupy bloků. *PLCTia* obsahuje dále *TagTable*, jenž drží kolekci *Tag*.

Pravá větev pro HMI zařízení je na první pohled jednodušší než u PLC zařízení. Třída *HMITia* spravuje dva objekty *HmiTagTable* a *ScreenTia*. *HmiTagTable* obsahuje *HmiTag*, který dědí ze třídy *Tag*.

U vývojových aplikací je většinou základní struktura nedostatečná a postupně se přidávají další a další objekty. Z Obr. 3.2 je patrné, že základní struktura zůstala stejná, nicméně se rozšířila o nemalý počet dalších tříd. Nejpodstatnější změna je vidět především u pinů, jelikož se vyskytla potřeba definovat různé piny pro funkční bloky, datové bloky či události pro obrazovky. Všechny piny jsou zastřešeny abstraktní třídou *Pin*. Dále byly přidány třídy *InstanceDB* pro instanci funkčního bloku, *Mapping* uchovávající připojené proměnné na vstupy/výstupy bloků a *TechnologyObjets* uchovávající kolekci regulátorů (třída *TO_PID*).

Další již méně podstatné změny se týkají přesunutí kolekce pinů do abstraktní třídy pro bloky a jednotlivé typy bloků si následně definují jen žádaný druh pinu. Obdobně došlo k přesunu tabulek tagů do abstraktní třídy pro zařízení.



Obr. 3.2: UML diagram struktury programu

3.1.2 Zdroj dat

Jak již bylo jednou řečeno, TIA Openness najde uplatnění především tam, kde se často programují velice podobné projekty. Programátor pak většinu věcí kopíruje ze starších projektů a skládá je dohromady v projektu novém. Přesně tento koncept je využit i v této práci, při automatickém generátoru projektu do vývojového prostředí TIA Protal. Programátor nejprve otevře starší projekt, z kterého by obvykle vycházel. Pomocí vytvořené Openness aplikace poté jedním kliknutím exportuje všechny potřebné objekty a následně je uloží do knihovny. Programátor poté již nikdy nebude muset otevřít starý projekt, jelikož všechny informace bude mít aplikace uložené ve své knihovně.

V knihovně se nacházejí XML soubory jednotlivých exportovaných objektů z TIA Portalu, jako jsou např. funkční a datové bloky, funkce, uživatelsky definované typy či obrazovky HMI. Aplikace následně z těchto XML souborů čerpá informace a zprostředkovává je uživateli pomocí jednotlivých obrazovek. Uživatel si následně pouze vybere, které objekty by chtěl generovat do nového projektu, vytvoří namapování

bloků, přidá uživatelsky definované typy, vytvoří tag tabulky, tagy apod. Každý objekt, který vychází z XML souboru a je přidán do projektu, nese informaci o zdrojovém souboru, který je při generování upraven podle navolené konfigurace a následně importován do TIA Portalu.

Třída LibraryManager

Tato třída vznikla pro správu XML zdrojových souborů. Uchovává všechny XML soubory, které byly uživatelem přidány, v jediném privátním atributu *entities*, který je typu *List<Entity>*. Jedná se tedy o kolekci entit zdrojových souborů. Při přidávání každého XML souboru do tohoto seznamu se poznamená i o jaký se jedná objekt pomocí výčtového typu *TypeObject*, viz kap. 5.1.1. Dále tato třída obsahuje dvě statické proměnné, a to:

- Files2ImportPath : string – Adresářová cesta, kam se ukládají upravené zdrojové XML soubory.
- LibraryPath : string – Adresářová cesta ke zdrojovým souborům.

LibraryManager obsahuje dále tři metody. *GetObjectEntity(TypeObject type)* je první metodou, která vrátí entity požadovaného objektu. Další dvě metody jsou velmi důležité pro celý program. Jedná se o metody *Serializuj* a *Deserializuj*, které slouží pro uložení, respektive načtení celé konfigurace. Pomocí druhé z těchto metod je možné generovat projekt z XML předpisu, který se předá aplikaci. Jedná se tedy o druhý způsob jak vytvořit konfiguraci projektu.

Třída Entity

Jedná se o třídu vzniklou za účelem uchování cest k jednotlivým zdrojovým souborům. Jejími atributy jsou jméno souboru včetně koncovky, cesta k souboru a typ objektu jenž soubor reprezentuje - např. jedná-li se o zdrojový XML soubor k funkčnímu bloku, datovému bloku aj. Dále obsahuje metody pro získání jednotlivých atributů, jejíchž účel byl měl být zřejmý z jejich názvů. Vysvětlena bude pouze jediná metoda *ChangePath*, která změní cestu k souboru, popřípadě i jméno souboru, pokud se liší. Metoda vznikla v důsledku přepisování zdrojových souborů a jejich následném ukládání na jiné adresářové místo, tak aby zůstávaly zachovány zdrojové soubory.

Třída TiaObject

Jedná se o velmi jednoduchou, avšak užitečnou třídu, od které dědí mnoho tříd z projektu. Tato abstraktní třída vznikla na základě toho, že většina generovaných

objektů vychází ze zdrojového XML souboru a tudíž si musí v rámci aplikace uchovávat informaci o tomto souboru, ze kterého vychází. Informace o zdrojovém souboru je uložena ve vlastnosti *FileInfo*, jenž je typu *Entity*. Další společnou vlastností pro všechny objekty, které vycházejí ze zdrojového XML souboru, je jméno, které získávají právě z tohoto souboru. K získání jména je tato třída vybavena jedinou metodou, určenou pouze pro potomky, s názvem *GetNameFromFile()*. Její kód pro získání jména ze souboru je ve výpisu 3.1.

```
protected string GetNameFromFile()
{
    XmlDocument file = new XmlDocument();
    file.Load(fileInfo.GetFilePath());  //Načtení souboru
    //Získání elementu se jménem objektu
    XmlNode Name = file.SelectSingleNode("//AttributeList/Name");

  //Vrácení jména objektu
    return Name.InnerText;
}
```

Výpis 3.1: Získání jména objektu

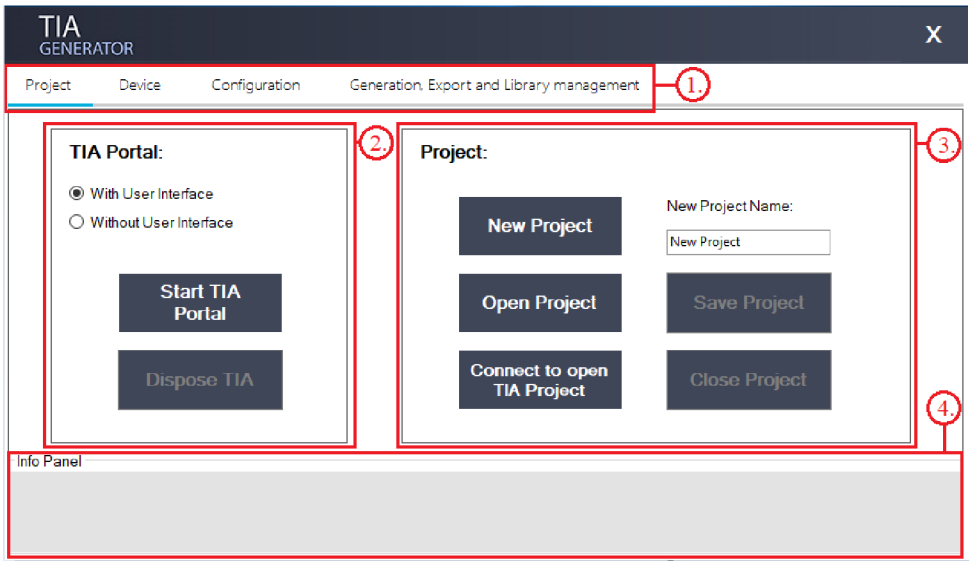
3.2 Vytvořená aplikace

Jak již bylo v této práci zmíněno, API aplikace je naprogramována pomocí platformy WinForm jako okenní formulářová aplikace. V této kapitole bude vytvořená aplikace představena a podrobně popsána, včetně použitých metod jednotlivých tříd pro konfiguraci projektu. Pro velké množství ovládacích prvků byla použita komponenta *TabControl*, která umožňuje vytvořit několik stránek a vytvořit tak menu. Pro lepší vizuální vzhled byla použita tato komponenta z knihovny *MetroModernUI.1.4.0.0*.

3.2.1 Hlavní obrazovka - správa projektu

Hlavní obrazovka aplikace je na Obr. 3.3. Obrazovka se skládá z několika částí, které jsou označeny čísly a popsány v následujícím seznamu.

1. Jednotlivé záložky komponenty *TabControl*, které představují hlavní menu aplikace. Z jednotlivých stránek, vyjmenováno zleva, je možné spravovat projekt, zařízení, konfigurovat zařízení, exportovat objekty, spravovat položky knihovny a generovat projekt. Na Obr. 3.3 je vybrána záložka Project, která je dále popisována.



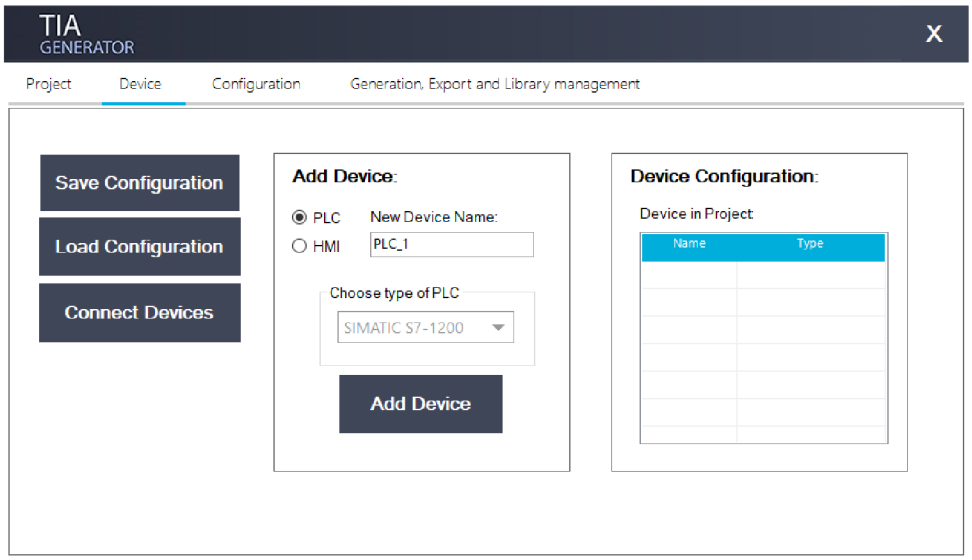
Obr. 3.3: Hlavní obrazovka aplikace - projekt

2. Menu pro spuštění TIA Portalu, a to buď s uživatelským grafickým rozhraním (*With User Interface*), nebo bez něj (*Without User Interface*). Spuštění je realizováno použitím metody *StartTiaPortal()* ze třídy *TIAService*.
3. Sekce pro práci s projektem, která umožňuje po stisknutí příslušného tlačítka založit nový projekt, otevřít již existující projekt, připojit se k již otevřenému projektu v TIA Portalu, uložit projekt a zavřít projekt. Po stisknutí vybraného tlačítka se stanou neaktivní ta tlačítka, která by již nedávala smysl. Například po stisknutí „založení nového projektu“, již nebude možné stisknout tlačítka pro otevření, či připojení k projektu. Naopak se zpřístupní možnost pro uložení či zavření projektu. Pod tlačítky se ukrývají metody třídy *ProjetTia* - *CloseProject()*, *ConnectProject()*, *NewProject()*, *OpenProject()* a *SaveProject()*.
4. Informativní panel, kde je vypsána informace o provedených akcích. Je důležitý zejména tehdy, pokud je zvolen režim bez uživatelského grafického rozhraní, kde není vizuální zpětná vazba. Informativní panel je plněn pomocí metody *ToString()* třídy *ProjetTia*.

3.2.2 Obrazovka zařízení

Po vybrání záložky Device se otevře okno pro správu zařízení v projektu. Z Obr. 3.4 je patrné, že v levé části lze nahrát či uložit konfiguraci celého projektu a přidat zařízení do sítí.

V prostřední části lze přidat zařízení do projektu. Jsou zde volby jak pro PLC, tak HMI zařízení a textové pole pro jejich pojmenování. U PLC zařízení je na výběr ze dvou možností, a to SIMATIC S7-1500 - typu CPU 1511C-1 PN



Obr. 3.4: Obrazovka pro volbu zařízení do projektu

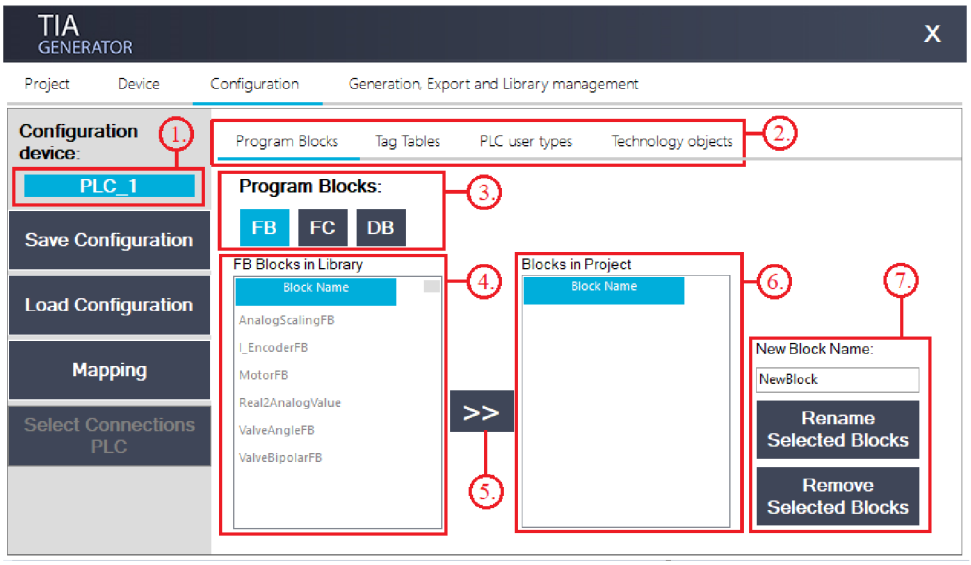
a SIMATIC S7-1200 - typu CPU 1212C DC/DC/DC. Jsou vybrány pravě tyto dvě, jakožto nejpoužívanější PLC hardware. Ve vygenerovaném projektu si programátor již snadno nahradí zařízení dle jeho specifikací. Zařízení se do projektu přidává metodou *AddDevice* třídy *ProjectTIA*.

V pravé části okna je tabulka, kde je výpis všech zařízení obsažených v projektu. Pro konfigurování jednotlivých zařízení je jej nutné označit a přesunout se do záložky Configuration. Druhou možností je dvojklikem vybrat příslušné zařízení.

3.2.3 Obrazovka konfigurace PLC zařízení

Pod záložkou Configuration se ve skutečnosti nacházejí dvě komponenty *TabControl*. Jedna pro PLC zařízení, zobrazené na Obr. 3.5, druhá pak pro HMI. Společnou částí je levé menu, kde pod označením číslem jedna je jméno konfigurovaného zařízení. Dále z tohoto menu je možné uložit dosavadní konfiguraci, či načíst již dříve rozpracovanou. Pro uložení konfigurace byla napsána metoda *Serializuj(List<DeviceTia>)*, obsažena ve třídě *Library Manager*. Naopak načtení konfigurace je umožněno metodou *Deserializuj()*. Dále lze z tohoto menu otevřít nové okno pro mapování, a to jak bloků, tak v případě HMI událostí obrazovek. Posledním tlačítkem tohoto menu je vybrání PLC zařízení, ze kterého se budou vybírat PLC tagy pro HMI vizualizaci. Toto tlačítko je přístupné pouze při konfigurování HMI zařízení.

Číslem dva je označeno menu pro PLC zařízení, kde je přes jednotlivé záložky možné spravovat bloky, tabulky tagů, uživatelsky definované typy a regulátory. Dále budou jednotlivé záložky popsány.



Obr. 3.5: Obrazovka pro konfiguraci PLC bloků

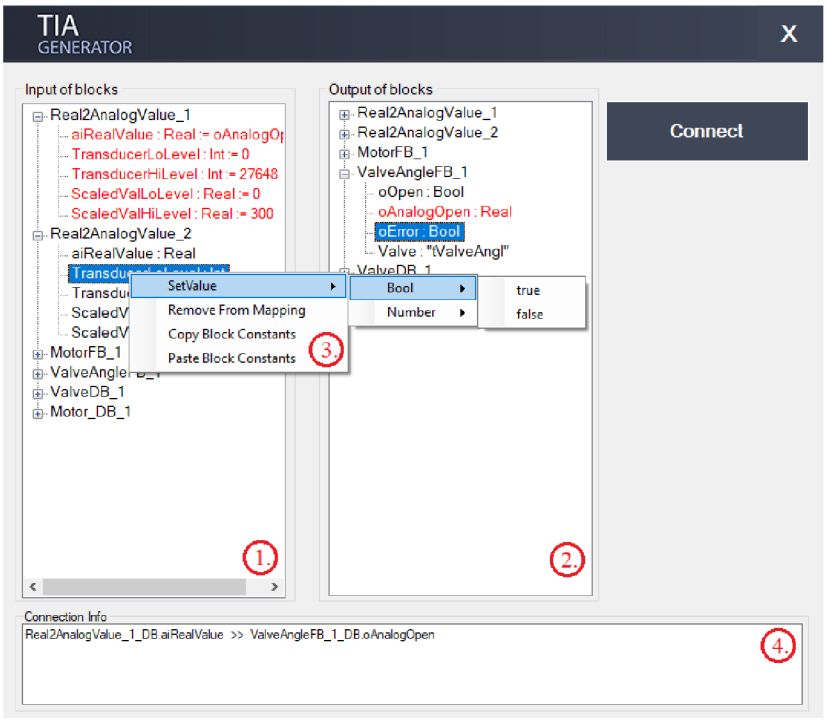
Záložka pro správu bloků

V záložce pro programové bloky je možné přidávat bloky z knihoven do projektu, kde se nachází:

3. Tlačítka pro přepínání typů bloků, které se zobrazují ve výpisu čtyři.
4. Výpis vybraných bloků obsažených v knihovně.
5. Tlačítko pro přidání bloků z knihovny do projektu metodou *AddBlock(PlcBlockTia)* třídy *ProgramBlocks*.
6. Výpis všech bloků obsažených v projektu.
7. Mazání a přejmenování bloků v projektu. Obě funkce obstarává třída *ProgramBlocks*, a to metodami *RemoveBlock(PlcBlockTia)*, respektive *Rename(PlcBlockTia, string)*.

Po přidání bloků, případně tagů je možné stisknout tlačítko *Mapping*, kterým se otevře nové formulářové okno z Obr. 3.6, kde:

1. Stromová struktura všech bloků v projektu s jejich vstupy, případně tabulky tagů s tagy, které jsou orientovány jako výstupní.
2. Obdobně jako předcházející výpis, jen jsou zde zobrazeny výstupy bloků a vstupy tagů.
3. Kontextové menu s možností nastavit konstantní hodnotu danému vstupu pomocí metody *SetConstant(Pin, string)*. Dále je možné smazat vytvořené spojení daného vstupu metodou *RemoveMaping(pin)*. Často využívanou položkou pro zadávání konstantních hodnot, např. pro převod analogové



Obr. 3.6: Obrazovka pro mapování bloků

veličiny, je kopírování a následné vkládání konstant mezi jednotlivými bloky. Za tímto účelem je vytvořena metoda *PasteConstants(PlcBlockTia)*, které je předán v parametru blok, z kterého se mají zkopírovat namapované konstantní hodnoty. Všechny zmíněné metody jsou obsaženy ve třídě *PlcBlockTia*.

4. Výpis namapovaných vstupů a výstupů.

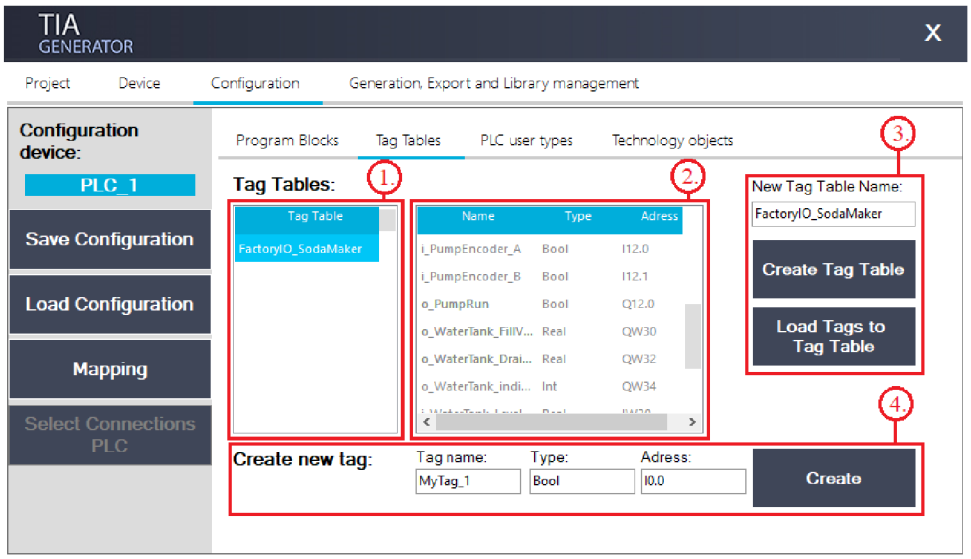
Jak lze dále vidět z Obr. 3.6, namapované vstupy/výstupy jsou zvýrazněny červenou barvou. Toho je docíleno díky metodě *IsMapped(Pin)*, pomocí které je zjištěno, zda je pin již namapován. Dále se za namapovanými vstupy zobrazí jméno proměnné nebo konstanty, která je na daný vstup přivedena. Toho je docíleno metodou *GetNameOfMappedPin(Pin)*, pomocí které je získáno jméno pinu přivedeného na daný vstup. U výstupních proměnných toto není realizováno, jelikož na výstup lze připojit neomezené množství pinů, což by způsobovalo nepřehlednost.

Vytvoření propojení vstupu s výstupem je docíleno vybráním požadovaných pinů a stisknutím tlačítka *Connect*. Mapování je poté provedeno metodou *MapInput(Pin, Pin)*, respektive *MapOutput(Pin, Pin)*.

Všechny zmíněné metody jsou opět obsaženy ve třídě *PlcBlockTia*.

Záložka pro tabulek tagů

Další záložka z Obr. 3.7 umožňuje vytvářet tabulky tagů s tagy, či nahrát tagy do tabulky z CSV souboru. Nachází se zde:



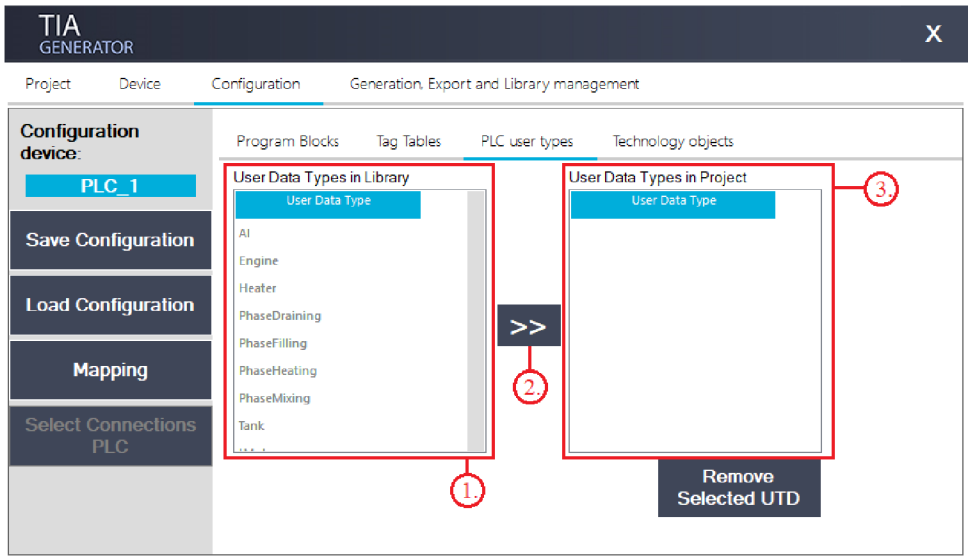
Obr. 3.7: Obrazovka pro konfiguraci PLC tagů

1. Seznam tabulek tagů v projektu.
2. Seznam tagů ve vybrané tabulce.
3. Menu pro vytvoření tabulky s požadovaným jménem pomocí metody *CreatTagTable(string)* třídy *PLCTia*, kde je upravena tato metoda od předka. Druhým tlačítkem tohoto menu je možné nahrát tagy z CSV souboru metodou *LoadTagsFromCSVFile(string)* třídy TagTable. CSV soubor musí být ve formátu z výpisu 5.5.
4. Menu pro vytvoření jednotlivých tagů a jejich přidání do tabulky pomocí metody *AddTag(pin)* třídy *TagTable*, představující danou tabulku.

Záložka pro uživatelsky definované typy

Tato záložka z Obr. 3.8 je velmi jednoduchá. Umožňuje pouze přidávat, popř. mazat UDT do/z projektu. Jmenovitě se zde nachází:

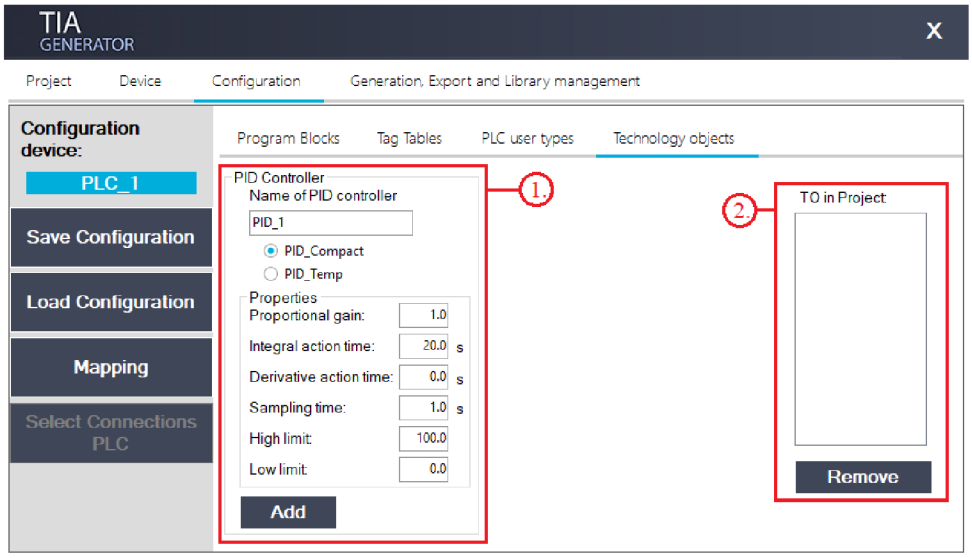
1. Seznam UDT v knihovně.
2. Přidání vybraných UDT z knihovny do projektu metodou *AddUDT(UserDataType)* z třídy *PLCTia*.
3. Seznam UDT v knihovně, pod kterým se nachází tlačítko pro smazání vybraných UDT v projektu. Mazání se provede metodou *RemoveUDT(UserDataType)*, opět z třídy *PLCTia*.



Obr. 3.8: Obrazovka pro konfiguraci UDT

Záložka pro technologické objekty

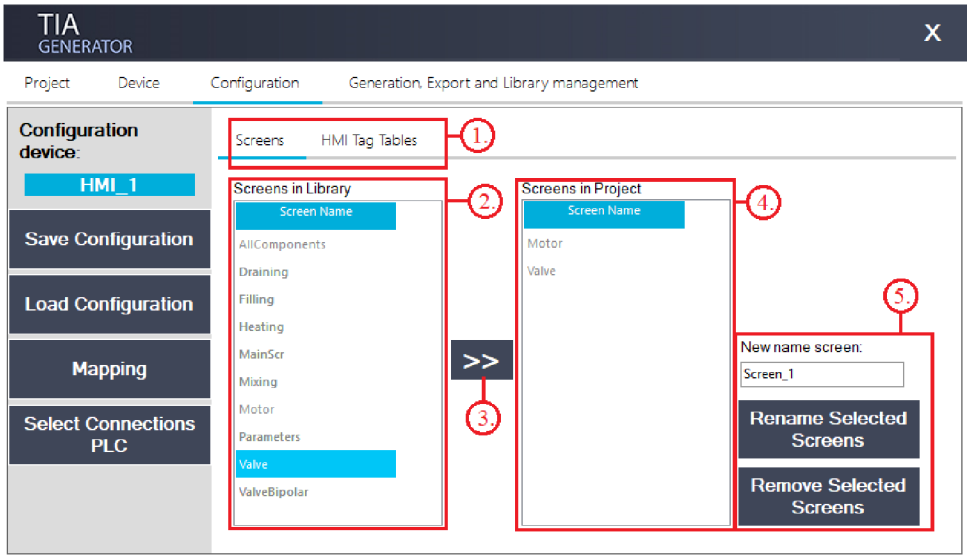
Na poslední záložce Obr. 3.9 pro PLC konfiguraci je možné vytvořit regulátor a nastavit jeho základní parametry. Menu pro vytvoření regulátoru s parametry je označeno rámečkem s číslem jedna. Pod číslem dva se nachází výpis všech regulátorů přidaných do projektu. Regulátor se vytvoří zadáním parametrů do jednoho z šesti konstruktorů třídy *TO_PID*. Následné přidání mezi technologické objekty je řízeno metodou *AddPID_Controller* třídy *TechnologyObjects*, které je předán regulátor v parametru.



Obr. 3.9: Obrazovka pro konfiguraci regulátorů

3.2.4 Obrazovka konfigurace HMI zařízení

Z Obr. 3.10 pod značením číslem jedna je patrné, že konfigurování HMI zařízení disponuje pouze dvěma záložkami.



Obr. 3.10: Obrazovka pro konfiguraci obrazovek

Záložka pro obrazovky

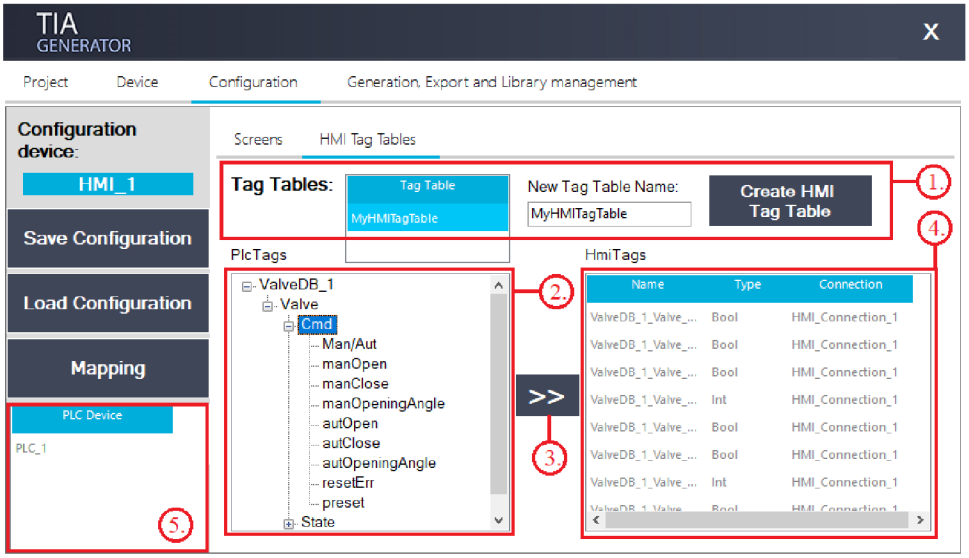
První ze záložek slouží pro správu obrazovek. Rozložení je obdobné jako u bloků nebo UDT. V následujícím seznamu jsou popsány jednotlivé funkce. Jmenované metody náleží třídě *HMITia*.

1. Záložky pro HMI konfiguraci.
2. Výpis obrazovek obsažených v knihovně.
3. Přidání obrazovky do projektu pomocí metody *AddScreen(ScreenTia)*.
4. Výpis obrazovek v projektu.
5. Menu pro přejmenování a smazání vybraných obrazovek z výpisu č. 4 pomocí metody *RenameScreen(ScreenTia, string)*, respektive *RemoveScreen(ScreenTia)*.

Po stisknutí tlačítka *Mapping* z levého menu se nejdříve otevře okno pro vybrání připojeného PLC, viz pátá očíslovaná položka z Obr. 3.11. Poté se otevře stejné formulářové okno jako je na Obr. 3.6. Rozdílem je, že v levém výpisu jsou obsaženy jednotlivé obrazovky s jejich událostmi, kterými disponují. V pravém výpisu jsou poté všechny bloky s jejich piny z vybraného PLC. Kontextové menu pak umožňuje pouze smazání namapované události. Pro zjednodušení mapování je možné přiřadit celou strukturu UDT, jenž je využita na dané obrazovce. Algoritmus metody *MappEvent(Pin, Pin)* třídy *ScreenTia* následně zajistí, že struktura UDT předaná jako druhý parametr této funkce, je namapována na odpovídající události.

Záložka pro HMI tagy

Druhá ze záložek je zobrazena na Obr. 3.11. Slouží pro vytváření HMI tabulek s HMI tagy. Nicméně tak jak je tomu i v samotném vývojovém prostředí, i zde dá uživatel nejspíše přednost tomu, že nechá vytvořit HMI tagy automaticky tím, že budou přiřazeny PLC tagy k událostem obrazovek. Proto ve většině případů se k této obrazovce uživatel ani nedostane. Jednotlivé objekty této obrazovky jsou popsány v následujícím seznamu.

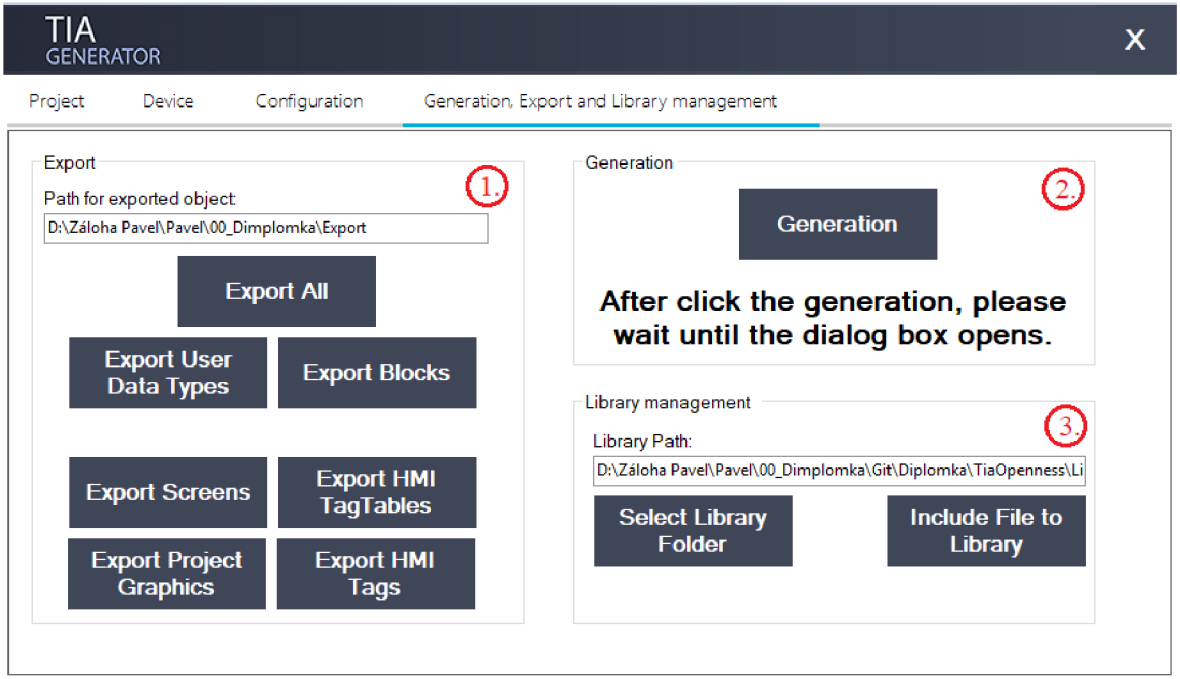


Obr. 3.11: Obrazovka pro konfiguraci HMI tagů

1. Možnost vytvoření HMI tag tabulky a výpis tabulek v zařízení. Pro založení tabulky slouží metoda *CreatTagTable(string)* třídy *HMITia*.
2. Výpis bloků s jejich piny ve vybraném PLC zařízení.
3. Přidání PLC tagů do tabulky. Algoritmus metody *AddTags(object)* třídy *HMITagTable* rozliší, zda se jedná o samotný pin, strukturu pinů nebo blok. Následně vytvoří HMI tagy z předaného pinu, všech pinů z předané struktury, či z pinů předaného PLC bloku.
4. Výpis všech HMI tagů ve vybrané HMI tabulce.
5. Okno pro výběr PLC zařízení, které je připojené k danému HMI. Slouží pro získání PLC objektů do výpisu č. 2.

3.2.5 Obrazovka pro exportování objektů a generování projektu

Poslední záložka hlavního menu z Obr. 3.12 slouží ke generování projektu, exportování objektů z TIA Portalu a správu objektů v knihovně. Obsah této záložky je popsán v následujícím výčtu.



Obr. 3.12: Obrazovka pro exportování objektů a generování projektu

1. Sekce pro export všech objektů či jen jednotlivě vybraných, do adresáře uvedeného v textovém poli. Exportování je zajištěno příslušnými metodami obsažených v obslužných třídách *TIAService, PLCService* a *HMIService*.
2. Po stisku tlačítka *Generation* v této sekci se zahájí generování projektu. Tento proces je již popsán v samostatné kapitole 5.3.
3. Správa knihovny, kde v textovém poli je adresář, ve kterém se nacházejí zdrojové soubory. Tlačítkem *Select Library Folder* lze vybrat jinou složku, např. nově vytvořené knihovny, což však není doporučováno. Druhým tlačítkem *Include File to Library* v této sekci je možné přidávat zdrojové XML soubory do knihovny. Metodě *IncludeFiles(string[])* třídy *LibraryManager* jsou v parametru předány cesty k souborům. Metoda následně zajistí, aby byly nově přidané soubory vhodně rozřazeny.

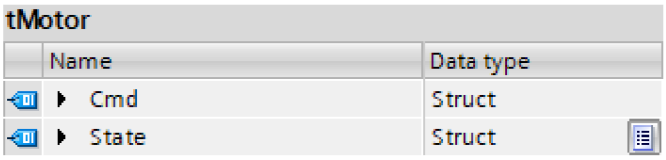
4 Vytvořená referenční sada modulů a ověření funkčnosti

V rámci této práce je nezbytné vytvoření referenční sady modulů zařízení pro PLC a HMI ve vývojovém prostředí TIA Portal, aby mohla být demostrována a ověřena funkčnost vytvořené aplikace pro generování programů. Dále byly vytvořeny příklady použití v simulačním nástroji Factory I/O. V této kapitole budou tyto náležitosti popsány.

4.1 Vytvořená referenční sada modulů

4.1.1 Sada modulů pro PLC

Veškeré moduly jsou vytvořeny jako funkční bloky. Jednotlivé bloky jsou zároveň naprogramovány tak, aby bylo umožněno jejich co nejuniverzálnější použití. Pro moduly zařízení jsou vytvořeny obdobné uživatelsky definované datové typy - UDT. Příklad pro UDT motoru je uveden na Obr. 4.1. Pro ostatní moduly zařízení je tato struktura stejná, liší se jen jejich obsah. Použitím UDT lze následně moduly snadno ovládat (struktura Cmd) či vyhodnocovat jejich stav (struktura State). To je důležité nejen pro jejich následné ovládání v automatickém režimu, ale i pro vytváření vizualizace, díky čemuž je jednodušší mapování události ve vytvořené aplikaci.

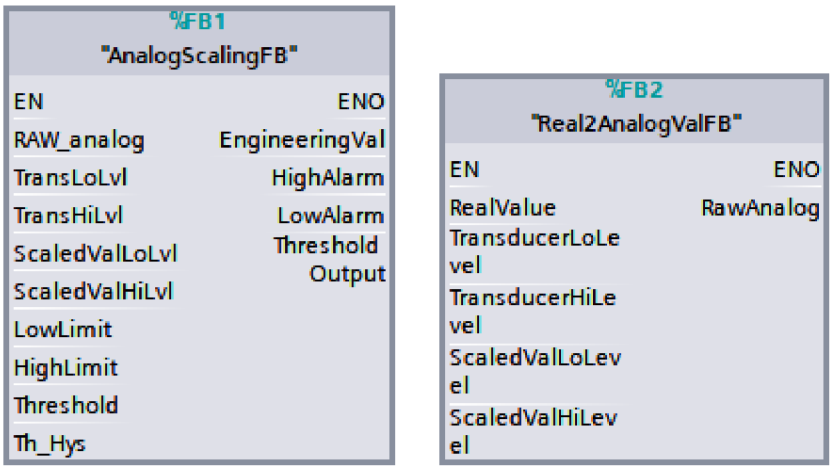


Obr. 4.1: Příklad UDT pro motor

Převod analogové veličiny

Pro převod analogové veličiny byly vytvořeny bloky jak pro převod analogové hodnoty na inženýrskou, tak i opačně. Oba tyto bloky jsou na Obr. 4.2. Převod analogové hodnoty na vypovídající reálné jednotky je zprostředkován pomocí funkčního bloku *AnalogScalingFB* vlevo, kde:

- RAW_analog – Analogová hodnota ze vstupu PLC, nazývaná také jako tzv. syrový analog. Přepočtená hodnota je přivedena na výstup EngineeringVal.



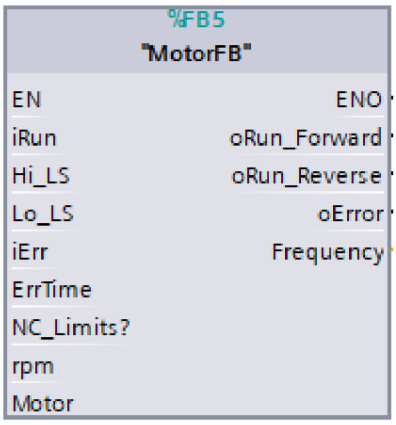
Obr. 4.2: Převod analogové hodnoty na inženýrskou

- TransLoLvl – Minimální hodnota analogového signálu. Standardně je pro unipolární signál rovna 0, což je i přednastavenou hodnotou tohoto vstupu.
- TransHiLvl – Maximální hodnota analogového signálu. Standardně je pro unipolární signál rovna 27 648, což je i přednastavenou hodnotou tohoto vstupu.
- ScaledValLoLvl – Odpovídající minimální reálná hodnota.
- ScaledValHiLvl – Odpovídající maximální reálná hodnota.
- LowLimit – Dolní limit reálné hodnoty. Po poklesu pod tuto hodnotu se aktivuje výstup LowAlarm.
- HighLimit – Horní limit reálné hodnoty. Po překročení této hodnoty se aktivuje výstup HighAlarm.
- Threshold – Mez reálné hodnoty při jejímž překročení se aktivuje výstup Threshold Output. Uplatní se např. pro použití analogového vstupu jako limitního snímače. Rozdíl oproti spodnímu a hornímu alarmu je v možnosti použití hystereze (poslední vstup bloku).
- Th-Hys – Hystereze pro aktivaci/deaktivaci výstupu Threshold Output. Používá se pro odstranění kmitání výstupu Threshold Output při kolísání reálné hodnoty.

Obrácený převod z reálné hodnoty na analogovou je distribuován pomocí funkčního bloku *Real2AnalogValFB* vpravo na Obr. 4.2. Tyto vstupy již popsány nebudou, neboť jsou totožné s popisem výše. Jediným rozdílem je, že na vstup je přivedena reálná hodnota a výstupem je „syrový analog“.

Blok pro ovládání motoru

Ovládání motoru umožňuje funkční blok *MotorFB* z Obr. 4.3, kde jsou zobrazeny i jeho vstupy/výstupy, jejichž funkce jsou popsány v následujícím výčtu.



Obr. 4.3: Blok pro ovládání motoru

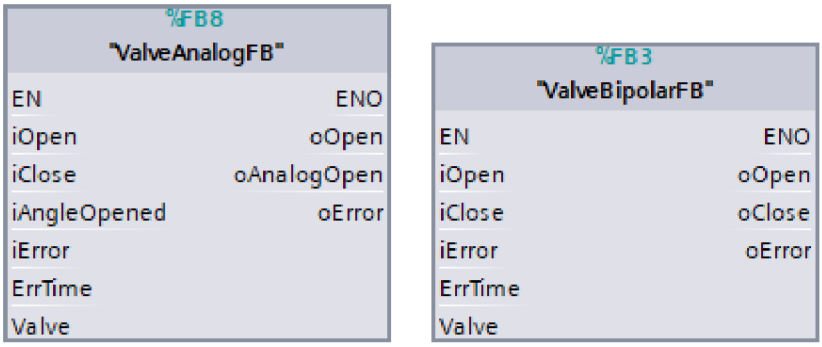
- iRun – Zpětná vazba o chodu motoru.
- Hi_LS – Horní limit, např. pro lineární posuv. Po dosažení vypne motor.
- Lo_LS – Dolní limit. Po dosažení vypne motor.
- iErr – Vstupující chyba, zablokování chodu.
- ErrTime – Chybový čas. Po příkazu pro spuštění motoru musí být po jeho uplynutí obdržena zpětná vazba o jeho chodu, jinak je hlášen chybový stav.
- NC_Limits? – Horní a dolní limit jsou snímače typu NC (normally closed).
- rpm – Požadované otáčky za minutu, přepočteno na frekvenci a přivedeno na výstup Frequency.
- Motor – UDT tMotor pro ovládání a vyhodnocování stavů.
- oRun_Forward – Výstup pro motor - normální chod motoru. V případech kdy není motor reverzován, je použit pouze tento výstup.
- oRun_Reverse – Výstup pro motor - obrácený chod.
- oError – Chyba motoru.

Blok pro ovládání ventilu

Pro ovládání ventilu vznikly dva referenční bloky. Prvním na Obr. 4.4 vlevo, je pro monostabilní ventil a vpravo pro bistabilní ventil. Monostabilní ventil navíc umožňuje analogové otevírání ventilu. Následující popis bude koncipován pro oba bloky i přesto, že se bloky nepatrně liší.

- iOpen – Zpětná vazba od ventilu - ventil otevřen.
- iClose – Zpětná vazba od ventilu - ventil zavřen.
- iAngleOpened – Analogová zpětná vazba od ventilu - procentuální otevření ventilu.
- iError – Vstupující chyba - zablokování chodu.

- ErrTime – Chybový čas, po jehož uplynutí musí po příkazu otevření/zavření ventilu být obdržena zpětná vazba o jeho stavu, jinak je hlášen chybový stav.
- Valve – UDT tValveAnalog, respektive tValveBip pro ovládání a vyhodnocování stavů.
- oOpen – Výstup pro ventil - otevření ventilu.
- oAnalogOpen – Analogový výstup pro ventil - procentuální otevření ventilu.
- oClose – Výstup pro bipolární ventil - zavření ventilu.
- oError – Chyba ventilu.



Obr. 4.4: Blok pro ovládání ventilu

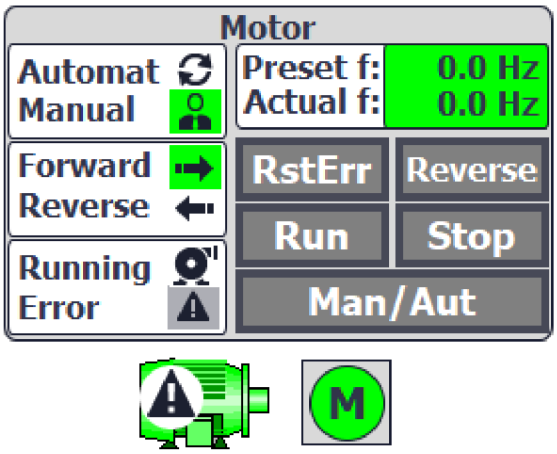
4.1.2 Sada modulů zařízení pro HMI

Byly vytvořeny vizualizace pro moduly zařízení. Pro každý modul byla vytvořena obrazovka s jeho ovládáním a reprezentací stavů pomocí UDT daného modulu. Při generování je možné pro každý modul vytvořit vlastní obrazovku s ovládáním a následně je přesouvat dle libosti a tvořit vlastní kompozice.

Na Obr. 4.5 lze vidět vizualizaci pro motor, která se skládá z tabulky pro ovládání a reprezentaci stavů a grafikou pro motor, umístěnou pod tabulkou.

V pravé dolní části tabulky se nacházejí tlačítka pro ovládání motoru. V levé části jsou pak zeleně podbarveny aktivní stavy motoru. Pod tabulkou se nachází grafika motoru a jeho schématická značka. Ty svým zeleným podbarvením reprezentují běh motoru (na Obr. 4.5). Pokud se motor dostane do chybového stavu, rozblikají se červeně, obdobně jako bipolární ventil na Obr. 4.6. Záleží pouze na preferencích uživatele, zda ve vizualizaci použije grafiku motoru, či jeho schématickou značku.

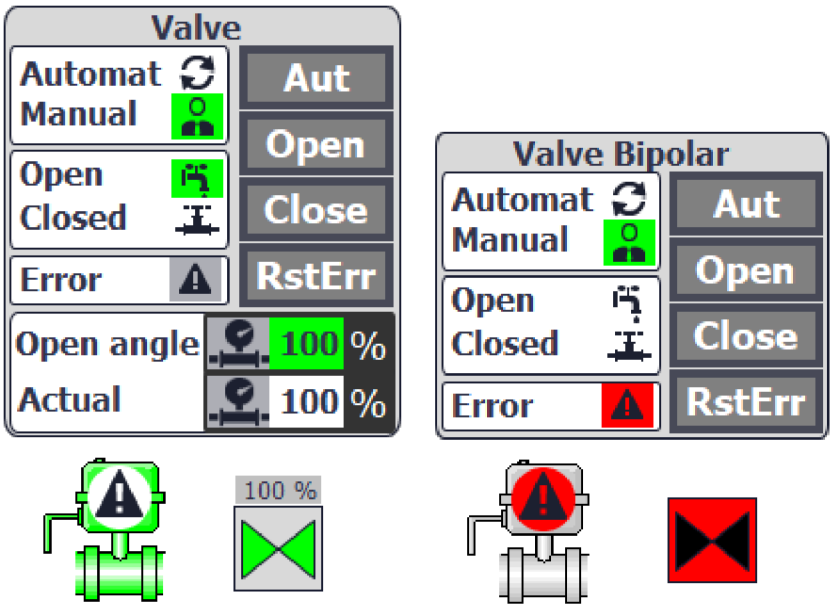
Na Obr. 4.6 jsou zobrazeny vizualizace pro ventily. Jejich design je podobný jako u motoru. Vlevo se nachází tabulka s grafikou pro analogový ventil. Procentuální úhel otevření analogového ventilu lze nastavit ve spodní části tabulky zapsáním hodnoty do zeleně podbarveného pole. Hodnota pod ní (Actual), či nad schematickou značkou pro tento ventil, pak ukazuje aktuální otevření ventilu. Při inicializaci je hodnota nastavena na hodnotu -1 a spíná se pouze digitální výstup pro ventil



Obr. 4.5: Vizualizace pro ovládání motoru

příkazy *Open* a *Close*. Jakmile se hodnota v zeleném poli přepíše na nezáporné číslo, aktivuje se kromě digitálního výstupu i analogový. Lze ho následně ovládat vepsanou hodnotou do zeleného pole nebo rovněž příkazy pro otevřít a zavřít, jimiž se nastavuje na hodnotu 100 %, respektive 0 %.

V pravé části obrázku se nachází vizualizace pro bipolární ventil, který se aktuálně nachází v chybovém stavu. V takovém případě se červeně rozblikají objekty, které jsou na obrázku podbarveny červeně, což je pro oba ventily stejné.



Obr. 4.6: Vizualizace pro ovládání ventilu

4.2 Předpis pro generování

Pro předpis byl zvolen XML jazyk, který je navržen pro uchovávání strukturovaných dat. Předpis slouží zároveň pro uložení dat konfigurace a vznikl serializací jednotlivých objektů, čímž je dáno, že struktura XML předpisu kopíruje tu, dle které je navržena aplikace, viz Obr. 3.2. Z výpisu E.1 v příloze E lze vidět, že kořenovým elementem je *Project*, jehož potomkem je vždy *DeviceTia*, který v atributu *type* nese informaci, zda se jedná o PLC (*PLCTia*) nebo HMI (*HMITia*). Potomci *DeviceTia* se dále liší dle typu zařízení. Pro oba typy zařízení jsou však stejné elementy určující jejich jméno, označení zařízení, typové číslo a tabulku tagů, pokud jí zařízení disponuje. V elementu pro tabulku tagů *TagTable* je atributem *Name* určeno jméno tabulky. Jediným potomkem *TagTable* je *Tags*, který v jeho potomcích obsahuje kolekci tagů. Tagy jsou dány elementem *Pin*, v jehož atributu *type* je určeno, že se jedná o tag (*Tag*), respektive HMI tag (*HMITag*).

U PLC zařízení následuje *ProgramsBlocks*, který pod elementem *PlcBlocks* sdružuje bloky. Blok je vždy reprezentován elementem *PlcBlockTia*, v jehož atributu *type* je určen typ bloku - *FB,DB,FC*. Potomky *PlcBlockTia* jsou *FileInfo, Pins, MapList*. *FileInfo* v atributech uchovává informaci o zdrojovém souboru a každý objekt vycházející z XML disponuje tímto elementem. U *Pins* je atributem *type* stanoven typ pinu, který v tomto případě nabývá hodnot *FunctionPin* nebo *DataItem* a reprezentuje vstupy/výstupy bloku. *MapList* obsahuje informace o připojených proměnných na konkrétní piny bloku. Dalším sourozencem *ProgramsBlocks* je *UserDataType* pro uživatelsky definované typy, který v potomcích *Members* obsahuje proměnné daného UDT. Posledním potomkem *DeviceTia* pro PLC je element *TechnologyObjects*, jenž archivuje regulátory pod elementem *PIDs*. Regulátory jsou dány elementem *TO_PID*, v jehož potomcích jsou parametry regulátoru.

Pro HMI zařízení je XML struktura jednodušší, neboť kromě tabulek tagů obsahuje pouze obrazovky (*ScreenTia*) pod elementem *Screens*. *ScreenTia* mimo jiné disponuje opět potomkem *FileInfo*. Dále obsahuje potomky reprezentující události obrazovky - *Events* a namapované události na PLC proměnné - *MapList*. Jednotlivé události jsou označovány elementem *ScreenEvent*.

4.3 Ověření funkčnosti

Pro účely ověření funkčnosti generovaného projektu byly zvoleny simulační nástroje S7-PLCSIM a Factory I/O. Factory I/O umožňuje propojení se simulátorem z TIA Portalu S7-PLCSIM a zpracování vstupů a výstupů PLC. Byly vytvořeny dvě simulační úlohy, a sice tvorba sody a třídička materiálů.

4.3.1 Soda Maker

Popis demonstračního příkladu

Jedním z bodů zadání je vygenerování demonstračního příkladu, konkrétněji buňky barmana. Byla vybrána buňka „SodaMaker“, neboli „sodovač“. Pro tuto buňku byl pomocí aplikace vytvořen předpis pro generování, který je přiložen jako příloha v elektronické verzi práce na CD pod názvem *FactoryIO_SodaMakerComplete.xml*. Pro generování programu byly použity referenční moduly z knihovny:

- ValveAnalogFB – 5x pro analogové ovládání ventilů.
- MotorFB – 1x pro ovládání čerpadla.
- i_EncoderFB – 1x pro zpětnou vazbu od motoru.
- Real2AnalogFB – 5x pro převod reálné hodnoty otevření ventilu na analogovou. Použitý pro výstup *oAnalogOpen* bloku *ValveAnalogFB*.
- AnalogScalingFB – 3x pro měření hladiny a 1x pro měření průtoku za účelem převodu analogové hodnoty na inženýrskou.
- Pro HMI bylo použito pět obrazovek pro ventil a jedna pro motor.

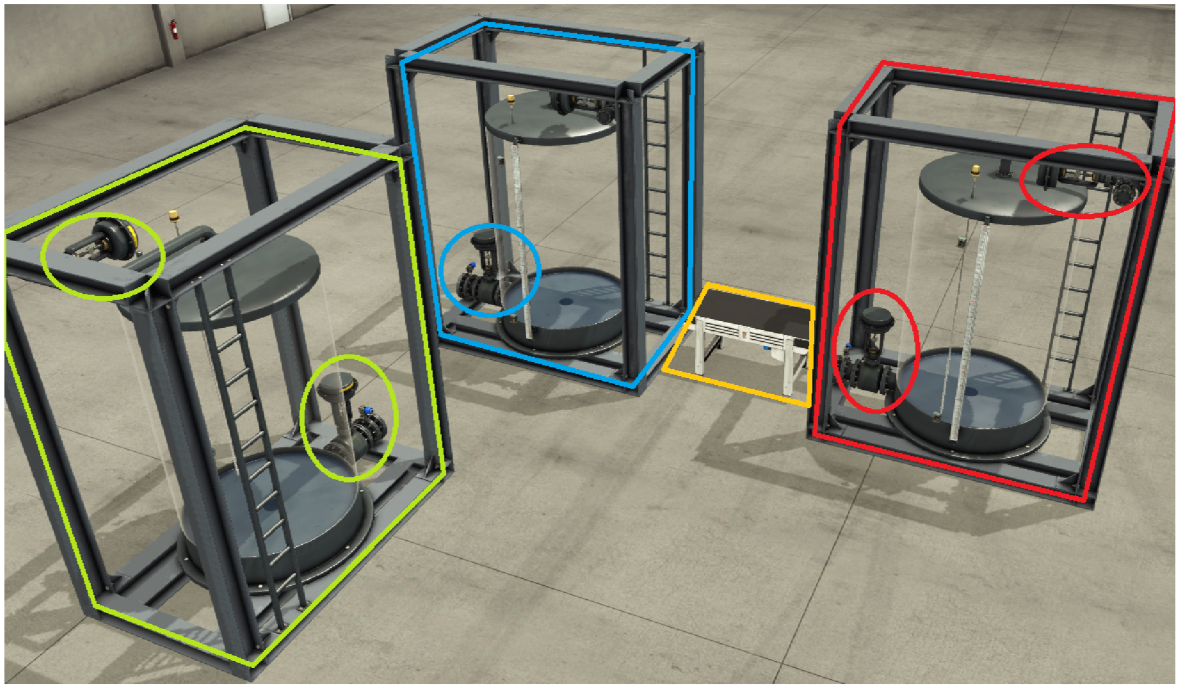
Pro demonstraci prostřednictvím Factory I/O byla tato buňka mírně zjednodušena tím, že se neuvažoval dekompresní ventil. Buňka tedy obsahuje celkem tři ventily, tři nádržky (s vodou, plynem a směšovací), jedno čerpadlo, jeden průtokoměr a tři analogové snímače hladin. Byly přidány další dva napouštěcí ventily pro možnost naplnit nádržky s vodou a plynem i v simulaci. Konkrétní uspořádání v simulačním nástroji zachycuje Obr. 4.7.

Červeně je orámován tank simulující nádržku s vodou s jeho napouštěcím ventilem umístěným nahoře a vypouštěcím ventilem dole. Zeleně je obdobně orámován tank pro plyn s jeho ventily. Modrou barvou je ohraničen směšovací tank, který má pouze vypouštěcí ventil. Jeho napouštěcí ventil je představován vypouštěcími ventily zbylých dvou tanků. Všechny ventily v tomto příkladu jsou analogové, aby bylo možné ověřit funkčnost analogového ovládání ventilů.

Oranžovou barvou je zvýrazněn dopravníkový pás, který v tomto případě simuluje čerpadlo. Ačkoliv se toto řešení může zdát směšné, Factory I/O nedisponuje velkým počtem aktuátorů, nicméně pro účely ověření funkčnosti je to dostačující. Čerpadlo i pás jsou ovládány stejným funkčním blokem pro motor.

Výsledek generování

Po vygenerování projektu bylo nutné pouze manuálně přidat volání bloku *FB_Call's* z *Main OB1* a definovat startovací obrazovku. Po těchto dvou triviálních úkonech bylo ověřeno, že vygenerovaný projekt je plně funkční v manuální režimu.



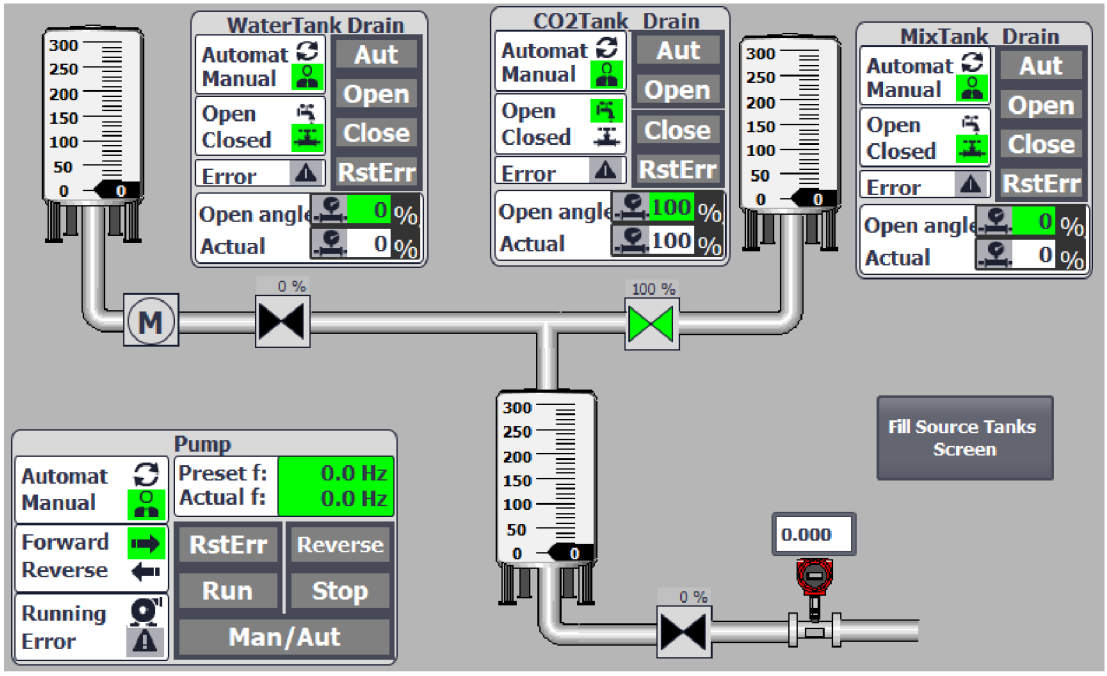
Obr. 4.7: Simulace SodaMaker ve Factory I/O

Výpis vygenerovaného a neupravovaného bloku *FB_Call's* s namapovanými bloky je přiložen v elektronické příloze této práce pod názvem *FB_Call's_SodaMaker.pdf*. Dále byly přeskládány komponenty obrazovek, aby bylo usnadněno ovládání jednotlivých modulů. Opět nebyly vygenerované komponenty nijak upravovány, pouze přesunuty. Dále byly přidány objekty pro potrubí a pro znázornění analogových veličin. Výsledná obrazovka je zobrazena na Obr. 4.8. Vygenerovaný projekt včetně vytvořené simulace je součástí přílohy na přiloženém CD.

4.3.2 Třídička materiálů

Popis demonstračního příkladu

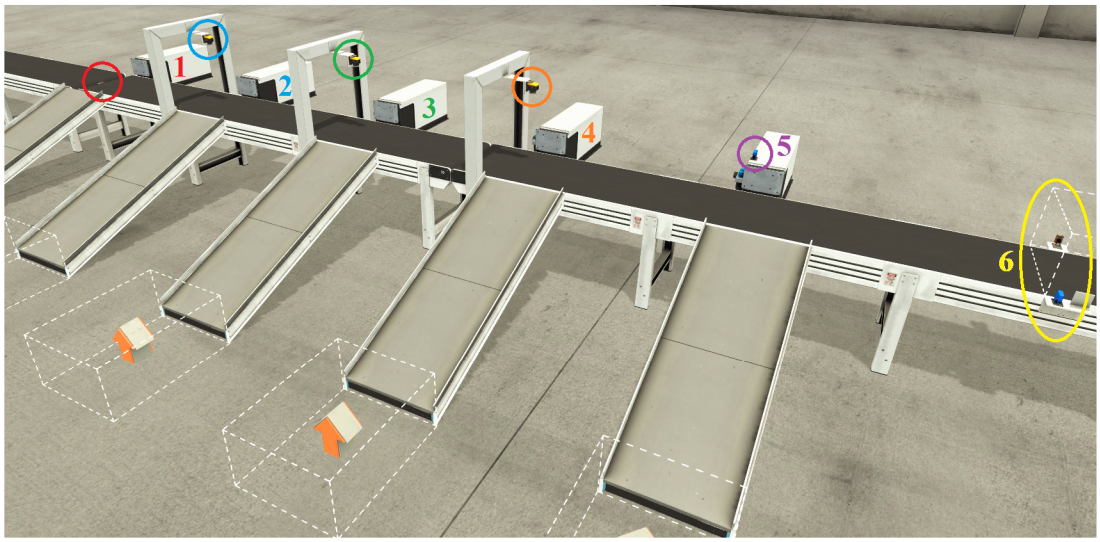
Druhým demonstračním příkladem pro ukázku fungování vytvořené aplikace je třídička materiálů. Simulátor třídičky byl opět vytvořen ve Factory I/O a je zobrazen na Obr. 4.9. Skládá se z pěti pneumatických posunovačů, označovaných také jako Pusher, které jsou opatřeny dvěma limitními snímači indikující jejich stav (otevřené/zavřené). Pushery jsou označeny barevně odlišenými čísly 1 až 5 a každému náleží snímač detekující daný druh materiálu. Tříděné materiály jsou k vidění v příloze F na Obr. F.1. Pokud by se některý z materiálů dostal za poslední posunovač, je třídička opatřena limitními snímači označenými číslem šest. Pro posun materiálů jsou použity celkem tři dopravníkové pásy s inkrementálními enkodéry obstarávající zpětnou vazbu o chodu.



Obr. 4.8: Vizualizace SodaMaker

Pro generování programu byly použity referenční moduly z knihovny:

- ValveAnalogFB – 5x pro digitální ovládání pneumatických ventilů posunovačů.
- MotorFB – 3x pro dopravníkové pásy.
- i_EncoderFB – 3x pro zpětnou vazbu od motorů.
- Pro HMI bylo použito pět obrazovek pro ventil a tři pro motor.



Obr. 4.9: Simulace třídičky ve Factory I/O

Výsledek generování

Po vygenerování projektu bylo nutné opět obstarat pouze sekvenci úkonů pro přidání volání bloku *FB_Call's* a definování startovací obrazovky. Výpis bloku *FB_Call's* pro tento příklad je opět přiložen v elektronické příloze této práce pod názvem *FB_Call's_Sorter.pdf*.

Předpis pro generování projektu pro třídičku je přiložen jako příloha v elektronické verzi práce na CD pod názvem *FactoryIO_Sorter.xml*. V tomto předpisu jsou mimo jiné namapovány limitní snímače na automatický chod motorů. Obdobně jsou pak namapovány vstupy jednotlivých snímačů rozpoznávající materiály na automatické otevírání přidružených posunovačů. I když žádaným výsledkem generování je funkční manuální režim, díky tomuto namapování je k dispozici bez dalších úprav i automatický režim pro třídění materiálů.

5 Dokumentace aplikace

5.1 Naprogramované třídy pro konfiguraci projektu

V této kapitole budou popsány vytvořené třídy, jejich účel a nejzajímavější vlastnosti. Popis nebude věnován všem metodám kvůli jejich velkému počtu, ale vždy budou vybrány jen ty, které jsou shledány jako zajímavé či důležité. Nejdříve se práce bude věnovat třídám pro práci s projektem či pro správu zdrojových souborů, následně třídám pro PLC zařízení, HMI zařízení a jejich objekty. Nakonec budou rozebrány obslužné třídy vykonávající komunikaci s vývojovým prostředím TIA Portal, ve kterých se využívá funkcí z knihoven TIA Openness. Vytvořený program včetně všech tříd, metod a návazností nejlépe vystihuje Class diagram, obsažený v příloze A.

Z diagramu lze vyčíst, že se často u názvů tříd vyskytuje postfix Tia. Díky němu je lepší přehlednost a orientace v programu, neboť třídy z knihovny TIA Openness disponují stejnými názvy tříd, jen se nacházejí v jiném jmenném prostoru. Díky tomuto postfixu je na první pohled patrné, že se jedná o objekt vytvořený v rámci aplikace a naopak.

5.1.1 Výčtové typy

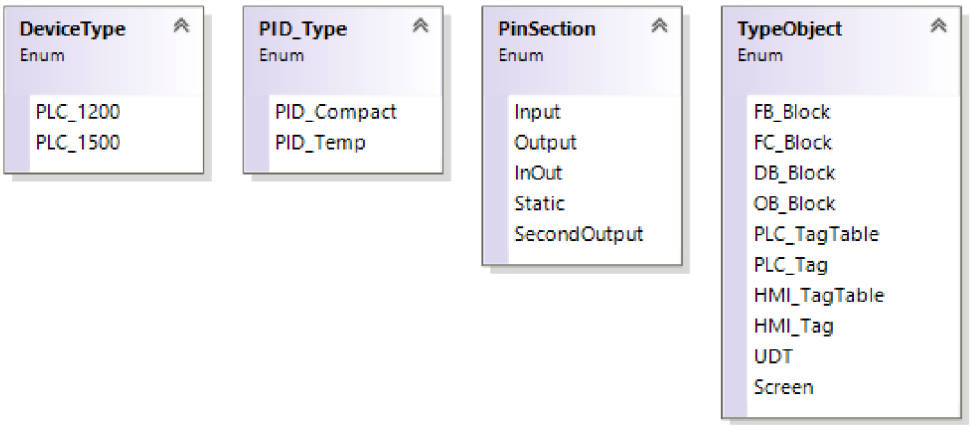
Při programování aplikace byly vytvořeny výčtové typy, které jsou k vidění na Obr. 5.1. *DeviceType* slouží pro rozeznání typu PLC zařízení. *PID_Type* rozeznává dva druhy PID regulátorů, které lze vytvořit. *PinSection* představuje vstupně/výstupní orientaci proměnných, přičemž *SecondOutput* se používá při mapování výstupů, pokud je výstup bloku namapován vícekrát. Nejrozsáhlejší výčtový typ *TypeObject* odkazuje na typ objektu. Využívá se především v rámci třídy *LibraryManager* a *Entity*, s jehož pomocí se rozlišuje k jakým objektům náleží zdrojové XML soubory.

5.1.2 Třídy spravující projekt a zařízení

Třída ProjectTia

Tato třída zastřešuje celý aplikační projekt a uchovává všechny potřebné nakonfigurované objekty pro následné generování projektu do TIA Portalu. Na Obr. A.1 lze vidět metody této třídy. Atributy a vlastnosti jsou uvedeny v následujícím seznamu:

- myProject : Project – Instance typu *Project* z knihovny *Siemens.Engineering*, která reprezentuje otevřený projekt ve vývojovém prostředí TIA Portal.



Obr. 5.1: Výčtové typy

- MyDevices : List<DeviceTia> – Seznam všech zařízení obsažených v projektu.
- MyTiaPortal : TiaPortal – Vlastnost typu *TiaPortal* z knihovny *Siemens.Engineering*, jehož instance odpovídá spuštěnému vývojovému prostředí Tia Portal.
- SubnetCollection : SubnetCollection – Kolekce připojených zařízení do sítě.

Metody *CloseProject, ConnectProject, DiposeProject, NewProject, OpenProject a SaveProject* slouží pro práci s projektem v TIA Portalu. Nejdůležitější metodou je metoda *Generate()*, která generuje celý nakonfigurovaný projekt do TIA Portalu.

Třída DeviceTia

Jedná se o abstraktní třídu, která sdružuje společné vlastnosti pro HMI a PLC zařízení. Přehled této třídy je zachycen na Obr. A.1.

DeviceTia obsahuje několik metod pro nastavení parametrů a vlastností zařízení. Pro generování programu jsou podstatné dvě. První je *GetSoftware()*, jenž vrací buďto třídu *PlcSoftware* nebo *HmiTarget* v závislosti na druhu zařízení. Pomocí těchto tříd se přistupuje k jednotlivým objektům zařízení v TIA Portalu. Druhou metodou je *Import()*, která importuje zařízení se všemi jeho nastavenými vlastnostmi do TIA Portalu. Obě tyto metody jsou abstraktní a jejich funkcionalita se definuje ve zděděných třídách.

Třída PLCTia

Uchování objektů a jejich konfiguraci pro PLC zařízení zajišťuje třída *PLCTia*, která je potomkem třídy pro obecné zařízení, viz UML diagram 3.2. Z přehledu této třídy na Obr. A.2 je patrné. že oproti předkovi obsahuje navíc vlastnosti:

- ProgramBlocks : ProgramBlocks – Obsahuje všechny programové bloky v zařízení.
- TechnologyObjects : TechnologyObjects – Správa technologických objektů.
- UserDataType : List<UserDataType> – Kolekce uživatelsky definovaných dat.

Kromě metod pro spravování uživatelsky definovaných typů obsahuje tato třída i privátní metody, které se volají při importování zařízení do TIA Portalu a kontrolují zda je konfigurace v pořádku a může dojít k importu. Jedná se o metody:

- CahngeBlockForPLC1200() – Pokud je generované zařízení typu S7-1200, kontroluje se, zda zdrojové XML soubory bloků pocházejí (byly exportovány) z PLC S7-1500. Pokud tomu tak je, musejí se z těchto souborů odstranit dva elementy, a sice *IsRetainMemResEnabled* a *MemoryReserve*, které S7-1200 nepodporuje a pokud v souborech zůstanou, import skončí chybou.
- CheckTypesForGeneration() – Zde se provádí kontrola zda je v jednotlivých blocích obsažen uživatelsky definovaný typ. Pokud ano, kontroluje se zda je obsažen v zařízení. Aplikace uživatele upozorní, že tento UDT nebyl v zařízení nalezen. Opět pokud by tomu tak nebylo, importování do TIA Portalu by se nedokončilo vzniklou chybou při importu.
- SortUdt() – Pouze se kontroluje zda některý z UDT neobsahuje další UDT. Pokud ano, zařadí se na poslední místo kolekce, aby byl importován jako poslední. To z toho důvodu, že při importu musí být již znám datový typ UDT, jenž je součástí jiného UDT.

Metoda *Import()* dále řeší pořadí importů jednotlivých objektů. Z výše uvedeného popisu je zřejmé, že v době importování do TIA Portalu musí být známy již všechny datové typy, a tudíž se UDT musejí importovat jako první. Následně se importují technologické objekty. Jako poslední se importují bloky.

Třída HMITia

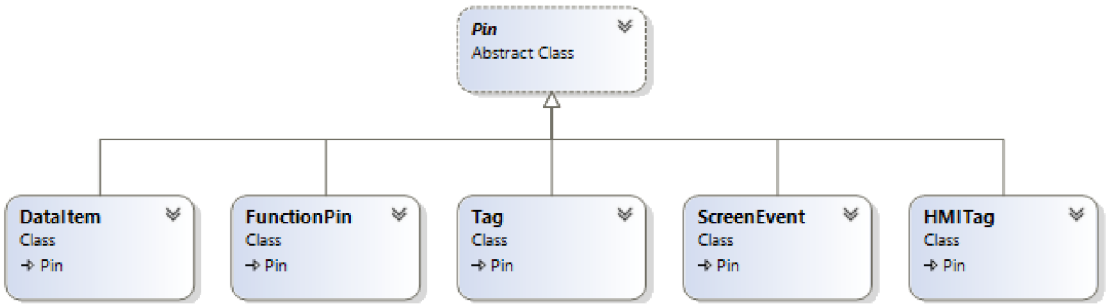
V této třídě se uchovávají informace pro generování HMI zařízení. Vlastnosti pro výšku a šířku obrazovky se vyčítají ze zdrojového souboru a jsou známy až v době generování programu. Jelikož se při vytvoření nového zařízení automatický vytváří i jeho prázdná obrazovka, tak se tato obrazovka exportuje a vyčte se z příslušných elementů XML souboru velikost zařízení a uloží do vlastností. Tyto hodnoty jsou poté použity pro úpravu velikostí objektů na obrazovce, viz kap. 5.1.8. Obdobně jako u třídy *PLCTia*, i zde je v metodě *Import()* důležité pořadí vykonávajících se funkcí. Nejdříve je nutné, aby byla volána metoda *LoadScreensEvenets()*, která zajistí přepsání namapovaných událostí v obrazovkách do XML souborů. Mimo jiné

vytvoří i příslušné HMI tagy, pokud ještě nejsou součástí zařízení. Až následně se mohou importovat i HMI tag tabulky.

5.1.3 Třídy pro piny

V rámci konfigurování projektu je nutné uchovávat proměnné jednotlivých bloků, tagů či HMI události. Aby bylo možné jednotlivé piny jednoduše mapovat dohromady, např. aby se na vstup funkčního bloku mohl v aplikaci připojit tag, je vhodné, aby proměnné byly objekty stejného typu. Avšak jednotlivé proměnné jsou odlišné a potřebují uchovávat různé informace.

Veškeré proměnné tedy zastřešuje abstraktní třída *Pin*, jak je vidět z Obr. 5.2. Tím je dosaženo požadavku, aby proměnné byly objekty stejného typu. Následně se definují jednotlivé třídy zděděné z této abstraktní třídy. Jedná se o třídy *DataItem*, *FunctionPin*, *ScreenEvent*, *HMITag* a *Tag*.



Obr. 5.2: Třídy pro piny

Třída Pin

Jak již bylo řečeno jedná se o abstraktní třídu, která obsahuje společné vlastnosti a metody pro veškeré proměnné v aplikaci. Obsahuje tři atributy, a to:

- dataType : string – Datový typ daného pinu (proměnné).
- name : string – Jméno pinu.
- pinSection : PinSection – Výčtový typ sekce pinu. Obsahuje informaci o tom, zda se jedná o vstupní, výstupní, nebo vstupně-výstupní proměnnou. Výčtový typ je popsán v kap. 5.1.1.

Třída *Pin* obsahuje abstraktní metodu *GetMapName()*, kterou si odvozené třídy přepisují tak, aby vracely své celé jméno. Metoda vrací typ *List<string>*. Tento typ se vrací z prostého důvodu. Pokud je proměnná uvnitř struktury, její název se skládá jednak z jejího jména, tak i z jména struktury, která daný pin vlastní. Struktura jenž vlastní tento typ, muže být obsažena opět v další struktuře a takto stále dokola.

Jednotlivá jména struktur se pak vkládají do listu, kde je poté obsažena celá cesta k proměnné. Byla použita kolekce list, jelikož při generování programu a přepisování XML souborů jsou vždy potřebná jednotlivá jména celé cesty k pinu a ne pouze jedno jméno přes tečkovou anotaci, tak jak se na proměnnou odkazuje např. v TIA Portalu.

Pro potřeby práce s pinem v rámci aplikace, např. vypsání jména proměnné na obrazovku, je přehlednější název s tečkovou anotací tak, aby programátor mohl vidět celou cestu k dané proměnné. Proto tato třída obsahuje metodu *GetDotAnnotationName()*, která vrací string s tečkovou anotací jména daného pinu.

Třída Tag

Tato třída reprezentuje PLC tag, jenž adresuje paměťové místo nebo skutečný fyzický výstup/vstup PLC. Oproti třídě *Pin* obsahuje atribut odkazující na adresu tagu a definuje konstruktor rozšířený právě u tuto položku.

Třída HMITag

Ač by se mohlo zdát, že tagy pro HMI budou stejné jako pro PLC, opak je pravdou. Bohužel HMI tagy nelze vytvářet stejně jako PLC tagy, viz kap. 2.2.4. Víceméně není jiný způsob, jak do TIA Portalu přidávat HMI tagy, než skrze importování. Tudíž jsou metody i atributy této třídy k tomuto uzpůsobeny. Oproti předkovi implementuje následující atributy:

- FileInfo : Entity – Vlastnost typu *Entity* (viz kap. 3.1.2) pro uchování zdrojového XML souboru s prázdným HMI tagem.
- HmiConnection : string – Obsahuje jméno HMI propojení s PLC.
- PlcTag : Pin – Vlastnost typu *Pin* jenž odkazuje na PLC proměnnou spjatou s tímto HMI tagem.

Dále *HMITag* obsahuje veřejnou metodu *Import*, která zprostředkovává jednak přepsání zdrojového XML souboru pro aktuálně navolený HMI tag, tak i jeho následné přidání do HMI tabulky, jenž je předána jako parametr této funkce. Za tímto účelem tato metoda používá další dvě privátní metody *OverrideXMLTag* a *Add2TagTable*.

Nejdříve se volá metoda *OverrideXMLTag*, která upraví zdrojový XML soubor. Úprava spočívá v přepsání následujících elementů:

- Name – Přepsání jména HMI tagu.
- Length – Určuje skutečnou datovou délku tagu.
- Coding – Kódování datového typu.

- Connection – Jméno HMI připojení.
- ControllerTag – PLC proměnná na níž odkazuje daný HMI tag.
- DataType – Datový typ proměnné.

Po přepsání zdrojového souboru se pomocí metody *Add2TagTable* se XML kód, reprezentující přepsaný HMI tag, vloží do XML souboru HMI tabulky, jenž obsahuje kolekci těchto tagů. Do TIA Portalu se tedy neimportuje přímo HMI tag, ale až naplněná HMI tabulka. To má velmi prostý důvod. Pokud by se importoval každý HMI tag individuálně do TIA Portalu, celé importování by trvalo velmi dlouho. Zatímco při importování pouze HMI tabulky se vždy provede jen jedno importování. To vede k velké časové úspoře při generování programu.

Třída ScreenEvent

Velmi jednoduchá třída pro události obrazovky. Obsahuje navíc jednu vlastnost, a sice jméno obrazovky, které událost náleží. Dále pak obsahuje privátní atribut rodiče, kde je uložena přímo obrazovka jakožto objekt *ScreenTia*, a metodu *GetParent()*, která vrací právě onoho rodiče.

Třída FunctionPin

Tato třída je přizpůsobena pro vstupy/výstupy funkčních bloků a funkcí. Přidává další tři atributy:

- BlockName : string – Jméno bloku, jenž je vlastníkem tohoto pinu.
- Retain : bool – Informace o tom, zda daný vstup/výstup je pro daný blok povinným.
- block : PlcBlockTia – Privátní atribut obsahující blok, jemuž pin patří.

Vlastnost *BlockName* se může jevit jako zbytečná, protože pin obsahuje i skutečně daný blok včetně jeho jména. K tomuto kroku bylo přistoupeno s ohledem na to, že při ukládání konfigurace (serializaci) vznikla chyba nekonečné smyčky. K chybě došlo, protože blok obsahoval piny reprezentující jeho vstupy/výstupy. Každý takový pin pak obsahoval blok, jenž ho vlastní a ten opět obsahoval náležité piny a tak stále dokola. Proto se při serializaci ukládá pouze jméno bloku. Při následném načítání konfigurace se dle tohoto jména vyhledá příslušný blok a opět se uloží k pinu.

Třída dále obsahuje dvě metody. Veřejnou metodu *GetOwnerBlock()*, která vrací obsažený blok. Druhá metoda je privátní se jménem *FindOwnerBlock()* a slouží k vyhledání rodičovského bloku dle jména.

Třída DataItem

Tato třída je vytvořena speciálně pro proměnné datových bloků. Jelikož datové bloky obsahují většinou složitější struktury, je tato třída navržena tak, aby je bylo možné uchovávat. Atributy této třídy jsou:

- BlockName : string – Jméno datového bloku, jenž je vlastníkem tohoto pinu.
- block : DB – Privátní atribut obsahující datový blok, jemuž pin patří.
- Members : List<DataItem> – Jestliže se jedná o strukturu, zde jsou uloženy potomci daného datového pinu.
- ParentName : string – Jméno rodiče, jemuž náleží daný pin. V případě tzv. kořenového pinu je nastaven na hodnotu null.
- Parent : DataItem – Privátní atribut obsahující pin, jehož potomkem je daný pin. V případě tzv. kořenového pinu je nastaven na hodnotu null.

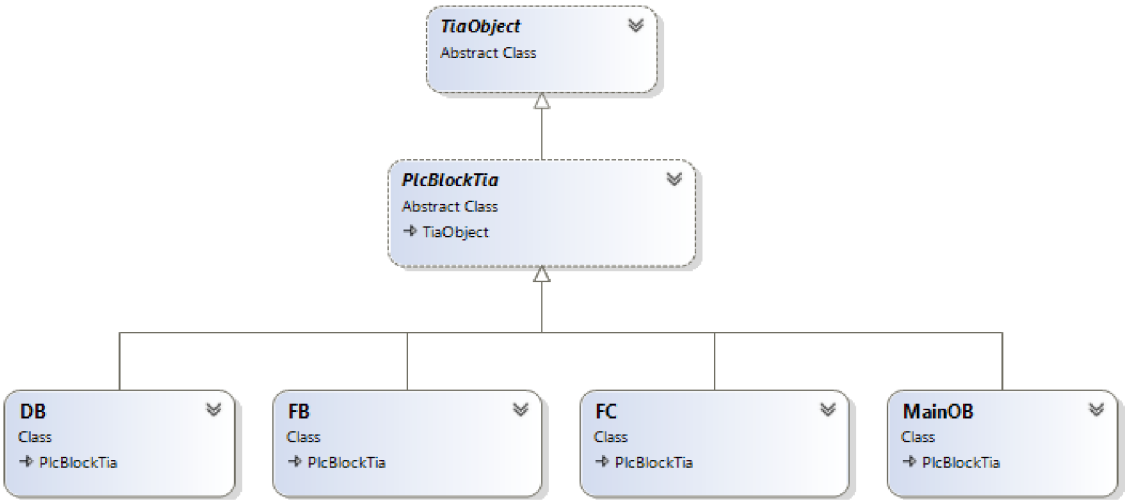
Stejně jako u třídy *FunctionPin*, byl i zde problém se serializací rodičovského bloku. Bohužel zde se problém přenesl i na rodičovský pin. Zatímco jméno bloku je v projektu unikátní a jde jej tedy v projektu snadno vyhledat, pro jméno pinu toto neplatí. Z tohoto důvodu se v této třídě přetížily operátory == a !=. Při přetížení se hledají piny daného bloku jenž mají stejné jméno, typ, předka, ale i stejné potomky. Obdobně jako u třídy *FunctionPin*, je zde metoda *GetOwnerBlock()* pro získání bloku. Přibyla další třída *GetParent()* pro získání rodičovského pinu. Poslední obsažená metoda je privátní *BuildName()*, jenž sestavuje jméno pinu.

5.1.4 Třídy pro PLC bloky

Při generování nových projektů je hlavním požadavkem dostat PLC bloky do nového projektu, jenž jsou stavebním kamenem každého PLC programu. Proto těmto objektům byla věnována největší pozornost. Stále se ladily a zdokonalovaly funkce pro jejich spravování a uchovávání, tak aby byly co nejuniverzálnější a nejpraktičtější. Opět bylo nezbytné, obdobně jako u pinů, aby byly bloky stejného typu a aby byly sjednoceny společné atributy a metody pro všechny bloky. Výsledkem je struktura bloků zobrazena na Obr. 5.3. Z obrázku je patrné, že všechny bloky dědí od abstraktní třídy *PlcBlockTia*. Zároveň si lze všimnout, že *PlcBlockTia* dědí ještě od třídy *TiaObject*, která je popsána výše v kap. 3.1.2.

Třída PlcBlockTia

Tato abstraktní třída sjednocuje vlastnosti a metody společné pro všechny bloky. Ve svých vlastnostech uchovává dvě kolekce, a sice:



Obr. 5.3: Struktura bloků

- MapList : List<Mapping> – Kolekce která uchovává informace o propojených vstupech/výstupech daného bloku.
- Pin : List<Pin> – Kolekce která obsahuje všechny vstupy/výstupy daného bloku včetně i jeho statických proměnných.

Z Obr. A.3 lze vidět přehled metod této třídy. Stěžejními metodami jsou *MapInput(Pin,Pin)*, *MapOutput(Pin,Pin)* a *SetConstant*, které umožňují přivedení požadované proměnné, respektive konstanty na jejich vstup/výstup. V těchto metodách je hned několik ošetření, zda je mapování v pořádku. V opačném případě se generuje výjimka, že nemohlo dojít k propojení se zprávou z jakého důvodu. Jedním takovým ošetřením je, že daný vstup je již propojen, jelikož aplikace v této verzi neumožňuje na vstup zapojit více proměnných. Dalšími důležitými kontrolami jsou např. zda se spojují stejné datové typy, nebo zda připojovaný vstup/výstup skutečně náleží danému bloku. Neméně důležitou metodou je *PasteConstants(PlcBlockTia)* , jenž převezme konstanty z pinů předaného bloku a přiřadí je pinům se stejným jménem v aktuálním bloku. Tato metoda je velmi užitečná např. při zadávání konstant pro blok převádějící analogovou veličinu.

Třída FB

Tato třída se stará o správu funkčních bloků. Nedefinuje funkční blok jako takový, ale pouze instanci funkčního bloku z vývojového prostředí TIA Portal, která je potřebná pro volání funkčních bloků. Z Obr. A.4 lze vidět, že obsahuje mnoho metod. Budou popsány jen ty, které jsou důležité pro vytváření bloků při konečném generování programu, nebo pokud není jejich význam patrný z názvu.

Atributy a vlastnosti:

- templateName : string – Jediný atribut třídy obsahující informaci o jméně zdrojového funkčního bloku, ke kterému se vztahuje tato instance.
- Instance : InstanceDB – Instanční datový blok.
- MappedPin : Dictionary<Pin, Boolean> MappedPin – „Slovník“ vstupů/výstupů sloužící jako klíč, obsahující hodnotu typu bool o tom, zda byly připojeny či nikoliv.

Metody:

- LoadFbIO() – Privátní metoda pro uložení vstupů/výstupů a statických proměnných do kolekce pinů.
- CheckRetainPins() – Metoda pro kontrolu, zda jsou připojeny všechny povinné vstupy/výstupy funkčního bloku. Pro kontrolu využívá vlastnosti *Retain* třídy *FunctionPin* a porovnává ji se „slovníkem“ zamapovaných pinů *MappedPin*. Jestliže metoda zjistí nesrovnalosti, pomocí výjimky vypíše seznam všech vstupů/výstupů, které musejí být připojeny pro správné generování výsledného programu.
- SetMapDictionary() – Tato metoda zajišťuje vyplnění „Slovníků“ *MappedPin* procházením kolekce *MapList* a určením, zda jsou piny skutečně zamapovány.

Třída FC

Tato třída je velice podobná třídě *FB*. Jedním z nepatrných rozdílů je, že postrádá vlastnost *InstanceDB* či některé z metod. Proto se popisu této třídy práce více věnovat nebude. Porovnání obsahu těchto dvou tříd je patrné z Obr. A.3 a Obr. A.4.

Třída InstanceDB

InstanceDB slouží pro uložení jména instančního datového bloku. Jeho hlavní funkcí je vytvoření instančního datového souboru skrze metodu *CreateInstance*. Soubor se vytvoří jako textový a následně je uložen s koncovkou „.db“, aby se mohl exportovat do TIA Portalu jako externí zdrojový soubor. Existuje ještě druhá varianta jak vytvořit instanční datový soubor, a to pomocí XML souboru. Tento způsob je však o poznání složitější. Proto byl v této práci zvolen prvně popsaný způsob. Vytvoření textového souboru ukazuje výpis 5.1, kde parametr *Name* udává jméno instančního datového bloku, *TemplateName* jméno funkčního bloku pro který se tvoří instance a *path* je cesta uložení souboru.

```
//Votvoření souboru
using (StreamWriter fs =
  new StreamWriter(path, true, Encoding.UTF8))
{
```

```
        string fristline = "DATA_BLOCK \"" + Name + "\"\n";
        string body = "{ S7_Optimized_Access := 'TRUE' }\n" +
            "VERSION: 0.1\n" +
            "NON_RETAIN\n";

        string FBname = "\"" + TemplateName + "\"";
        string endDB = "\n\nBEGIN\n\nEND_DATA_BLOCK\n";

        fs.Write(fristline + body + FBname + endDB);
    }
```

Výpis 5.1: Vytvoření instančního datového souboru

Třída DB

Tato třída je velmi jednoduchá, obsahuje přetíženou metodu *AddMembers(+1 overload)*, která se pouze stará o vytvoření kolekce pinů, tak aby se zachytila celá struktura proměnných a mohla se tak uložit do pinu typu *DataItem*. Druhou metodou této třídy je pak výpis všech obsažených pinů, včetně těch uvnitř struktur. Tato metoda se využívá při vytváření HMI tagů z datového bloku, kde se takto přidávají všechny proměnné v datovém bloku do HMI tagů.

Třída ProgramBlocks

Ve vývojovém prostředí TIA Portal jsou pod záložkou Program blocks obsaženy a spravovány veškeré bloky. Podobně i ve vytvářené aplikaci vznikla třída *ProgramBlocks*, jenž udržuje kolekci všech bloků a umožňuje jejich spravování, jako např. jejich přidávání či mazání. Tato třída má pouze dvě vlastnosti, přičemž první je kolekce všech bloků typu *List<PlcBlockTia>*. Druhá vlastnost je typu bool a obsahuje pouze informaci o tom, zda namapované bloky budou volány v jazyku SCL (hodnota true) nebo v LAD (hodnota false). Implicitně je nastaveno volání bloků prostřednictví jazyka SCL, neboť v této verzi programu nelze bloky v LAD volat z důvodů popsaných v kap. 5.1.5.

5.1.5 Volání bloků - Třída MainOB

Ačkoliv název může klamat, nejedná se o správu organizačního bloku OB1 - main v TIA Portalu. Jedná se o třídu reprezentující vlastní hlavní blok main pro generovaný projekt. V podstatě se jedná o funkční block s názvem *FB_Call's*, ve kterém dochází k volání všech namapovaných bloků. Zároveň se v této třídě vytváří v době generování XML soubor tak, aby byly volány všechny namapované bloky, včetně přiřazení odpovídajících proměnných na jejich vstupy/výstupy.

V principu se jedná o to, že v knihovně je připraven prázdný funkční blok s programovacím jazykem LAD. Následně se při generování projektu vytvoří network/y s voláním bloků, které se vkládají do připraveného prázdného bloku. Na začátku práce se stanovily dva možné způsoby, jak by se mohly jednotlivé bloky volat. Prvním z nich bylo pomocí jazyku LAD. Druhým způsobem bylo pomocí vložení SCL networku a bloky volat pomocí jazyku SCL. Byly vytvořeny a odzkoušeny oba zmíněné způsoby. V práci se nakonec použil pouze druhý zmíněný způsob, a to především ze tří důvodů. Prvním a hlavním důvodem bylo, že úprava networku s jazykem LAD je o poznání složitější a nepřehledná. Jestliže se v rámci konfigurace připojí na vstup/výstup proměnná, která není definována, generování končí chybou a přerušením importu, což se při volání pomocí jazyku SCL nestane. Druhým důvodem bylo, že SCL network lze vložit na libovolné místo programu i v případě, že je celý program psán např. v jazyku LAD, což se opačně říci nedá. Třetím a posledním důvodem je, že v jazyku SCL se daleko snadněji přiřazuje proměnná např. do datového bloku, zatímco u jazyku LAD je nutno použít speciální blok *Move*.

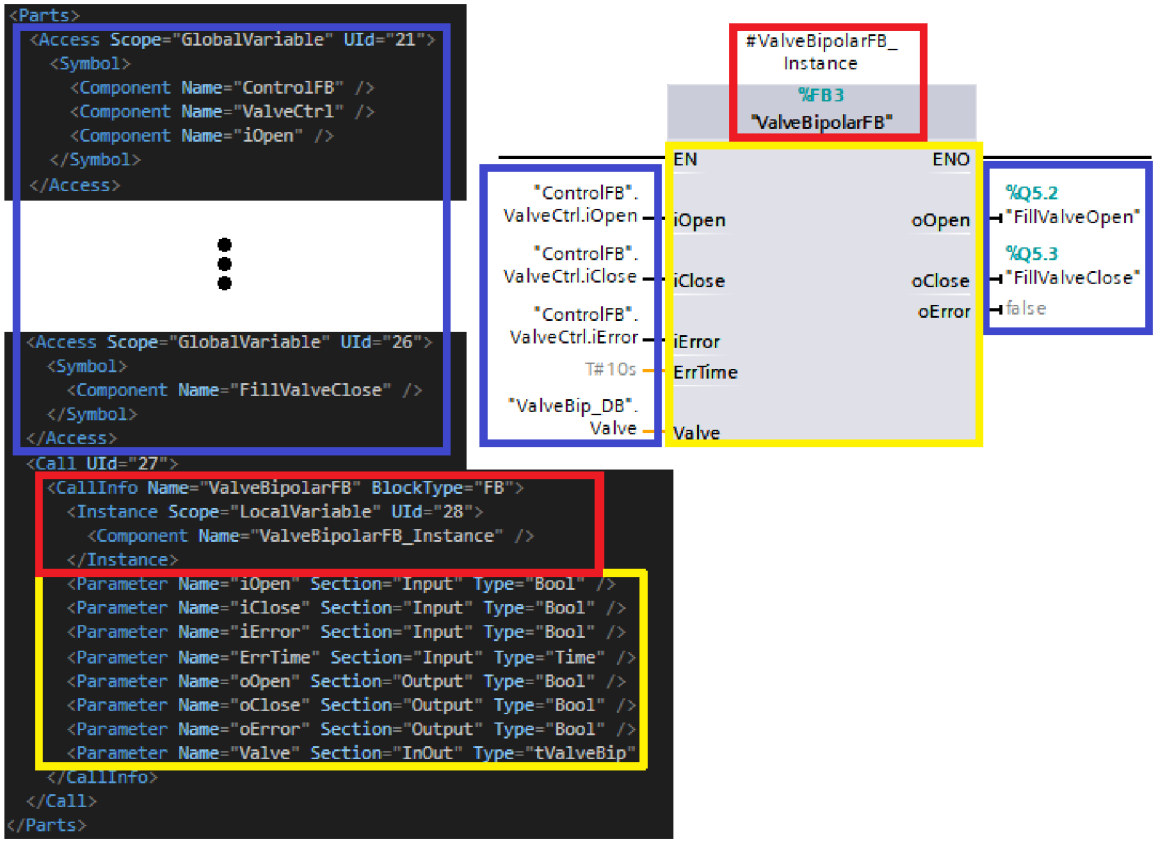
V následujícím popisu bude vysvětleno, jakým způsobem lze dosáhnout vytvoření networku s voláním bloků. Network je v XML souboru představován elementem *<SW.Block.CompileUnit>* s jeho potomky. Tento element se nebude vytvářet od začátku, ale použije se předem připravený template pro prázdný network, který je zobrazen v příloze ve výpisu B.1, pro programový jazyk LAD a ve výpisu B.2 pro jazyk SCL.

Volání bloků pomocí jazyku LAD

Připravený template pro tento jazyk se bude dopisovat o vstupy a výstupy, které obsahuje daný blok. Doplní se jméno bloku a vytvoří se namapování proměnných vstupujících a vystupujících do/z bloku. Tento proces se může rozdělit do dvou kategorií, a to úprava elementu *<Parts>* a elementu *<Wires>* s jejich potomky.

Prvním krokem je vytvoření elementu *<Parts>*. Jak lze vidět z Obr. 5.4, nejdříve jsou vyjmenovány proměnné vstupující do/z bloku - modré obdélníky. Každé takové proměnné je přiřazeno v rámci networku unikátní identifikační číslo *UId*, které se využije při mapování vstupů či výstupů daného bloku. Následuje samotné volání bloku, které je červeně ohraničené a spadá pod element *Call*, kde jestliže se jedná o funkční blok, je nejdříve uvedena instance bloku a následně jméno bloku. Nakonec jsou vyjmenovány všechny vstupy/výstupy bloku - žluté obdélníky. Popsanou úpravu zprostředkovává privátní metoda *WriteParts()*.

V druhé části se musí vytvořit spojení mezi proměnnými, které vstupují do bloku se samotnými vstupy bloku, na které jsou přivedeny. Tyto informace jsou uloženy v elementu *<Wires>*, kde pod jednotlivými potomky *<Wire>* jsou uložena



Obr. 5.4: Network s voláním bloku v jazyku LAD - element Parts

všechna spojení. V každém tomto elementu je potom *<NameCon>*, který značí jméno vstupu či výstupu bloku. Pokud je na vstup přivedena proměnná, je dalším potomkem *<IdentCon>*, kde je již pouze atribut *UId* vstupu z elementu *<Parts>*. Jestliže na vstup není přivedena žádná proměnná, potom je nahrazeno *<IdentCon>* za *<OpenCon>*, což přivádí předdefinovanou hodnotu na daný vstup. Rozeznávání, která proměnná patří ke kterému vstupu či výstupu, zajišťuje atribut *UId*. Tyto modifikace zprostředkovávají metody *WireInputWires()* a *WireOutputWires()*. Např. ve výpisu 5.2 lze vidět, jakým způsobem je namapován vstup *iOpen* bloku *ValveBipolarFB* z Obr. 5.4.

```
<Wires>
  <Wire UId="32">
    <IdentCon UId="21" />
    <NameCon UId="27" Name="iOpen" />
  </Wire>
</Wires>
```

Výpis 5.2: Mapování vstupů pomocí LAD - element Wires

V aktuální verzi programu jsou metody pro vytvoření volání bloků v jazyku LAD opatřeny komentářem a nejsou s touto verzí kompatibilní, jelikož v průběhu práce bylo od tohoto řešení ustoupeno. Nicméně je ověřeno, že takto vytvořené volání bloků je funkční a lze jej použít.

Volání bloků pomocí jazyku SCL

Jak bylo řečeno výše, v práci je použit tento způsob. Postup jakým se vytváří XML soubor pro volání bloků jazykem SCL bude vysvětlen na příkladu. Ve výpisu 5.3 je ukázka kódu, kde je volán inkrementální enkodér.

```
"Pump_IEncoderFB_DB"(Run =>"Pump_MotorFB_DB".iRun,
                      A :="i_PumpEncoder_A",
                      B :="i_PumpEncoder_B",
                      "Resolution [m/s]" :=0.0225);
```

Výpis 5.3: Volání bloku v jazyku SCL

Toto volání se zprostředkuje pomocí XML kódu tak, jak je zobrazeno ve výpisu 5.4. Níže bude vysvětleno jakým způsobem se při generování skládá tento XML kód.

Volání bloku či přiřazení proměnné vždy začíná elementem *Access*. Jediné co se liší, je hodnota atributu *Scope*. Jestliže jde o volání bloků, je tato hodnota *Call*. Pokud se jedná o přiřazení, definuje se jako *GlobalVariable*. Následují informace o volaném bloku - jeho jméno a jestliže se jedná o funkční blok tak i jeho instance. Tímto se vypíše jméno bloku, které je voláno. Následuje znak závorky. Znaky se vkládají do elementu *Token*. Dále již následují elementy *Parameter*, ve kterých dochází k přiřazení namapovaných proměnných na příslušné vstupy/výstupy. Vstup bloku je vždy uveden v atributu elementu *Parameter*, za nímž následuje mezera. Znak mezery představuje atribut *Blank*, v jehož atributu je počet vynechaných míst. Pak již následuje znak pro přiřazení. Jestliže se jedná o výstup, je znakem => (XML značka pro > je gt), zatímco pro vstup je znakem :=. Poté již následuje přiřazení proměnné přes elementy *Access, Symbol, Component*. Lze si všimnout, že pokud se jedná o strukturu, je vždy mezi komponenty vložen znak tečky. Po vytvoření a ukončení elementu *Parameter* následuje příprava dalšího řádku, kde je nejdříve vložen symbol čárky, skok na novou řádku (*NewLine*) a následně vynecháno tolik míst dle délky jména volaného bloku.

```
<Access Scope="Call" UId="572">
  <CallInfo UId="573" BlockType="FB" Name="Pump_IEncoderFB">
    <Instance Scope="GlobalVariable" UId="574">
      <Component Name="Pump_IEncoderFB_DB" UId="575" />
    </Instance>
    <Token Text="(" UId="576" />
```

```
    <Parameter Name="Run" UId="577">
      <Blank Num="1" UId="578" />
      <Token Text="=&gt;" UId="579" />
      <Access Scope="GlobalVariable" UId="580">
        <Symbol UId="581">
          <Component Name="Pump_MotorFB_DB" UId="582" />
          <Token Text="." UId="583" />
          <Component Name="iRun" UId="584" />
        </Symbol>
      </Access>
    </Parameter>
    .
    .
    <Token Text="," UId="603" />
    <NewLine Num="1" UId="604" />
    <Blank Num="21" UId="605" />
    <Parameter Name="Resolution [m/s]" UId="606">
      <Blank Num="1" UId="607" />
      <Token Text=":=" UId="608" />
      <Access Scope="LiteralConstant" UId="609">
        <Constant UId="610">
          <ConstantValue UId="611">0.0225</ConstantValue>
        </Constant>
      </Access>
    </Parameter>
    <Token Text=")" UId="612" />
  </CallInfo>
</Access>
<Token Text=";" UId="613" />
<NewLine Num="2" UId="614" />
```

Výpis 5.4: Úryvek XML kódu pro volání bloku pomocí jazyku SCL

Nepatrně se liší vytváření XML kódu, pokud se přiřazuje konstanta. To lze vidět v druhé části výpisu 5.4. Rozdíl je v tom, že je zde nahrazen element *Symbol* a *Component* za *Constant* a *ConstantValue*, přičemž hodnota konstanty je hodnotou elementu *ConstantValue*. Po posledním vstupu/výstupu bloku je vložen symbol pro závorku, středník a vynechány dva řádky, a poté může následovat volání dalšího bloku. Vytváření jednotlivých elementů je zprostředkováno metodami statické třídy *SCLParse*.

V rámci generování se vytvoří celé volání tak, jak je zobrazeno ve výpisu 5.4, které je následně vloženo do elementu *StructuredText* z výpisu B.2. Toto se opakuje pro všechny bloky obsažené v konfiguraci.

Dokončení volání bloků

Takto doplněný element *<SW.Block.CompileUnit>* se následně importuje do XML souboru, funkčního bloku *FB_Call's*.

Důležité je na závěr přepsat všechny hodnoty atributů ID v upraveném XML souboru, jehož hodnotou je hexadecimální číslo tak, aby nebyly duplicitní. O to se stará rekurzivní metoda *reNumberID*.

5.1.6 Uživatelsky definované typy a tabulky tagů

Třída UserDataType

Tato třída je velmi jednoduchá. Uchovává uživatelsky definované typy, které již programátor kdysi vytvořil a následně jej přidal do projektu z knihovny. Po přidání do projektu se v instanci této třídy vyčtou proměnné ze zdrojového XML souboru a uloží do kolekce, jenž je jedinou vlastností této třídy. *UserDataType* opět pracuje s XML souborem, tudíž dědí od třídy *TiaObject*.

Třída TagTable

Tato třída reprezentuje tabulky tagů stejně tak, jako jsou ve vývojovém prostředí TIA Portal. Z Obr. A.2 vpravo nahoře je patrné, že obsahuje pouze dvě vlastnosti, a sice *Name* - jméno tabulky a *Tags* - seznam tagů obsažených v tabulce, typu *List<Pin>*. Tato třída disponuje převážně metodami pro správu obsažených tagů. Za zmínění stojí především metoda *LoadTagsFromCSVFile(string)*, která načte tagy z CSV souboru do tabulky. Kód pro načtení tagů a příklad formátu CSV souboru je zobrazen ve výpisu 5.5. Po zavolání této metody se odstraní všechny dosavadní tagy v tabulce a následně se přidávají tagy definované v CSV souboru.

```
bool first = true;
using (StreamReader sr = new StreamReader(filePath))
{
    string s;
    while ((s = sr.ReadLine()) != null) // čte řádek po řádku
    {
        // rozdělí string řádku podle středníků
        string[] parameters = s.Split(';');
        string name = parameters[0];
        string type = parameters[1];
        string adress = parameters[2];

        //Přidání tagu
        if (first == false)
            CreateTag(name, type, adress);
```

```
            else
                first = false;
        }
}
//Formát CSV souboru
Name;Type;Adress   //Hlavička
Tag1;Bool;I4.0     //Přidávaný tag
```

Výpis 5.5: Nahrání tagů do tabulky tagů z CSV souboru

Třída HMITagTable

Jelikož pomocí knihoven TIA Openness nelze vytvořit HMI tabulku stejně jako u PLC, je zde využito jiného řešení. Základ HMI tabulky bude stejný jako u PLC tabulky, proto bude od této třídy zděděná, ostatně jak lze vidět i na Obr. A.2. Tím tato třída získá základní metody pro správu tagů a nemusí se definovat znovu. V *HMITagTable* se pouze přepisuje metoda *AddTag(Pin)*, tak aby si třída ohlídala, že se do ní přidávají pouze HMI tagy, tedy aby byly typu *HMITag*. Po této kontrole využívá zděděnou metodu.

Mnohem zajímavější jsou však metody, které se zde musely vytvořit. Jak bylo řečeno, HMI tabulku nelze snadno vytvořit, proto se do knihovny aplikace přidal zdrojový XML soubor s prázdnou HMI tabulkou, který tato třída přepisuje a upravuje. Pro generování tabulky do TIA Portalu se volá metoda *Import()*, která postupně používá privátní metody *OverrideXMLTagTabel()*, *AddTags2XMLTagTable* a *reNumberID(XmlDocument)*.

Nejdříve metoda *OverrideXMLTagTabel()* změní v XML souboru jméno tabulky, aby bylo možné následně pomocí metody *AddTags2XMLTagTable* přidat veškeré HMI tagy udržované v kolekci do této tabulky. V této metodě se proiterují veškeré tagy, přičemž se volá jejich metoda *Import(HMITagTable)*, čímž se tagy „samy přidají“ do tabulky, jak již bylo popsáno v kap. 5.1.3. Na závěr se volá metoda *reNumberID(XmlDocument)*, která prochází celý XML dokument a postupně přečíslovává všechny atributy *ID* tak, aby byly unikátní v rámci celého dokumentu. Takto upravený soubor je připraven pro import do vývojového prostředí TIA Portal.

5.1.7 Technologické objekty

Bylo uvažováno, že by aplikace mohla umět přidávat různé technologické objekty. Ukázalo se však, že většina technologických objektů se pro různé verze PLC zařízení liší. Je zde tolik parametrů k nastavování, že pro programátora je přehlednější si technologický objekt nastavit na míru přímo ve vývojovém prostředí TIA Portal.

Jedinou výjimku tvoří PID regulátory. Ty jsou pro PLC typu S7-1200 a S7-1500 totožné. Aplikace umožňuje vytvoření dvou regulátorů, a sice obecného regulátoru označovaného v TIA Portalu jako PID_Compact a regulátoru zaměřeného pro regulování teploty PID_Temp.

Seznam regulátorů a jejich správu udržuje třída *TechnologyObjects*, jejíž jedinou vlastnosti je kolekce PID regulátoru typu *List<TO_PID>*.

Samotný regulátor je představován třídou *TO_PID*, která je na Obr. A.4 vpravo dole. Lze se všimnout, že disponuje velkým počtem konstruktorů, a to z důvodu, že je možné vytvořit regulátor s různými parametry a ostatní nechat na přednastavených hodnotách. Přehled všech parametrů, které lze nastavovat nebo číst je uveden v literatuře [22]. Význam vlastností je uveden v následujícím seznamu.

- Name : string – Jméno regulátoru.
- Type : PID_Type – Typ regulátoru, zda-li se jedná o obecný (Compact) nebo pro teplotu (Temp). Defaultním typem je obecný regulátor.
- DerivativeTerm : double – Derivační složka regulátoru.
- IntegralTerm : double – Integrační složka regulátoru.
- ProportionalGain : double – Proporcionální zesílení regulátoru.
- HighLimit : double – Horní limit regulované hodnoty.
- LowLimit : double – Dolní limit regulované hodnot.
- Sampling : double – Vzorkování.
- Version : double – Verze regulátoru, doplní se automaticky podle jeho typu.

5.1.8 HMI obrazovky

Pro spravování HMI obrazovek vznikla třída *ScreenTia*, která je opět zděděná od třídy *TiaObject*, neboť pracuje se zdrojovým XML souborem. *ScreenTia* obsahuje metody především pro úpravu velikosti a pozice objektů na obrazovce a pro přiřazení PLC proměnných jednotlivým událostem obrazovky.

Změna velikosti a pozice objektů

V situaci, kdy by programátor uložil obrazovky ze starého projektu, kde bylo použito HMI zařízení s velikostí dvaadvacet palců, do aplikační knihovny a pak tyto obrazovky použil v novém projektu, kde je k dispozici zařízení o velikosti pouze čtrnácti palců, došlo by ke dvěma problémům. První problém by byl ten, že by importování do vývojového prostředí selhalo s vygenerováním výjimky, že rozměry obrazovky neodpovídají rozměrům HMI zařízení, což je ostatně popsáno i v manuálu k TIA Openness, viz [10] str. 457. Tyto rozměry obrazovky se dají však v XML souboru jednoduše změnit tak, aby se shodovaly s rozměry HMI zařízení. Zde však nastává

druhý, složitější problém. Import do TIA Portalu se provede. Jelikož je obrazovka navržena pro větší displej, některé komponenty budou příliš velké, ba dokonce úplně mimo obrazovku. Proto je nezbytné změnit nejen velikost všech komponent, ale i jejich umístění a velikost použitého písma. Třída obsahuje celkem tři metody, které řeší tyto problémy.

Pouze jediná metoda *ChangeComponentSize(int,int,int)* je veřejná, z které se volají dvě privátní metody. V parametrech této metody jsou předány výška a šířka HMI zařízení, do kterého mají být obrazovky importovány. Třetím parametrem je pořadové číslo obrazovky, které musí mít každá obrazovka v zařízení unikátní.

První z privátních metod se nazývá *LoadScreenRatio*, která nejdříve vyčte ze zdrojového XML souboru pro danou obrazovku její rozměry, aby následně mohla určit poměr mezi rozměry obrazovky a HMI zařízení. Po uložení těchto poměrů se změní velikost aktuální obrazovky a její pořadové číslo, aby se mohla zavolat druhá privátní metoda *ChangeComponentSizeInScreen*. Tato metoda postupně prochází veškeré vrstvy (Layer) dané obrazovky a v těchto vrstvách pak všechny komponenty. Postupně se mění elementy *Height, Width, Top a Left*. Některé další komponenty např. kruhy nebo elipsy obsahují ještě element *Radius*, jenž představuje velikost zaoblení komponenty. Jestliže komponenta tento element obsahuje, nejdříve se porovná jestli se shoduje jeho výška a šířka. Pakliže se shodují, jedná se o kruh a rádius bude polovina těchto rozměrů. Jestliže se nerovnají, vypočte se radius dle vzorce (5.1). Příklad výseku z XML souboru s obrazovkou a vyznačenými elementy, které se musejí měnit jsou obsaženy v příloze C ve výpisu C.1.

$$h = R - \sqrt{R^2 - L^2/4} \qquad [21] \tag{5.1}$$

kde: h - Výška úseče
R - Poloměr kružnice
L - Tětiva

Přiřazení PLC proměnných událostem

Každá obrazovka obsahuje události, nazývané také eventy, pomocí kterých lze zadávat příkazy do PLC či zobrazovat stavy z PLC. Proto je při generování nového projektu nezbytné tyto události přiřadit příslušným PLC proměnným, jejichž stav bude zobrazovat/měnit.

Jednotlivé události jsou v XML dokumentu obsaženy pod elementy *<Tag>* (většinou se jedná o různé animace) a *<Value>* - (nejčastěji ovládání tlačítek). Obsahem těchto elementů je název HMI tagu. Při vytváření obrazovky se postupně

procházejí tyto elementy a ukládají se jednotlivé události do třídy *ScreenEvent*, jejíž jménem je název HMI tagu. Algoritmus je naprogramován tak, že pokud se některý HMI tag vyskytuje ve více událostech, uloží se jen jeden. Po přiřazení PLC proměnné se však následně přepíší všechny události se stejným původním HMI tagem.

Pro přiřazení PLC proměnných k událostem disponuje tato třída metodou *MappEvent*. V této metodě se mimo jiné kontroluje zda mapovaná proměnná není uživatelsky definovaným typem. Jestliže není, přiřadí se proměnná k události. Pokud však je UDT, postupně se přiřazují shodná jména jednotlivých proměnných k příslušným událostem do vlastnosti *MapList* typu *Dictionary*, jejíž klíčem je mapovaný event. Toto velmi urychlí mapování událostí a nemusí se přiřazovat jedna po druhé. I z tohoto důvodu byla referenční sada modulů tvořena pomocí UDT.

5.2 Servisní třídy vykonávající komunikaci pro TIA Portal

Tyto třídy vznikly z důvodu oddělení logiky pro komunikaci s TIA Portalem od konfigurování projektu. Tudíž pouze v těchto třídách se používají třídy z knihoven *Siemens.Engineering* a *Siemens.Engineering.Hmi* od firmy Siemens. Pomocí těchto knihoven je umožněno API aplikaci komunikovat s vývojovým prostředím TIA Portal a ovládat jej. Tyto třídy vznikly jako statické z toho důvodu, že není nutné, aby uchovávaly jakékoli informace, ale pouze poskytovaly metody pro obsluhu TIA Portalu. Vznikly celkem tři obslužné třídy, které jsou popsány v této kapitole.

5.2.1 TIAService

Tato servisní třída disponuje obecnými funkcemi pro komunikaci s vývojovým prostředím TIA Portal a spravování projektu. Proto obsahuje metody pro spuštění TIA Portalu a pro správu projektu, jako je např. otevření projektu, uložení, zavření atd. Obsahuje však i metody pro spravování zařízení v projektu. Jedná se např. o přidání nového zařízení, vyhledávání zařízení v projektu či spojení vybraných zařízení do společné subsítě. Většina metod je volána ze třídy *ProjectTIA* a byly již popsány v rámci této třídy v kap. 5.1.2.

Za zmínku stojí především metody *Connect2Subnet a CreateDevice*, které budou podrobněji vysvětleny níže.

Připojení zařízení do subnet

Nejdříve bude popsána metoda *Connect2Subnet*, která připojí zařízení předané v parametru této metody do subsítě, jejíž jméno je rovněž předáno jako parametr.

V prvním kroku se prohledají subsítě, které jsou již součástí TIA Portalovského projektu. Jestli daná síť již existuje, převezme se její instance, do které se následně zařízení přidá. Pakliže síť neexistuje, vytvoří se nová pomocí metody *Create*. Parametry této metody jsou typ sítě a její jméno. Tato metoda je obsažena ve třídě *SubnetComposition*, která udržuje kolekci sítí v projektu. Popsaný postup je zobrazen ve výpisu 5.6, kde se na konci opět volá metoda *Connect2Subnet*. Jelikož je tato metoda přetížená, tak se tentokrát odkazuje na privátní verzi s parametry zařízení a vytvořené nebo převzaté sítě.

```
public static void Connect2Subnet(DeviceTia device, string
    nameSubnet)
{
//Získani kolekce síti pomocí vlastnosti Subnets třídy Siemens.
    Project
     SubnetComposition subnets = projectTia.GetProject().Subnets;
     Subnet subnet = null;

//Jestliže síť se jménem v par. nameSubnet neexistuje, vytvoř jej
     if (subnets.Find(nameSubnet) == null)
         subnet = subnets.Create("System:Subnet.Ethernet",
            nameSubnet);
//Síť s předaným jménem existuje
     else
         subnet = subnets.Find(nameSubnet);

     Connect2Subnet(device, subnet);
}
```

Výpis 5.6: Vytvoření subsítě v projektu

V privátní verzi metody dochází již k samotnému přidání zařízení do subsítě. Jak lze vidět z dalšího výpisu 5.7, musejí se použít dva cykly *foreach*, které postupně procházejí všechny položky zařízení *DeviceItem* uvnitř předcházejících *DeviceItem*, obsažených v samotné třídě *Siemens.Device*. Proč tomu tak je vystihuje Obr. 2.3 ze kterého je patrné, že první úroveň *DeviceItem* představuje pouze strukturu zařízení např. tzv. Rack. Až pod ním se nachází položky jako PLC nebo v tomto případě hledané síťové rozhraní pro připojení do subsítě. Ta se nalezne pomocí metody *GetService<NetworkInterface>()* a následném procházení položek *Node* tohoto rozhraní. Připojeni se provede metodou *ConnectToSubnet*. Následně se může ještě nastavit IP adresa zařízení pomocí metody *SetAttribute*.

```
foreach (DeviceItem devitem in device.DeviceItems)
{
  Device siemensDevice = device.GetDeviceHW();
    foreach (DeviceItem item in siemensDevice.DeviceItems)
    {
```

```
            NetworkInterface itf = ((IEngineeringServiceProvider)item).
                GetService<NetworkInterface>();
            if (itf != null)
            {
                foreach (Node node in itf.Nodes)
                {
                    node.DisconnectFromSubnet();
                    node.ConnectToSubnet(subnet);
                    node.SetAttribute("Address", device.IP);

                    return;
                }
            }
        }
    }
```

Výpis 5.7: Přidání zařízení do subsítě

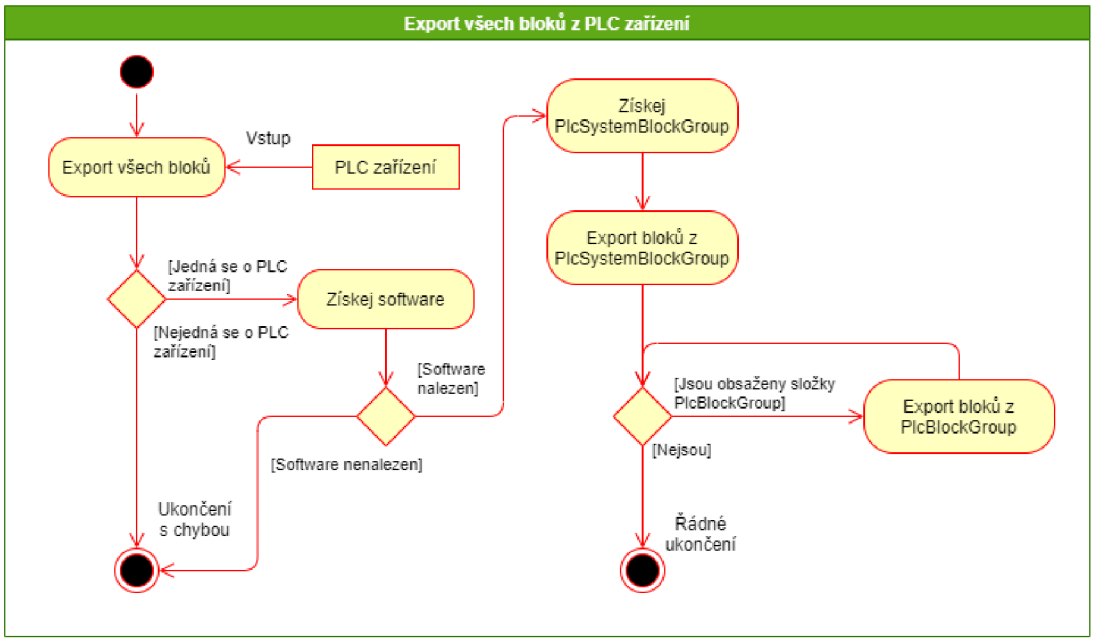
Vytvoření zařízení

Jak vytvořit zařízení bylo již popsáno v kap. 2.2.3. Zde však bude rozebrán problém při tvorbě HMI zařízení. Dle manuálu k TIA Openness by se dalo předpokládat, že HMI zařízení se bude vytvářet úplně stejně jako PLC. Realita je však taková, že rozhraní TIA Openness nepodporuje vytváření HMI. Při pokusu jej vytvořit dojde k vygenerování výjimky, že se jedná o neznáme zařízení a otevřené vývojové prostředí TIA Portal se neočekávaně bez uložení ukončí. Jednou z variant jak by se toto dalo obejít, je vytvořit HMI zařízení pomocí master kopie z jiného projektu. To by však znamenalo otevírat další projekt a z tohoto projektu zařízení převzít. Tento způsob je však pro generování velice časově náročný. Proto je v této práci použita druhá manuální možnost, avšak mnohem rychlejší. Při potřebě vytvořit HMI zařízení se pomocí metody *ShowHwEditor(Siemens.Engineering.HW.View.Device)*, obsažené ve třídě *Project* , otevře v TIA Portalu editor pro zařízení a aplikace vyzve programátora, aby požadované zařízení přidal manuálně.

5.2.2 PLCService

Druhou servisní třídou je *PLCService*, která spravuje PLC hardware. Převážně je používána třídou *PLCTia* v metodě *Import* pro vytvoření jednotlivých komponent v PLC zařízení. Jednotlivým metodám je vždy předávána parametrem třída *PLCTia*, z níž jsou brány vlastnosti (např. tabulky s tagy, bloky) pomocí kterých se tvoří objekty ve vývojovém prostředí TIA Portal.

Zajímavými metodami jsou však naopak metody pro získávání objektů z projektu, neboli jejich exportování. Exportování všech objektů je vždy na stejném principu, který je zachycen v diagramu pro exportování všech bloků na Obr. 5.5.



Obr. 5.5: Aktivity diagram - Exportování všech bloků

Z diagramu je patrné, že jediným vstupem pro metody zajišťující exportování bloků, je PLC zařízení, z nějž je nutné nejdříve získat instanci třídy *PlcSoftware*, což je více rozebráno v kap. 2.2.3. Následně se z PLC softwaru získá systémová skupina pro bloky *PlcSystemBlockGroup* představující složku *Program blocks* v TIA Portalu. Z této skupiny se exportují všechny bloky. Posléze se aplikace dotazuje, zda se nacházejí v aktuální skupině další složky s bloky, z nichž je možné bloky exportovat. Toto probíhá rekurzivně, dokud nejsou prošlé všechny složky s bloky.

Tento rekurzivní způsob exportu je v rámci této práce stejný pro všechny exportované objekty. Vždy se jen liší názvy tříd systémových skupin a jejich dílčí skupiny.

5.2.3 HMIService

Tato třída spravuje HMI zařízení přičmž většina metod slouží pro export jednotlivých objektů z TIA Portalu, fungující na stejném principu, který byl již popsán u třídy *PLCService*. Obsahuje dvě metody pro import HMI tabulek tagů a obrazovek. Práce se bude věnovat pouze popisu jediné metody, a sice *FindConnection*. Teoreticky vzato, by tato metoda měla prohledat HMI software, tedy třídu *HmiTarget* z knihovny TIA Openness. Jednou z vlastností *HmiTarget* je *Connections*, jenž reprezentuje kolekci všech připojení HMI zařízení. Zde nastává

problém. V kolekci se nenachází žádné připojení, ačkoliv v TIA Portalu toto připojení s navázaným PLC zařízením existuje. Takové připojení se označuje jako integrované. Pomocí knihoven z TIA Openness se však nedá jakkoli dotazovat ani jej vytvářet či importovat. Připojení lze vyhledat nebo importovat jen v případě, že je tzv. non-integrated, tedy není připojeno k PLC zařízení. Jenže takové připojení poté nelze připojit k PLC.

Nemožnost vytvoření HMI propojení představuje problém pro importování HMI tagů, jelikož jméno připojení je jedním z nutných atributů při importu. Pokud jméno HMI připojení uvedené u HMI tagu v daném HMI zařízení neexistuje, importovaní selže a vývojové prostředí TIA Portal se nečekaně bez uložení předchozí nahrané konfigurace ukončí, tzv. spadne.

5.3 Generování projektu

Jakmile je konfigurace hotová, je možno přistoupit ke generování projektu. Konfiguraci je možné vytvořit v rámci průvodce aplikace. Druhou možností je nahrát konfiguraci z XML předpisu, kterou je možné ihned generovat nebo ji upravit či doplnit v průvodci pro aktuální projekt.

Proces generování nejlépe vystihuje tzv. activity diagram zobrazen v příloze D. Je zde vidět, jak datový tok programu postupuje při generování projektu. Diagram je rozdělen do tří sektorů, a sice projekt, PLC a HMI zařízení. V projektové části se voláním metody *ProjectTia.Generate()* odstartuje akce generování. Nejdříve je provedena kontrola, jestli je aplikace připojena k TIA Portalu s otevřeným projektem, do kterého se má konfigurace provézt. Následuje podmínka tázající se, zda je počet nakonfigurovaných zařízení v projektu roven alespoň jedné. Jestliže nejsou splněny výše uvedené podmínky, generování se ukončí výjimkou s popsaným problémem. V opačném případě se začnou importovat nejdříve PLC zařízení, následována HMI. Generování jednotlivých druhů zařízení obstarává metoda *DeviceTia.Import()*, což je popsáno v následujících dvou podkapitolách.

Po importu všech nakonfigurovaných zařízení se vytvoří v TIA Portalu vytvořené sítě. Tento krok bohužel postrádá smysl a mohl by se z procesu vyškrtnout. Důvodem je, že vytvořením sítě a připojením zařízení do této sítě nevznikne *HMI connection* propojující softwarově HMI s PLC. Toto propojení nelze v současné verzi TIA Openness nijak vytvořit. Proto se vytváří *HMI connection* manuálně, viz kap. 5.3.2-Generování HMI. Při vytvoření *HMI connection* se zároveň vytvoří i síť mezi oběma zařízeními, tudíž již není potřeba ji vytvářet v rámci konfigurace.

Na závěr se importuje projektová grafika. Jedná se zejména o použité grafické prvky ve vizualizaci. Pro připravené vizualizace jsou tyto grafické prvky součástí knihovny. Při vytváření vlastních vizualizací si aplikace sama stáhne použitou grafiku k obrazovkám.

5.3.1 Generování PLC

Generování PLC zařízení je zobrazeno v pravé části Obr. D.1. Nejdříve je provedena kontrola zda je atribut *myDevice* prázdný. V podstatě se zde dotazuje, zda je v projektu v TIA Portalu již toto zařízení obsaženo, např. manuálně vytvořeno uživatelem. Jestliže není, zařízení se vytvoří metodou *TIAService.CreateDevice(DeviceTia)*.

Následně je kontrolováno jestli jsou známy všechny datové typy. Kontroluje se zda je součástí nakonfigurovaného projektu UDT, jenž jsou použity u některých pinů bloků. Pokud je využíváno u funkčního bloku multi-instancí, tak se kontroluje zda je znám blok, ke kterému se multi-instance vztahuje. Pokud se naleznou neznámé datové typy, je vygenerována výjimka s obsahem neznámých typů a uživatel je musí v rámci konfigurace doplnit. V případě, že je vše v pořádku, tak se setřídí uživatelsky definované typy. Jestliže některé UDT ve své struktuře obsahuje jiné UDT, je zařazeno na konec kolekce tak, aby bylo importováno jako poslední. Proč tomu tak je, je popsáno v kap. 5.1.2 - popis metod třídy *PLCTia*.

Dalším krokem je pomocí metody *PLCService.CreateTagTables(PLCTia)* vytvořit tabulky tagů a PLC tagy. Zde se v případě shody názvů tabulek aplikace dotazuje uživatele, zda ji má přepsat či nikoliv. Dle diagramu následuje kontrola, zda jsou při volání bloků připojeny všechny povinné vstupy/výstupy. Opět při nesplnění této podmínky je vygenerována výjimka s názvem bloku a jeho vstupů/výstupů, které nejsou namapovány. Toto byla poslední kontrola před generováním.

Před samotným importováním objektů je nutné provést jiné důležité náležitosti. První z nich je vytvoření XML souboru pro funkční blok *FB_Call's*, kde budou volány všechny namapované bloky. Poté dochází u všech bloků k přečíslování jejich čísla. Třetím a posledním krokem před importem je úprava XML souborů všech bloků, pokud se jedná o zařízení typu S7-1200.

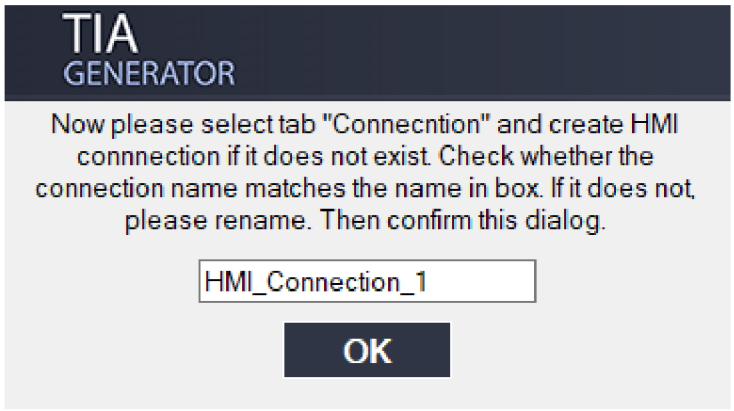
Nakonec se postupně importují UDT (*PLCService.ImportUDT(PLCTia)*, bloky do vytvořených složek podle typu bloku metodou *PLCService.ImportBlocks(PLCTia)* a vytvoření PID regulátorů - *PLCService.CreateTechnologicalObject(PLCTia)*.

5.3.2 Generování HMI

Jak lze vidět z diagramu na Obr. D.2, začátek pro generování HMI je obdobný jako pro PLC. Rozdíl je však v tom, že nefunguje vytvoření zařízení stejně jak pro PLC. Z tohoto důvodu se otevře v TIA Portalu editor pro správu zařízení a následně je uživatel vyzván k přidání HMI zařízení manuálně. Poté se otevře dialogové okno z Obr. 5.6 s pokyny pro uživatele, aby zkontroloval či vytvořil nové HMI propojení. Dále již následuje úprava XML souborů, které se budou generovat.

Nejdříve se přepisují události všech obrazovek, které se budou posléze importovat. Přepis funguje tak, že se prvně hledá HMI tag tabulka. Pokud není žádná nalezena, vytvoří se nová. Poté se pro každou událost kontroluje, zda je již vytvořen odpovídající HMI tag. Pakliže ano, načte se. V opačném případě se vytvoří nový. Na závěr již zbývá přepsat název události v XML souboru pro obrazovku tak, aby odpovídal HMI tagu. Dále v tomto souboru dochází k přepsání velikosti a pozice jednotlivých komponent obrazovky následované importem do TIA Portalu pomocí metody *HMIService.ImportScreens(HMITia)*.

Po importu všech obrazovek se pro všechny HMI tagy vytvoří XML kód, který je následně vkládán do XML souboru pro HMI tag tabulku. Po přidání všech tagů se importuje i tabulka tagů do TIA Portalu použitím *HMIService.ImportHMITagTables(HMITia)* a generování pokračuje opět v sekci projekt.



Obr. 5.6: Dialogové okno - kontrola HMI připojení

Závěr

Cílem práce bylo vytvořit uživatelskou aplikaci, která by umožňovala generovat nový projekt pro PLC a HMI ve vývojovém prostředí TIA Portal. Pomocí vytvořené referenční sady modulů zařízení pro PLC a HMI měla být následně ověřena funkčnost generované aplikace.

Po vytvoření kostry aplikace se ladila struktura programu tak, aby byl program logicky členěn do odpovídajících objektů. Po vytvoření struktury byly jednotlivým objektům postupně přiřazovány funkce potřebné pro jejich konfigurování a následné generování do nového projektu v TIA Portalu. Nejobtížnější a stále zdokonalovanou problematikou bylo umožnění přivedení proměnných na vstupy/výstupy bloků a následné programové volání těchto bloků. V konečném řešení této problematiky je umožněno nejen mapovat vstupy/výstupy bloků, tagů a událostí obrazovek, ale i zapisování konstantních hodnot na piny bloků. I přesto, že je vytvořený způsob dostačující a funkční, je právě zde největší prostor pro zdokonalení aplikace, který spočívá v převodu přiřazování proměnných z textové podoby do grafické. Výhodou však je, že veškerá funkcionalita je obsažena v objektech, a tak po případném vylepšení vizuální stránky aplikace je zachována struktura programu.

Vytvořená aplikace disponuje hned několika funkcemi, které zásadně zefektivní vytváření nových projektů. Jedná se například o vytváření tagů z CSV souboru, rychlé vkládání žádaných bloků a obrazovek na jejichž piny, respektive události, je umožněno přivést proměnné. Největším pozitivem pak je umožnění kopírování konstantních hodnot mezi bloky a možnost přiřazení obrazovkám UDT struktury, přičemž se na události automaticky přiřadí odpovídající proměnná z této struktury. Tyto dvě posledně jmenované funkce zrychlují vytváření nových projektů nejpodstatněji. Dalším významným pozitivem je možnost uložit nedokončenou konfiguraci do XML souboru a opětovně ji načíst. Takto uložená konfigurace může rovněž sloužit jako předpis pro vygenerování projektu.

Jedním z bodů zadání této diplomové práce bylo i vytvoření sady modulů zařízení pro PLC a HMI. Byly vytvořeny dva moduly pro ovládání ventilu a jeden pro ovládání motoru. Dále byly vytvořeny univerzální bloky pro převod analogových veličin.

K ověření funkčnosti generované demonstrační aplikace bylo využito simulačního nástroje Factory I/O, kde byly vytvořeny dva simulační scénáře. První simulující buňku pro tvorbu sody z projektu Barman a druhý pro třídění materiálů. Díky těmto scénářům byla skutečně otestována jednak referenční sada modulů zařízení pro PLC a HMI, tak i funkčnost vygenerovaného projektu pro daný scénář. Dále bylo pomocí těchto scénářů ověřeno, že lze generovat různé programy a aplikace má univerzální využití.

Literatura

[1] Lekce 1 - Úvod do objektově orientovaného programování v C#. *Itnetwork.cz - Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další.* [online]. [cit. 2020-01-02]. Dostupné z: <https://www.itnetwork.cz/csharp/oop/c-sharp-tutorial-uvod-do-objektove-orientovaneho-programovani>

[2] C# OOP (Object-Oriented Programming). *W3Schools Online Web Tutorials* [online]. [cit. 2020-01-02]. Dostupné z: <https://www.w3schools.com/cs/cs_oop.asp>

[3] JANEČEK, Pavel. *Automatické generování PLC programu pomocí TIA Portal Openness.* Brno, 2018, 62 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce: Ing. Jakub Arm. Dostupné z: <https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=171303>

[4] Třída (programování). *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2020-01-02]. Dostupné z: <https://cs.wikipedia.org/wiki/T%C5%99%C3%ADda_(programov%C3%A1n%C3%AD)>

[5] RICHTER, Miroslav. *Praktické programování v C++* [online]. Brno, 2017 [cit. 2020-01-02]. Dostupné z: <http://www.uamt.feec.vutbr.cz/~richter/vyuka/XPPC/prednes.pdf>

[6] Dědičnost (objektově orientované programování). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2020-01-02]. Dostupné z: <https://cs.wikipedia.org/wiki/D%C4%9Bdi%C4%8Dnost_(objektov%C4%9B_orientovan%C3%A9_programov%C3%A1n%C3%AD)>

[7] Lekce 8 - Aréna s mágem (dědičnost a polymorfismus). *Itnetwork.cz - Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další.* [online]. [cit. 2020-01-02]. Dostupné z: <https://www.itnetwork.cz/csharp/oop/c-sharp-tutorial-dedicnost-a-polymorfismus-arena-s-magem>

[8] Lekce 9 - Statika. *Itnetwork.cz - Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další.* [online]. [cit. 2020-01-02]. Dostupné z: <https://www.itnetwork.cz/csharp/oop/c-sharp-tutorial-statika-staticke-atributy-metody-tridy-konstruktor>

[9] Lekce 16 - Abstraktní třída, porovnávání a přetěžování operátorů. *Itnetwork.cz - Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další.* [online]. [cit. 2020-01-02]. Dostupné z: <https://www.itnetwork.cz/csharp/oop/c-sharp-net-tutorial-abstraktni-trida-icomparable-pretizeni-operatoru>

[10] SIEMENS. *SIMATIC Automating projects with scripts: System Manual* [online]. Mnichov: SIEMENS, 2017 [cit. 2020-01-02]. Dostupné z: <https://cache.industry.siemens.com/dl/files/163/109477163/att_926042/v1/TIAPortalOpennessenUS_en-US.pdf>

[11] Working with FileInfo and DirectoryInfo classes. *C# Corner - A Social Community of Developers and Programmers* [online]. 2009 [cit. 2020-01-02]. Dostupné z: <http://www.c-sharpcorner.com/UploadFile/skumaar_mca/working-with-fileinfo-and-directoryinfo-classes/>

[12] OPÍCHAL, Antonín. *Řízení a automatizace procesů pomocí distribuovaných sítí systémů firmy Siemens* [online]. Olomouc, 2016 [cit. 2020-01-02]. Dostupné z: <http://docplayer.cz/27199566-Diplomova-prace-univerzita-palackeho-v-olomouci-rizeni-/a-automatizace-procesu-pomoci-distribuovanych-siti-systemu-firmy/-siemens.html> DIPLOMOVÁ PRÁCE. Univerzita Palackého. Vedoucí práce Doc. RNDr. Jiří Pechoušek, Ph.D.

[13] *Siemens TIA Portal – jednotné vývojové prostředí pro automatizaci v průmyslu* [online]. Ústí nad labem: Časopis Automa pro automatizační techniku, 2011, 11(3) [cit. 2020-01-02]. Dostupné z: <http://automa.cz/cz/casopis-clanky/siemens-tia-portal-jednotne-vyvojove-prostredi-pro-automatizaci-/v-prumyslu-2011_03_43212_6058/>

[14] Programovací jazyky pro PLC. *COPTel – výukový portál SŠ-COPT Kroměříž* [online]. Kroměříž, 24.11.2019 [cit. 2020-01-02]. Dostupné z: <https://coptel.cz/mod/page/view.php?id=6737>

[15] KOVÁŘ, Josef, Zuzana PROKOPOVÁ a Ladislav ŠMEJKAL, CSC. *Programování PLC* [online]. Praha [cit. 2020-01-02]. Dostupné z: <http://www.spszl.cz/soubory/plc/programovani_plc.pdf>

[16] KADLENČÍK, Pavel. *Možnosti využití programovatelných automatů Simatic řady S7-1200 od firmy Siemens* [online]. Zlín, 2017 [cit. 2020-01-02]. Dostupné z: <https://digilib.k.utb.cz/bitstream/handle/10563/41283/

kadle%C4%8D%C3%ADk_2017_dp.pdf?sequence=1> Bakalářská práce. Univerzita Tomáše Bati. Vedoucí práce Ing. Pavel Nevrátil, Ph.D.

[17] SIEMENS. *SCE Training Curriculum pro Integrované Automatizační Řešení Totally Integrated Automation (TIA): Blokové typy pro SIMATIC S7-1200* [online]. 2012 [cit. 2020-01-02]. Dostupné z: <https://assets.new.siemens.com/siemens/assets/api/uuid:cc24966d-efef-4ba2-bdcd-cc7aac19c135/SCE-CZ-010-020-R1209-S7-1200-Blocks.pdf>

[18] *Nový TIA Portal vám urychlí a zjednoduší práci* [online]. Praha: TT | Technický týdeník, 2017 [cit. 2020-01-02]. Dostupné z: <https://www.technickytydenik.cz/rubriky/archiv-technik/novy-tia-portal-vam-urychli-a-zjednodusi-praci_40558.html>

[19] *Siemens představuje vizualizační software WinCC v rámci nové verze TIA Portal V14* [online]. Ústí nad labem: Časopis Automa pro automatizační techniku, 2016, 16(11) [cit. 2020-01-02]. Dostupné z: <http://automa.cz/cz/casopis-clanky/siemens-predstavuje-vizualizacni-software-wincc-v-ramci-nove-ver\ze-tia-portal-v14-2016_11_0_9197/>

[20] KVÁČ, Josef. *TIA Portal Openness Generování projektu* [online]. Praha, 2016 [cit. 2020-01-02]. Dostupné z: <https://docplayer.cz/30075547-Tia-portal-openness-generovani-projektu-https-workspace/-automation-siemens-com-content.html> Prezentace.

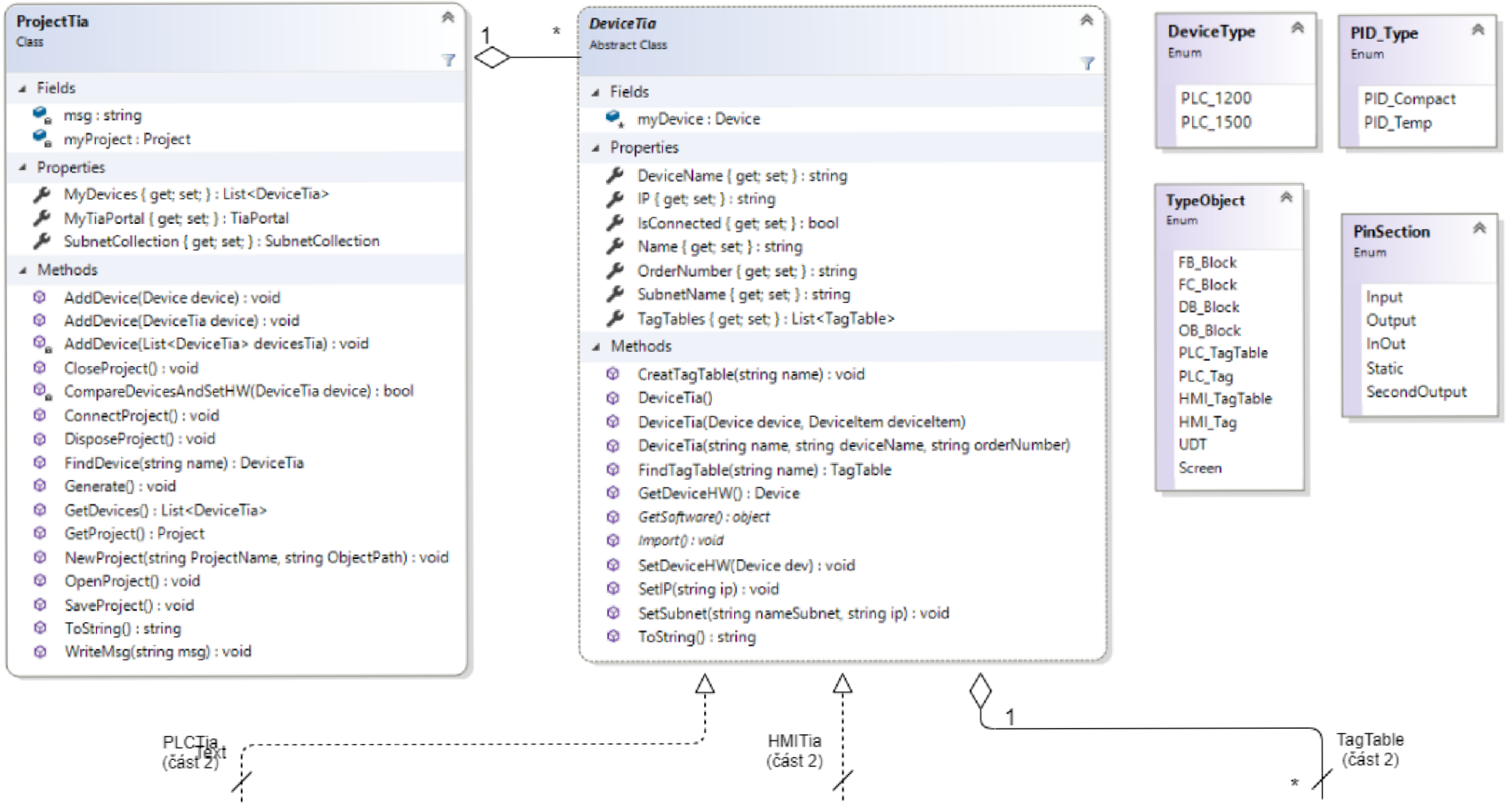
[21] Výška úseče kruhu. *Matematické vzorce* [online]. [cit. 2020-03-15]. Dostupné z: <http://vzorce-matematika.hys.cz/vyska-usece-kruhu-kruznice.php>

[22] *SIMATIC S7-1200, S7-1500 PID control: Function manual* [online]. 11/2019 [cit. 2020-04-14]. DOI: A5E35300227-AE. Dostupné z: <https://support.industry.siemens.com/cs/document/108210036/simatic-s7-1200-s7-1500-pid-control?dti=0&lc=en-WW>

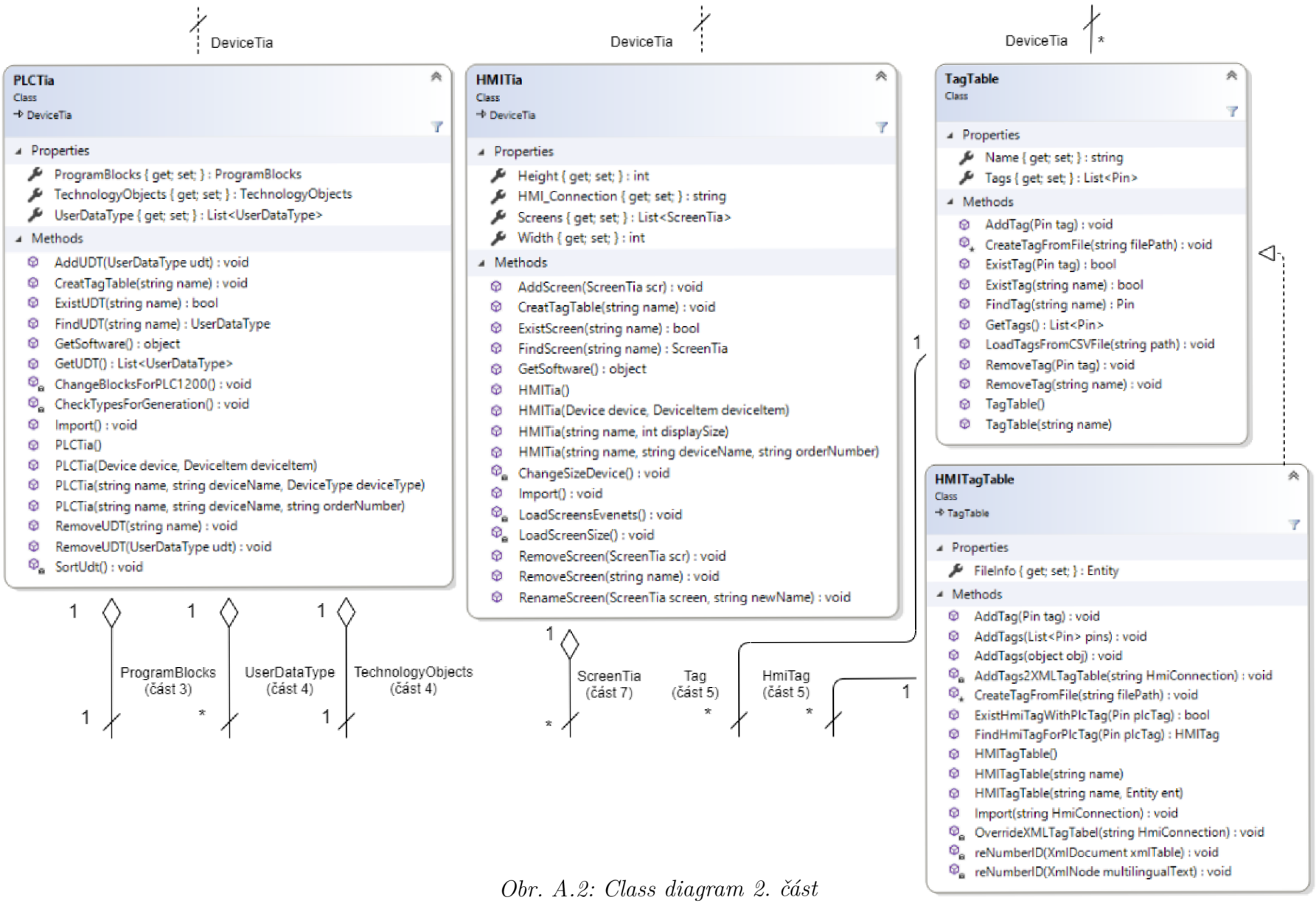
[23] Items - Factory I/O - Documentation. *Factory I/O - Documentation* [online]. Factory I/O [cit. 2020-05-06]. Dostupné z: <https://docs.factoryio.com/manual/parts/items/>

Seznam příloh

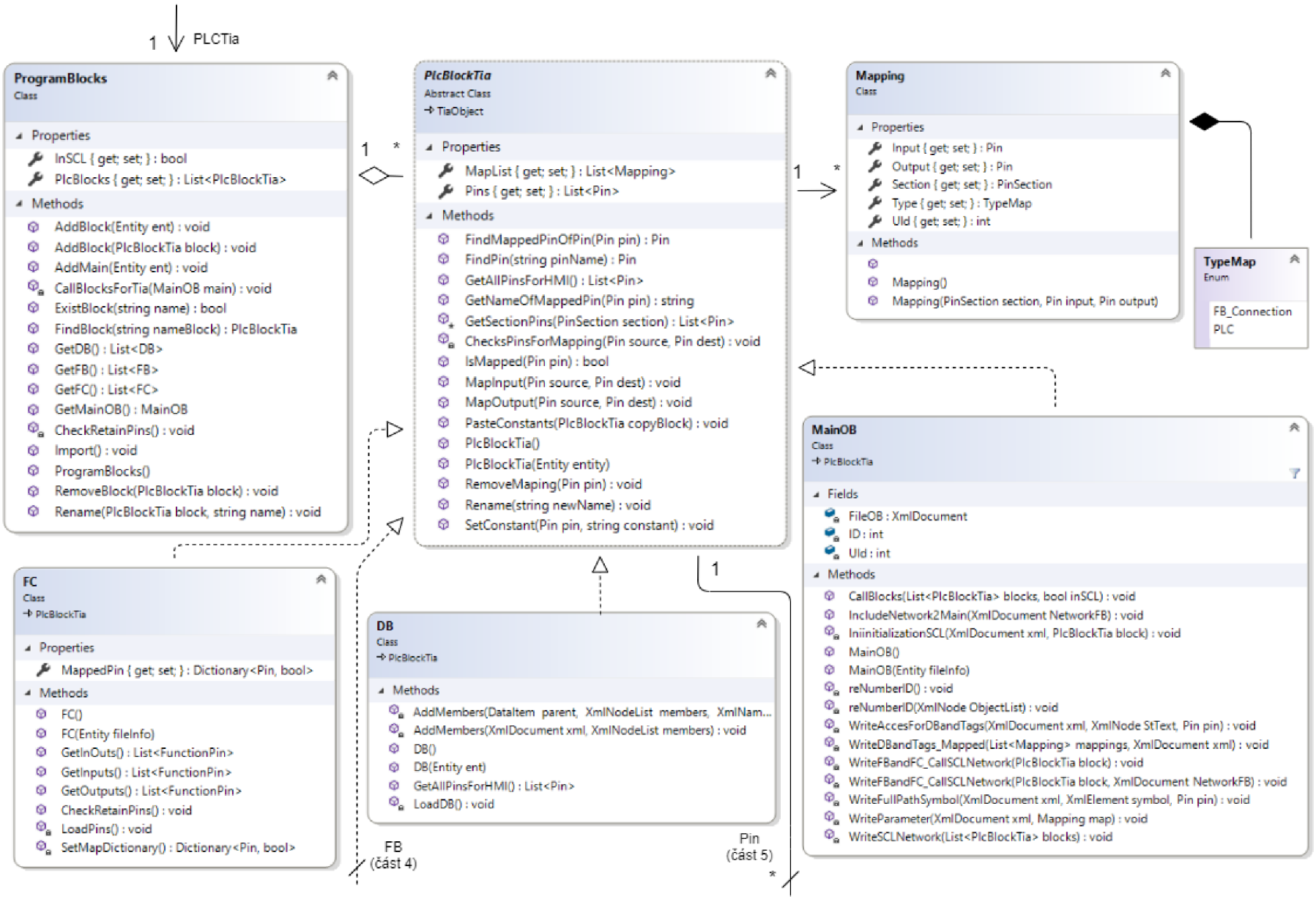
A Class diagram



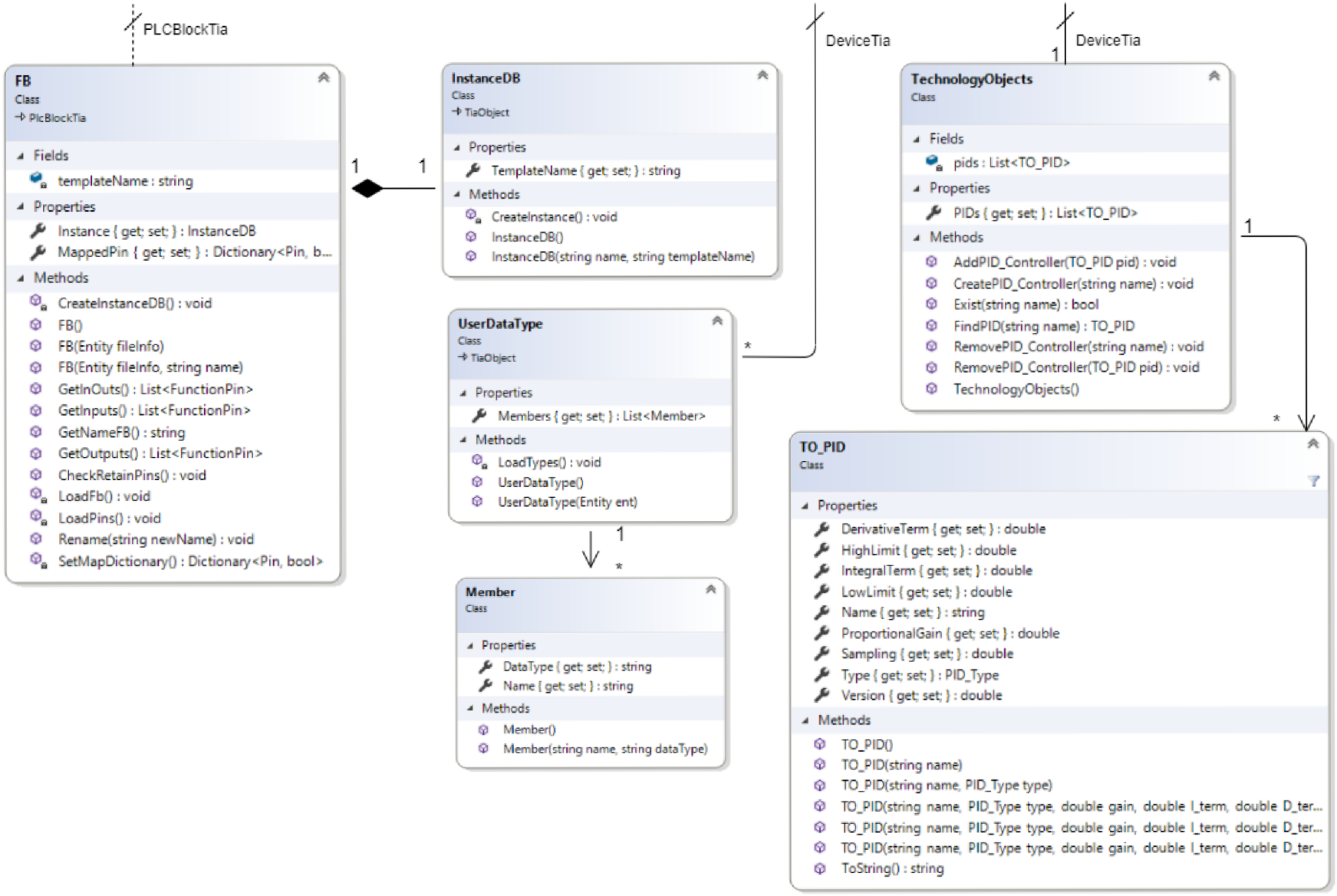
Obr. A.1: Class diagram 1. část



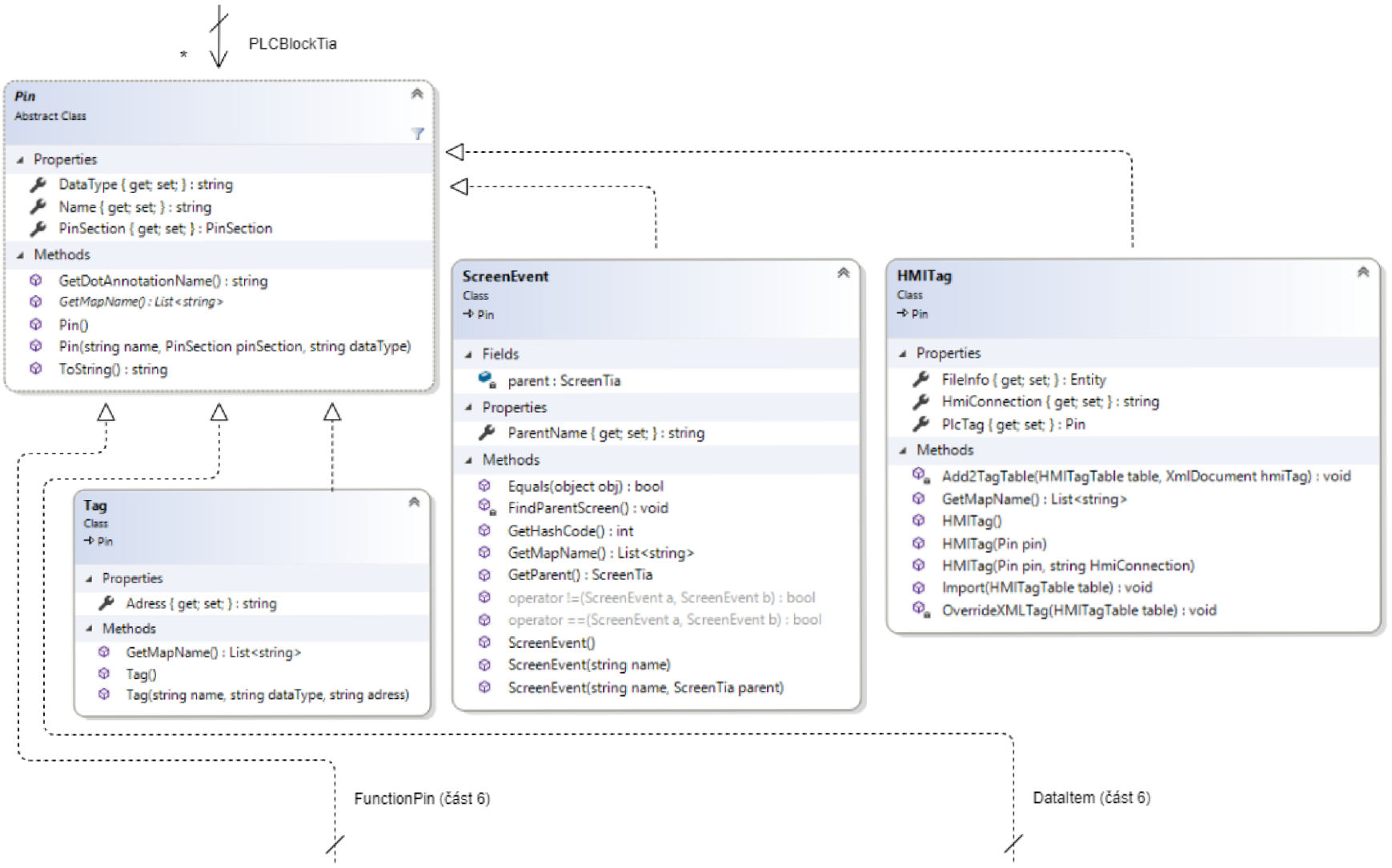
Obr. A.2: Class diagram 2. část



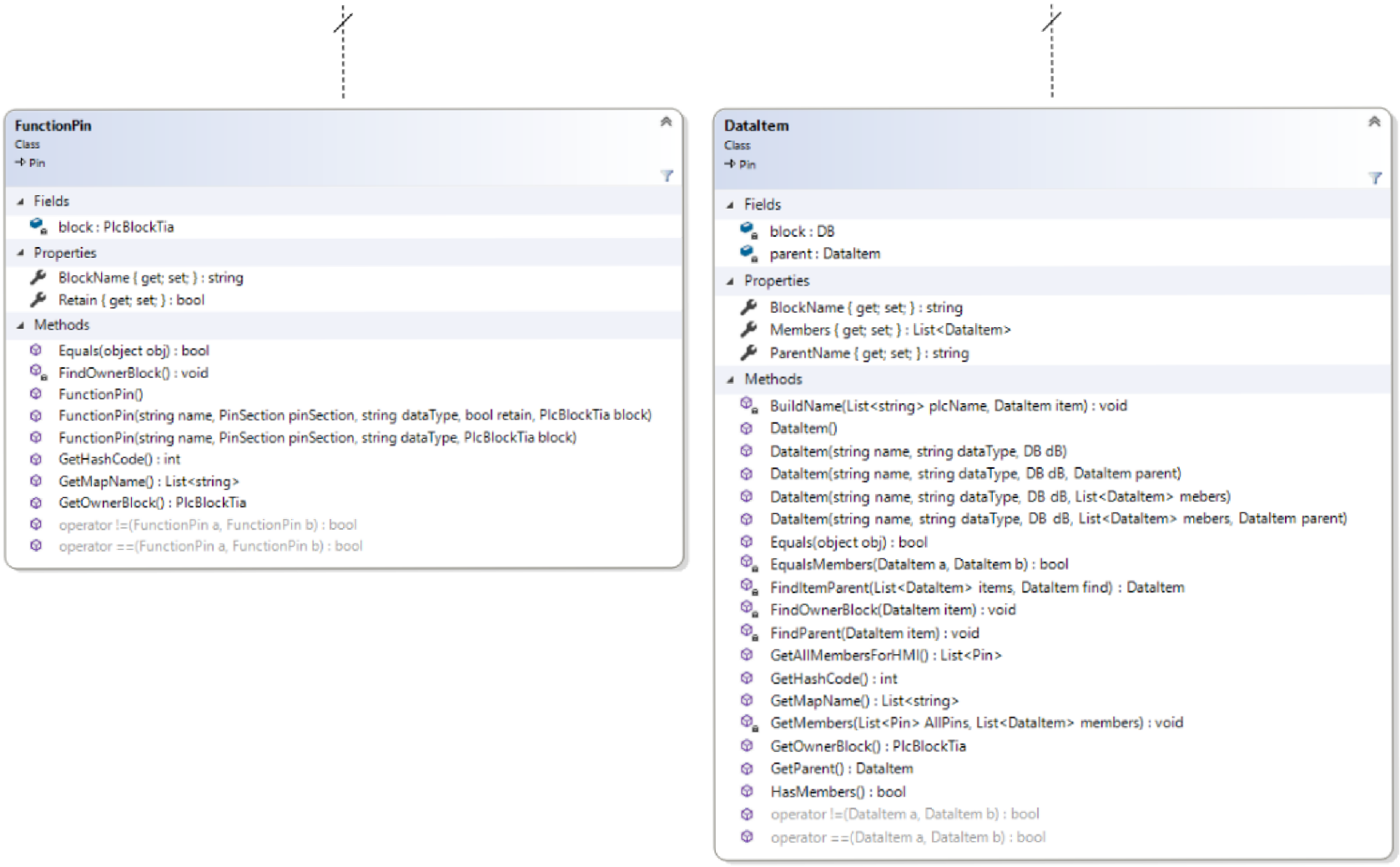
Obr. A.3: Class diagram 3. část



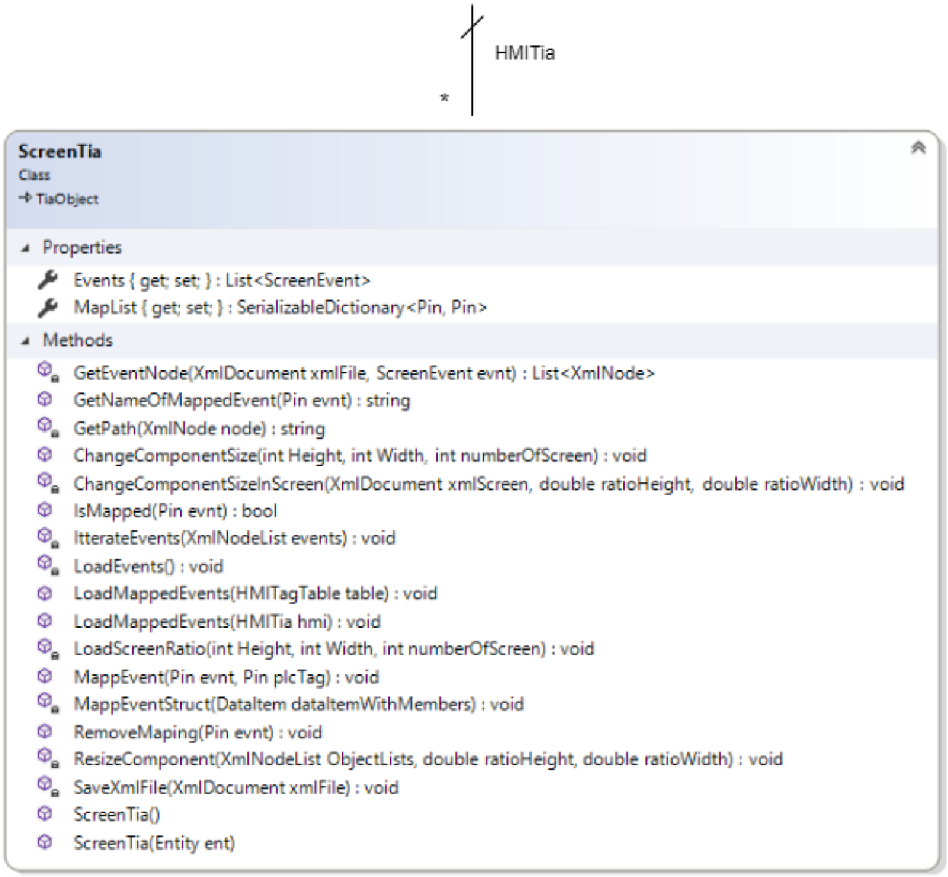
Obr. A.4: Class diagram 4. část



Obr. A.5: Class diagram 5. část



Obr. A.6: Class diagram 6. část



Obr. A.7: Class diagram 7. část

B XML kód pro prázdný network

```
<Document>
  <SW.Blocks.CompileUnit ID="3" CompositionName="CompileUnits">
    <AttributeList>
      <NetworkSource>
        <FlgNet>
          <Parts>
            <Call UId="24">
              <CallInfo Name="NameFB_Block" BlockType="FB">
                <Instance Scope="GlobalVariable" UId="25">
                  <Component Name="NameInstanceDB" />
                </Instance>
              </CallInfo>
            </Call>
          </Parts>
          <Wires>
            <Wire UId="30">
              <Powerrail />
              <NameCon UId="24" Name="en" />
            </Wire>
          </Wires>
        </FlgNet>
      </NetworkSource>
      <ProgrammingLanguage>LAD</ProgrammingLanguage>
    </AttributeList>
    <ObjectList>
      <MultilingualText ID="4" CompositionName="Comment">
        <ObjectList>
          <MultilingualTextItem ID="5" CompositionName="Items">
            <AttributeList>
              <Culture>en-US</Culture>
              <Text />
            </AttributeList>
          </MultilingualTextItem>
        </ObjectList>
      </MultilingualText>
      <MultilingualText ID="6" CompositionName="Title">
        <ObjectList>
          <MultilingualTextItem ID="7" CompositionName="Items">
            <AttributeList>
              <Culture>en-US</Culture>
              <Text />
            </AttributeList>
          </MultilingualTextItem>
        </ObjectList>
```

```
      </MultilingualText>
    </ObjectList>
  </SW.Blocks.CompileUnit>
</Document>
```

Výpis B.1: XML kód pro prázdný LAD network

```
<Document>
  <SW.Blocks.CompileUnit ID="3" CompositionName="CompileUnits">
    <AttributeList>
      <NetworkSource>
        <StructuredText>
        </StructuredText>
      </NetworkSource>
      <ProgrammingLanguage>SCL</ProgrammingLanguage>
    </AttributeList>
    <ObjectList>
      <MultilingualText ID="4" CompositionName="Comment">
        <ObjectList>
          <MultilingualTextItem ID="5" CompositionName="Items">
            <AttributeList>
              <Culture>en-US</Culture>
              <Text />
            </AttributeList>
          </MultilingualTextItem>
        </ObjectList>
      </MultilingualText>
      <MultilingualText ID="6" CompositionName="Title">
        <ObjectList>
          <MultilingualTextItem ID="7" CompositionName="Items">
            <AttributeList>
              <Culture>en-US</Culture>
              <Text />
            </AttributeList>
          </MultilingualTextItem>
        </ObjectList>
      </MultilingualText>
    </ObjectList>
  </SW.Blocks.CompileUnit>
</Document>
```

Výpis B.2: XML kód pro prázdný SCL network

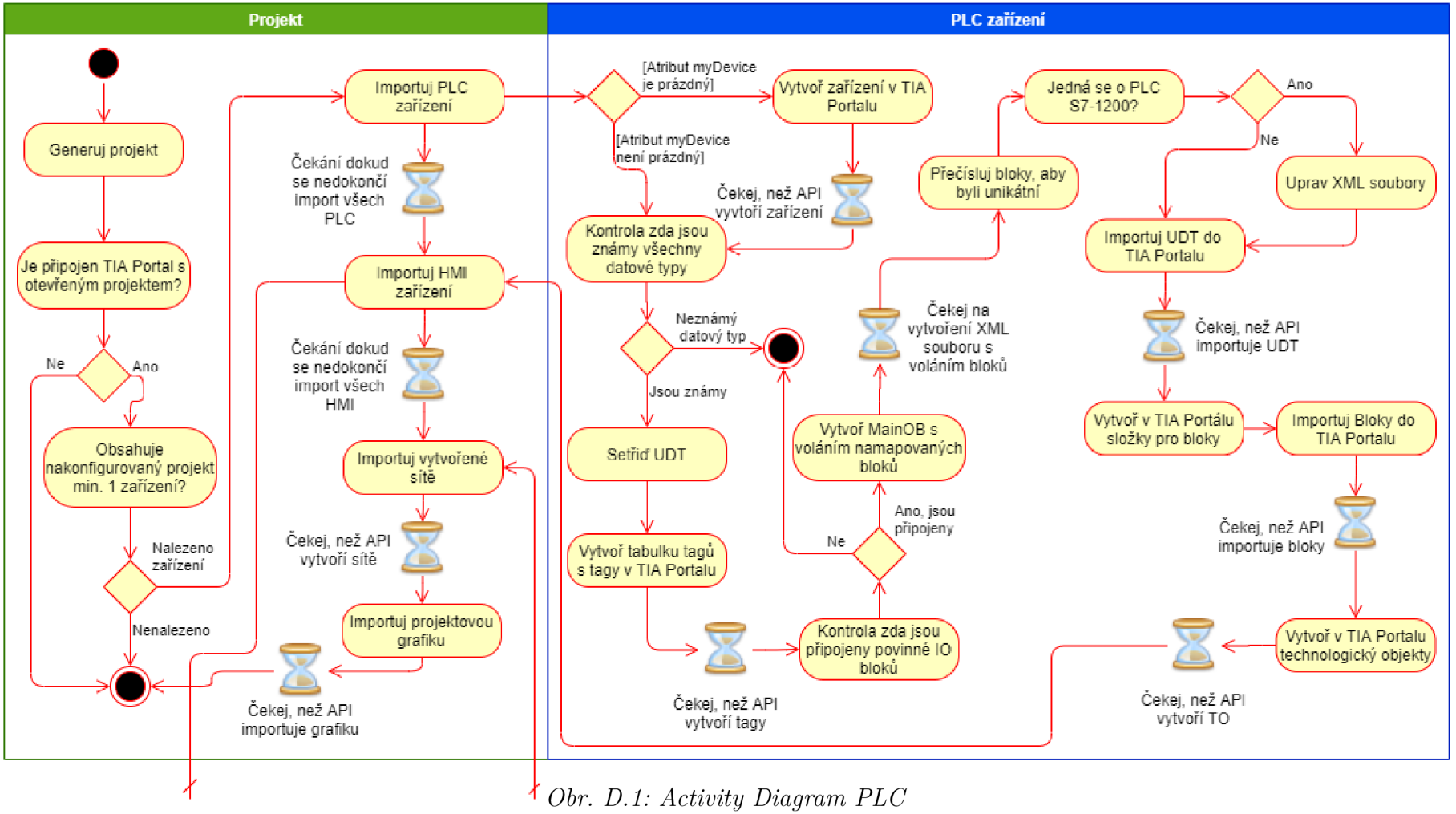
C Výsek z XML souboru pro HMI obrazovku

```
<Hmi.Screen.Screen ID="0">
    <AttributeList>
      <ActiveLayer>0</ActiveLayer>
      <BackColor>182, 182, 182</BackColor>
      <GridColor>0, 0, 0</GridColor>
      <Height>480</Height>            //Výška obrazovky
      <Name>MainScr</Name>            //Jméno obrazovky
      <Number>1</Number>             //Pořadové č. obr.
      <Visible>true</Visible>
      <Width>800</Width>             //Šířka obrazovky
    </AttributeList>
    .
    .
    .
    //Vrstva (Layer)
    <Hmi.Screen.ScreenLayer ID="3" CompositionName="Layers">
        <AttributeList>
          <Index>1</Index>
          <Name>DefaultScreen</Name>      //Jméno vrstvy
          <VisibleES>true</VisibleES>
        </AttributeList>
        <ObjectList>

      //Jedna z komponent ve vrstvě - Objekt kruh (Circle)

          <Hmi.Screen.Circle ID="D0" CompositionName="ScreenItems">
            <AttributeList>
              <BackColor>0, 255, 0</BackColor>
              <BackFillStyle>Solid</BackFillStyle>
              <BorderColor>24, 28, 49</BorderColor>
              <BorderWidth>3</BorderWidth>
              <EdgeStyle>Solid</EdgeStyle>
              <Flashing>None</Flashing>
              <Height>20</Height>       //Výška komponenty
              <Left>122</Left>          //Levé zarovnání
              <ObjectName>Circle_1</ObjectName>
              <Radius>10</Radius>       //Rádius
              <TabIndex>-1</TabIndex>
              <Top>149</Top>           //Horní zarovnání
              <UseDesignColorSchema>false</UseDesignColorSchema>
              <Width>20</Width>         //Šířka komponenty
            </AttributeList>
            .
            .
```

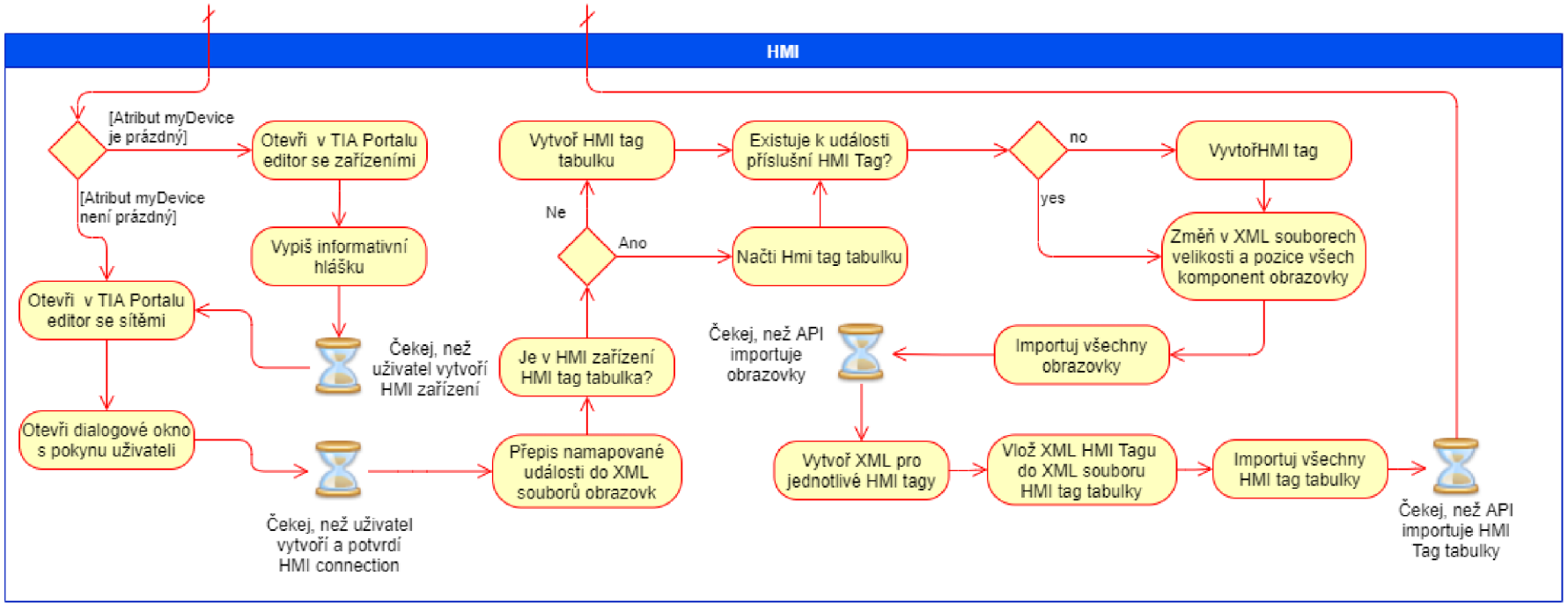
```
            .
        </Hmi.Screen.Circle>
        .
        .
        .
    </Hmi.Screen.ScreenLayer>
    .
    .
    .
</Hmi.Screen.Screen>
```

Výpis C.1: XML kód pro HMI obrazovku

D Activity diagram pro generování projektu



Obr. D.1: Activity Diagram PLC



Obr. D.2: Activity Diagram HMI

E Struktura předpisu

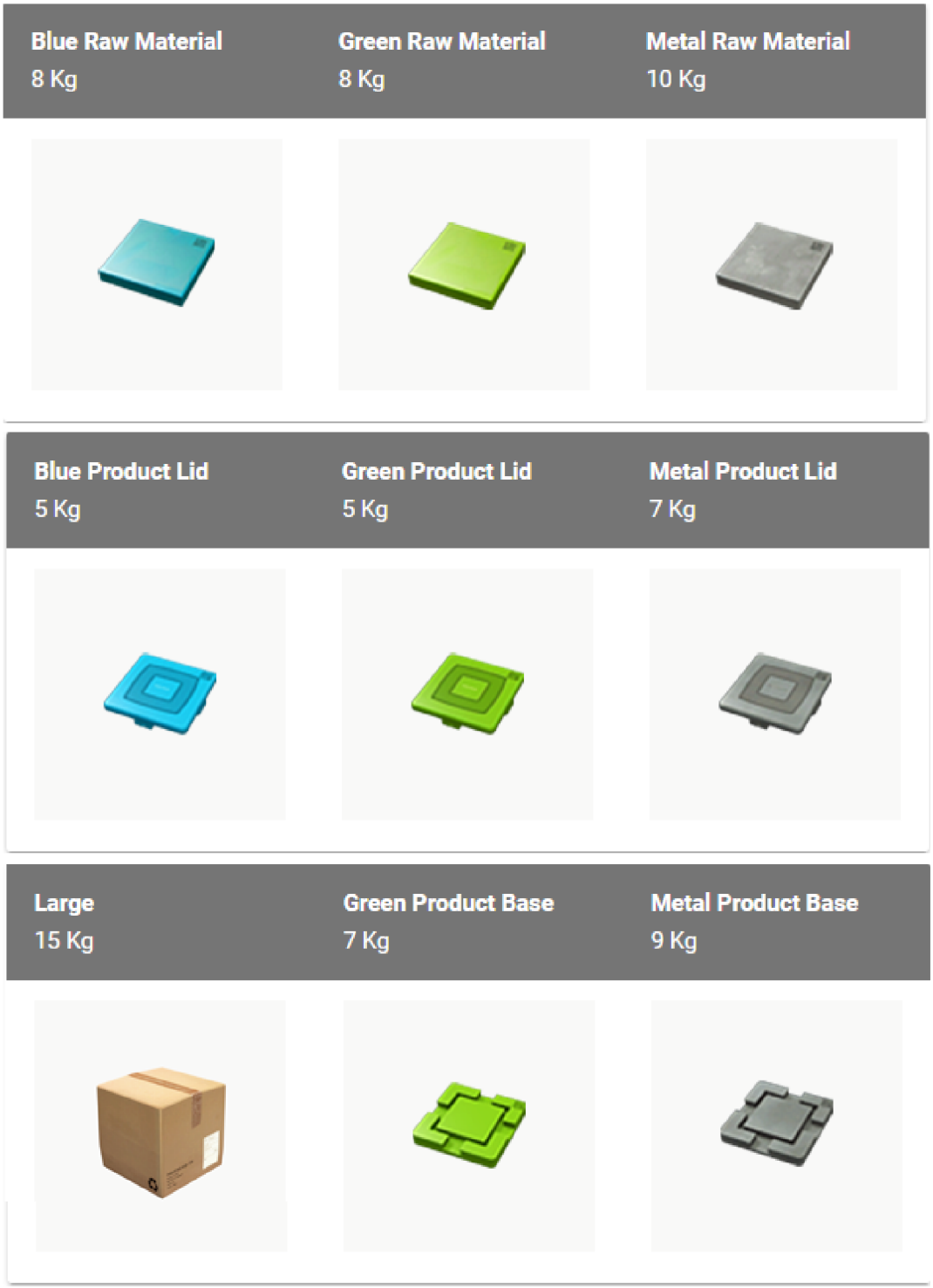
```
<?xml version="1.0" encoding="utf-8"?>
<Project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="Mynamespace"
    >
  <DeviceTia xsi:type="PLCTia">
    <Name>PLC_1</Name>
    <DeviceName>S71500/ET200MP-Station_1</DeviceName>
    <OrderNumber>OrderNumber:6ES7 511-1AK00-0AB0/V1.8</OrderNumber>
    <TagTables>
      <TagTable Name="TableName">
        <Tags>
          <Pin xsi:type="Tag" Name="TagName" Section="Input" Type="
              Bool" Adress="I20.0" />
        </Tags>
      </TagTable>
    </TagTables>
    <ProgramBlocks>
      <PlcBlocks>
        <PlcBlockTia xsi:type="FB" Name="BlockName">
          <FileInfo Path="FilePath.xml" FileName="FileName.xml" />
          <Pins xsi:type="FunctionPin" Name="iRun" Section="Input"
              Type="Bool" Remanence="false" BlockName="BlockName" />
          <MapList Type="FB_Connection">
            <Input xsi:type="FunctionPin" Name="iRun" Section="
                Input" Type="Bool" Remanence="false" BlockName="
                BlockName" />
            <Output xsi:type="FunctionPin" Name="Run" Section="
                Output" Type="Bool" Remanence="false" BlockName="
                ConnectedBlockName" />
            <UId>0</UId>
            <Section>Input</Section>
          </MapList>
        </PlcBlockTia>
      </PlcBlocks>
      <InSCL>true</InSCL>
    </ProgramBlocks>
    <UserDataType>
      <UserDataType Name="tMotor">
        <FileInfo Path="FilePath.xml" FileName="FileName.xml" />
        <Members>
          <Member Name="Cmd" Type="Struct" />
          <Member Name="State" Type="Struct" />
        </Members>
      </UserDataType>
```

```
    </UserDataType>
    <TechnologyObjects>
      <PIDs>
        <TO_PID Name="PID_1" Type="PID_Compact" Version="2.3">
          <ProportionalGain>1</ProportionalGain>
          <IntegralTerm>20</IntegralTerm>
          <DerivativeTerm>0</DerivativeTerm>
          <Sampling>1</Sampling>
          <HighLimit>100</HighLimit>
          <LowLimit>0</LowLimit>
        </TO_PID>
      </PIDs>
    </TechnologyObjects>
  </DeviceTia>
  <DeviceTia xsi:type="HMITia">
    <Name>HMI_1</Name>
    <DeviceName>HMI_RT_1</DeviceName>
    <OrderNumber>OrderNumber:6AV2 124-0JC01-0AX0/15.1.0.0</
       OrderNumber>
    <TagTables />
    <Screens>
      <ScreenTia Name="Conveyor_M100">
        <FileInfo Path="FilePath.xml" FileName="FileName.xml" />
        <Events>
          <ScreenEvent Name="State_Error" Section="InOut" Type="
             Event" ParentName="Motor" />
        </Events>
        <MapList>
          <item key="First Item" value="Second Item">
            <Pin xsi:type="ScreenEvent" Name="Cmd_Man/Aut" Section=
               "InOut" Type="Event" ParentName="Motor" xmlns="" />
            <Pin xsi:type="DataItem" Name="Man/Aut" Section="InOut"
                Type="Bool" DataBlockName="OwnerBlock" Parent="
               ParentName" xmlns="" />
          </item>
        </MapList>
      </ScreenTia>
    </Screens>
  </DeviceTia>
</Project>
```

Výpis E.1: XML předpis pro PLC a HMI

F Tříděné materiály pro projekt Sorter

Tříděné materiály



Obr. F.1: Simulace třídičky - Tříděné materiály [23]

G Obsah přiloženého CD

- `PRILOHA_DP_JANECEK_PAVEL` — Kořenový adresář přiloženého CD
  - `Janecek_DP.pdf` — Elektronická verze bakalářské práce
  - `Aplikace` — Složka s aplikací průvodce generováním projektů
  - `PredpisyGenerovani` — XML předpisy generovaných projektů
    - `FactoryIO_SodaMaker.xml` — XML předpis pro generovaní projektu SodaMaker
    - `FactoryIO_Sorter.xml` — XML předpis pro generovaní projektu Sorter
  - `SimulaceFactoryIO` — Vytvořené simulátory ve Factory I/O
    - `SodaMaker` — Simulátor pro SodaMaker
    - `Sorter` — Simulátor pro Sorter
  - `VygenerovaneBloky` — Vygenerované bloky s voláním namapovaných bloků
    - `FB_Call's_SodaMaker.pdf` — Obsah bloku FB_Call's pro projekt SodaMaker
    - `FB_Call's_Sorter.pdf` — Obsah bloku FB_Call's pro projekt Sorter
  - `FactoryIO_SodaMaker` — Vygenerovaný projekt pro SodaMaker

Byly použity následující verze programu: TIA Portal V15.1., Factory I/O 2.4.2.