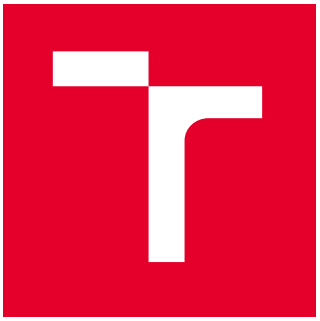


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

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

IMPLEMENTACE ASSET ADMINISTRATION SHELL

ASSET ADMINISTRATION SHELL IMPLEMENTATION

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jakub Maslowski**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Benešl**

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Jakub Maslowski ***ID:*** 221003

Ročník: 2 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Implementace Asset Administration Shell

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout a implementovat AAS do PLC. Současně bude vytvořena aplikace Service Requester, která bude, dle požadavků VDI/ZVEI, komunikovat se standardizovaným AAS.

1. Objasněte pojem Asset Administration Shell.
2. Rozšiřte funkcionalitu AAS v PLC o možnosti bufferu operací – fronty.
3. Navrhněte software sloužící jako AAS „Service Requester“.
4. Definujte informační a datový model komunikačního rozhraní a navrhnuté aplikace.
5. Realizujte SW.
6. Diskutujte dosažené výsledky a jejich potenciální využití.

DOPORUČENÁ LITERATURA:

Bader, S.R. and Maleshkova, M., 2019. The semantic asset administration shell. In Semantic Systems. The Power of AI and Knowledge Graphs: 15th International Conference, SEMANTiCS 2019, Karlsruhe, Germany, September 9–12, 2019, Proceedings 15 (pp. 159-174). Springer International Publishing.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 15.5.2024

Vedoucí práce: Ing. Tomáš Benešl

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato diplomová práce navazuje na autorovu bakalářskou práci a postupně shrnuje poznatky ohledně I4.0, DT a především pak AAS. Soustředí se na popis AAS s ohledem na nezbytné kompromisy od teoretického konceptu AAS pro její praktickou implementaci ve výrobním procesu.

Rozsáhlá praktická část se zabývá implementací AAS v decentralizované produkci do PLC systému firmy Siemens řady S7-1500 s OPC UA servery včetně individuálního řešení aplikace prioritních front. Protistranu reprezentující produkty zastupuje centrální AAS Hub Client v jazyce Python. Výsledky prezentuje ve formě testování na síti 29 virtuálních PLC.

KLÍČOVÁ SLOVA

Asset Administration Shell, AAS, Decentralizovaná výroba, OPC UA, S7-1500 PLC, Průmysl 4.0

ABSTRACT

This master's thesis builds upon the author's bachelor's thesis and gradually summarizes insights regarding Industry 4.0, Digital Twins, and primarily AAS. It focuses on describing AAS with regard to the necessary compromises from the theoretical concept of AAS to its practical implementation in the manufacturing process.

The extensive practical part deals with the implementation of AAS in decentralized production into Siemens S7-1500 series PLC systems with OPC UA servers, including an individual solution for prioritized queue application. The counterpart representing products is a central AAS Hub Client implemented in Python. The results are presented in the form of testing on a network of 29 virtual PLCs.

KEYWORDS

Asset Administration Shell, AAS, Decentralized Production, OPC UA, S7-1500 PLC, Industry 4.0

Vysázeno pomocí balíčku `thesis` verze 4.09; `https://latex.fekt.vut.cz/`

MASLOWSKI, Jakub. *Implementace Asset Administration Shell*. Diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2024. Vedoucí práce: Ing. Tomáš Benešl

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Jakub Maslowski
VUT ID autora:	221003
Typ práce:	Diplomová práce
Akademický rok:	2023/24
Téma závěrečné práce:	Implementace Asset Administration Shell

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné 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 závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné 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 ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing.Tomáši Benešlovi za odborné vedení práce. Dále pak mé snoubence a rodině za dlouhotrvající podporu v průběhu studia.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V dnešním rychle se vyvíjejícím technologickém prostředí je stále rostoucí snaha o praktické zavádění částí průmyslu 4.0 do praxe. Je však nutné vždy analyzovat, zdali je to pro dané případy výhodné, co se týče poměru zvýšené ceny z pohledu jejich aplikací ku jejich výhodám, které by potenciálně mohly přinést.

Jednou z klíčových složek průmyslu 4.0 je Asset Administration Shell (AAS). Tato práce přináší podrobný pohled na problematiku AAS a zahrnuje požadavky a doporučení předních výzkumníků v této oblasti, zejména německé platformy Platform Industrie 4.0 a mezinárodní asociace IDTA[1]. Neopomíjí však její limitace a postupně zmiňuje oblasti, ve kterých je AAS pozměněno od jakéhosi standardu, uvedeného řečenými uskupeními.

Výzkum AAS je v době psaní práce soustředěn pouze na její obchodní a marketingovou stránku. Praktické aplikace AAS jsou nevídané, proto je tato práce vedena stylem **proof of concept**, kde namísto optimalizace na určitou výrobu zkouší a analyzuje různé přístupy implementace AAS.

Cílem této práce je, jak už z názvu vyplývá, implementovat Asset Administration Shell do prostředí simulující výrobu, kde bude možné otestovat její schopnosti zefektivnit onu výrobu a zároveň přehledně poskytovat informace o jejím stavu tak, aby bylo možné sledovat a upravovat jednotlivé procesy ve výrobě. Dále se taktéž zabývá tím, aby veškeré komunikace uvnitř procesů byly dostatečně zabezpečeny a zároveň bere ohledy na nutnou interoperabilitu.

Diplomová práce úzce navazuje na autorovu bakalářskou práci [1] a významně rozšiřuje možnosti implementace AAS do PLC, které zde byly představeny. Samotné téma práce bylo vybráno z důvodu jejího enormního potenciálu zefektivnit aplikace AAS do průmyslu.

[1] International Digital Twin Asociation

2 Úvod do problematiky AAS v souvislostech

Pro správné pochopení AAS je nezbytné uvést jeho zařazení a význam mezi ostatními pojmy, často označovanými jako tzv. buzzwords. Tedy pojmy, pod kterými si průměrný čtenář představí něco jiného, než co tyto pojmy skutečně označují. Z tohoto důvodu se tato kapitola zabývá několika vybranými buzzwords, která mají přímou souvislost s AAS a bez jejich správného pochopení by debata na téma AAS byla dosti obtížná.

2.1 Digitální dvojče (*Digital Twin*)

Koncept digitálního dvojčete není nikterak nový, byl popsán již v roce 1993 v knize *Mirror worlds...* autora *Davida Gelerntera*[2], přestože ho v té době takto nenazývali. Podrobnější koncept byl pak prezentován v roce 2002 Michaelem Grievesem[3]. Vzhledem k správě životního cyklu produktu (PLM) byl pak využit v NASA pro popisy letadel a odtud si našel cestu i do širšího spektra průmyslu, kde jeho velký boom přišel v roce 2016, kdy se začal stále častěji objevovat v odborných publikacích[4].

Je tedy patrné, že jeho vývoj byl velice dlouhý a každý autor si tak pod tímto pojmem mohl představovat něco jiného. Naštěstí v roce 2021 vznikl ISO 23247, udávající seznam pravidel pro tvorbu DT. V obecné rovině se jedná o virtuální model či simulaci reálného objektu uchovávající jeho aktuální i historické data.

Přidává výhody ve formě zvyšování efektivity produkce, lepší preventivní údržbě, dostupnosti, zrychlené možnosti výroby prototypů a s tím spojené snížení cenových nákladů na jeho vývoj. Vše je však zajištěno tvorbou DT, která je finančně náročná na výrobu. Z tohoto důvodu je vhodné rozlišovat DT dle úrovně jeho integrace[5]:

2.1.1 Digitální Model (*Digital Model*)

Jedná se o model, u kterého je přenos dat z fyzického objektu na jeho digitální a naopak prováděn manuálně. Jedná se o finančně nejpřívětivější variantu a z tohoto důvodu je i nejvíce rozšířená. Můžeme si zde představit model motoru, sloužící pro teplotní zkoušky dle jeho materiálů a fyzikálních rozměrů. Takto lze kupříkladu otestovat různé typy motorů bez nutnosti jejich fyzické přítomnosti.

2.1.2 Digitální stín (*Digital Shadow*)

Zde již probíhá přenos dat z jeho fyzického dvojčete automaticky, avšak změny od jeho virtuální instance se musí na fyzickém objektu provádět manuálně. Zde si mů-

žeme představit digitální stín pacienta na operačním sále, kde jsou přes širokou škálu senzorů snímány jeho parametry a veškeré změny musí být provedeny manuálně.

2.1.3 Digitální dvojče (*Digital Twin*)

Umožňuje oboustranný přenos dat. Poskytuje nejširší možnosti výhod, avšak za cenu nemalých finančních nákladů. V tomto případě postačí po představu digitální dvojče inteligentního auta, které aktualizuje informace o tlaku v pneumatikách, jeho zatížení a v případě havárie i aktivace airbagů. Zároveň však umožňuje automaticky měnit jeho stav, kupříkladu zneschopnit pohyb v případě ohlášení krádeže či umožnění vyhřívání sedadel dle předplatného.

2.2 Průmysl 4.0 (*Industry 4.0*)

Snad každý čtenář již slyšel pojem průmysl 4.0. Tento pojem byl prvně představen veřejnosti na mezinárodním veletrhu Hannover Messe 2011 pro automatizaci a postupně nabíral na popularitě. Za jeho vznikem, který byl výrazně podporován německou vládou, stála spolupráce společností BITKOM, reprezentující IT sektor, VDMA, reprezentující továrny, a ZVEI, reprezentující dodavatele elektrických zařízení.[6]

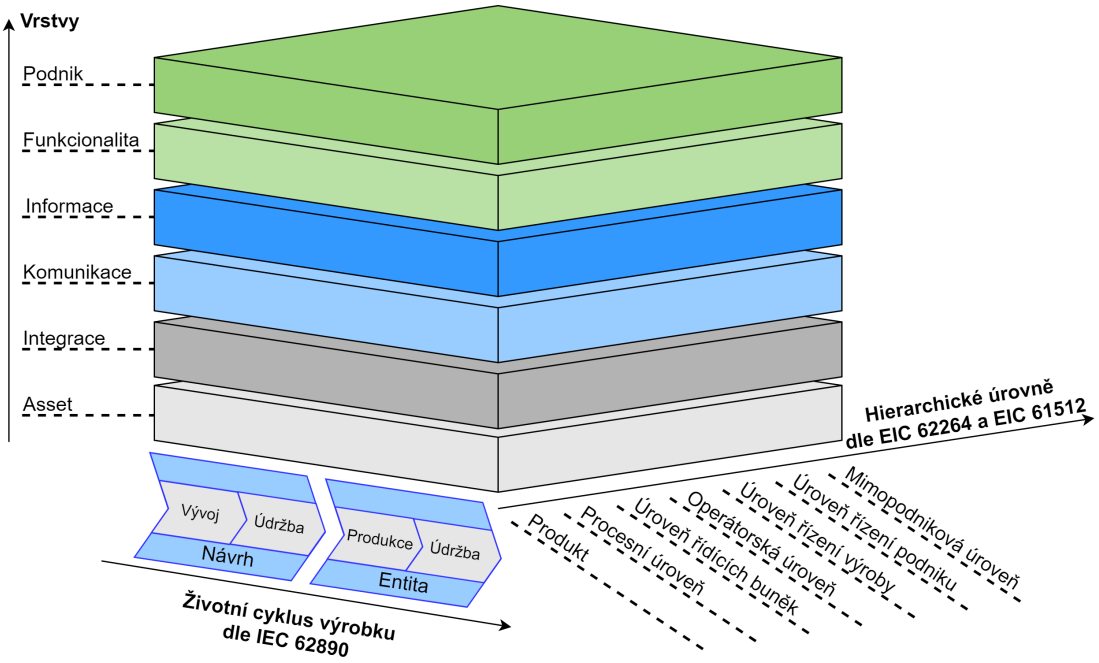
Průmysl 4.0 vychází jednak z přirozeného vývoje výpočetního výkonu, jednak z možností přenosu dat. Jedná se o propojení strojů včetně jejich procesů ve vhodném prostředí tak, aby bylo umožněno efektivní zpracování a skladování dat.[7]

Lze si ho představit pod jeho referenčním modelem architektury, který je zobrazen na obrázku 2.1. Model obsahuje krychli reprezentující tři osy. Může sloužit jako jakýsi pomocník pro komunikaci a uvědomění si, v jaké jeho části se daná aplikace nachází.

2.3 Průmysl 5.0 (*Industry 5.0*)

Průmysl 5.0, někdy nazývaný též Industry 5.0, představuje další etapu vývoje průmyslu, která následuje po konceptu Průmyslu 4.0. Tento koncept byl poprvé představen Evropskou komisí v roce 2021 a přináší nové perspektivy a výzvy v oblasti průmyslového rozvoje a inovací [8].

Jedním z hlavních rysů Průmyslu 5.0 je zaměření na ještě větší integraci a kolaboraci mezi lidskými pracovníky a stroji. Namísto striktní automatizace a separace lidských pracovníků od strojů, jako tomu bylo u Průmyslu 4.0, se Průmysl 5.0 snaží o harmonický a synergický vztah mezi lidmi a stroji. Tímto způsobem lze dosáhnout vyšší flexibility a schopnosti adaptace průmyslových procesů na změny v prostředí a v poptávce [9].



Obr. 2.1: Referenční architektonický model pro průmysl 4.0.[1]

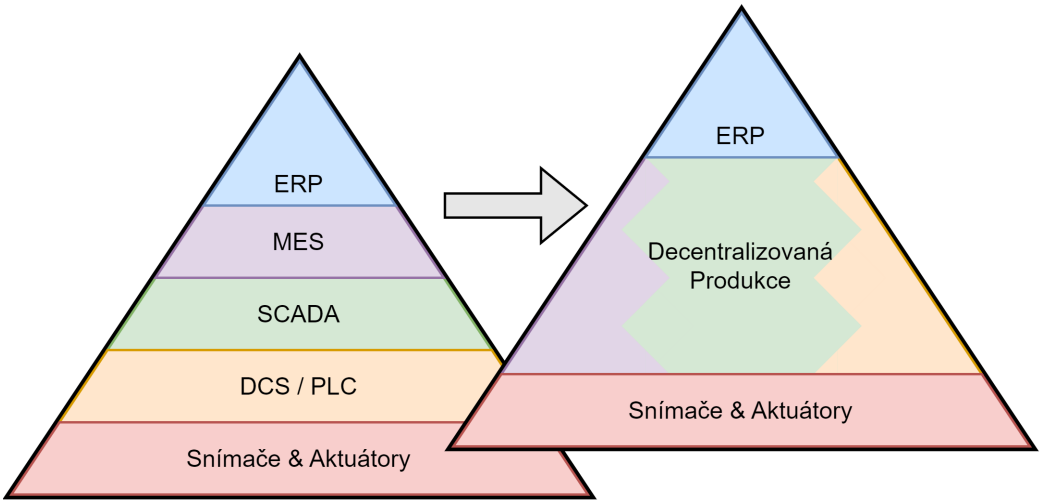
Je však důležité zdůraznit, že Průmysl 5.0 zatím zůstává spíše v rovině vize a teorie než v praktické aplikaci. V současné době není ještě plně definován a existuje spíše jako konceptuální rámec pro budoucí vývoj průmyslu. [10][8]

Vzhledem k tomu, že jeho implementace a konkrétní formy dosud nejsou zcela jasné, není vhodné v této fázi zahrnovat Průmysl 5.0 do souvislosti s Asset Administration Shell a Průmyslem 4.0. Takže i přesto, že Průmysl 5.0 přináší nové myšlenky a přístupy k průmyslovému rozvoji, autor této práce se v této fázi nezabývá jeho kompatibilitou s Asset Administration Shell. Místo toho se zaměřuje na koncept AAS a jeho význam v rámci stávajícího paradigmatu Průmyslu 4.0.

2.4 Decentralizovaná produkce

Decentralizovaná produkce představuje zásadní změnu v průmyslových procesech. Upravuje klasickou průmyslovou pyramidu, jak je nastíněno na obrázku 2.2. Samotná decentralizace výrobních procesů je spojena s nástupem Průmyslu 4.0 a CCPS (cyber-physical production systems), kde není potřeba nadřazený centrální kontrolní systém, který by řídil všechny prvky výrobního systému. Tato koncepce umožňuje rozdělení výrobních procesů na určité decentralizovaně řízené autonomní výrobní systémy a je podporována dvěma přístupy: holonické výrobní systémy (HMS) a multi-agentní systémy (MAS). Tyto přístupy se zaměřují na inteligenci v provádění

výrobních procesů, která je propojena s CPPS.[11][12][13]



Obr. 2.2: Decentralizovaná průmyslová pyramida.

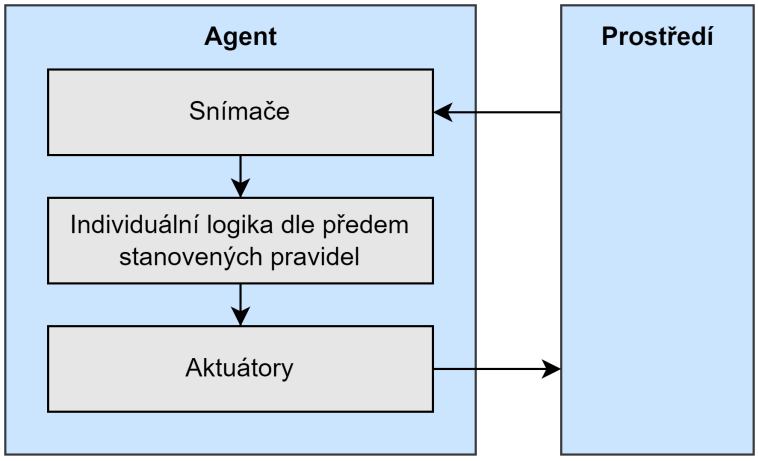
2.4.1 Holonické Výrobní Systémy (HMS)

Holonické výrobní systémy (HMS) vycházejí z konceptu holarchie, který umožňuje vytváření autonomních jednotek nazývaných holony. Tyto holony představují jak jednotlivé prvky výrobního systému, tak i celé subsystémy, které se mohou dynamicky organizovat a spolupracovat na řešení výrobních úkolů. Každý holon je schopen autonomního rozhodování a akce, ale zároveň je schopen komunikovat a spolupracovat s ostatními holony v systému. To umožňuje flexibilní a adaptabilní řízení výrobních procesů, protože holony mohou reagovat na změny a nepředvídané události bez nutnosti centrálního řízení. HMS tak podporují decentralizovanou kontrolu výroby a jsou klíčovým prvkem při implementaci konceptu Průmyslu 4.0, který zdůrazňuje využití digitálních technologií pro transformaci výrobních procesů.[14]

2.4.2 Multi Agentní Systémy (MAS)

Multi Agentní Systémy (MAS) jsou založeny na distribuované architektuře, ve které jsou jednotliví agenti schopni autonomního rozhodování a komunikace s ostatními agenty za účelem dosažení cílů výrobního procesu, jak je naznačeno v schématu na obrázku 2.3. Daný obrázek slouží k ilustraci principu jeho fungování. Každý agent v MAS je nezávislý a má své vlastní cíle, znalosti a dovednosti, což mu umožňuje efektivně reagovat na změny a podílet se na kooperativních úkolech s ostatními agenty. MAS jsou flexibilní a robustní, protože umožňují decentralizované rozhodování a distribuovanou kontrolu výrobních procesů. Navíc MAS podporují adaptabilitu a učení

se, protože agenti mohou v průběhu času zlepšovat své schopnosti a přizpůsobovat se novým podmínkám. Díky těmto vlastnostem jsou MAS vhodné pro aplikace v rámci Průmyslu 4.0, kde je důležité mít flexibilní a adaptabilní výrobní systémy schopné efektivně reagovat na změny v prostředí a požadavky zákazníků.[15][16]

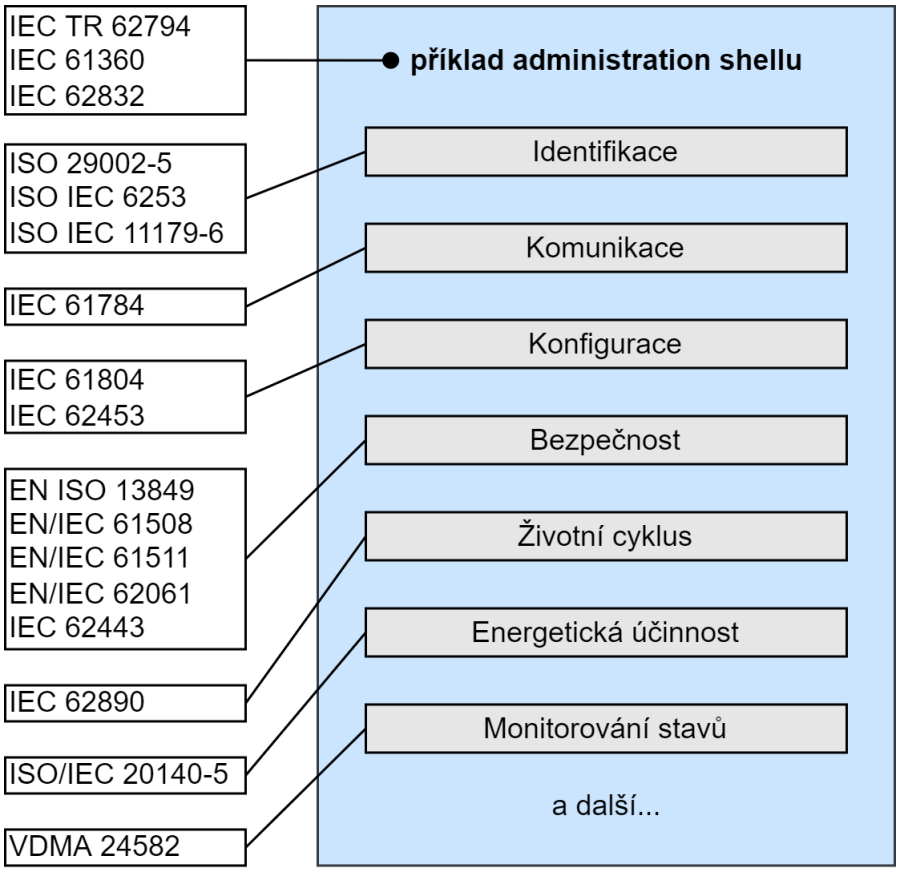


Obr. 2.3: Funkce agenta v multi agentním systému.

Oba přístupy, HMS i MAS, se zabývají inteligencí v provádění výrobních procesů a podporují decentralizovanou kontrolu výroby v souladu s konceptem Průmyslu 4.0. V rámci této práce však bude uplatněn přístup MAS, neboť je daleko vhodnější pro aplikace AAS. Tento přístup k decentralizované produkci klade důraz na samostatnou kontrolu každého prvku výrobního systému za účelem využití výhod propojení a lokálně dostupných informací.[13][14][15]

3 Co je AAS?

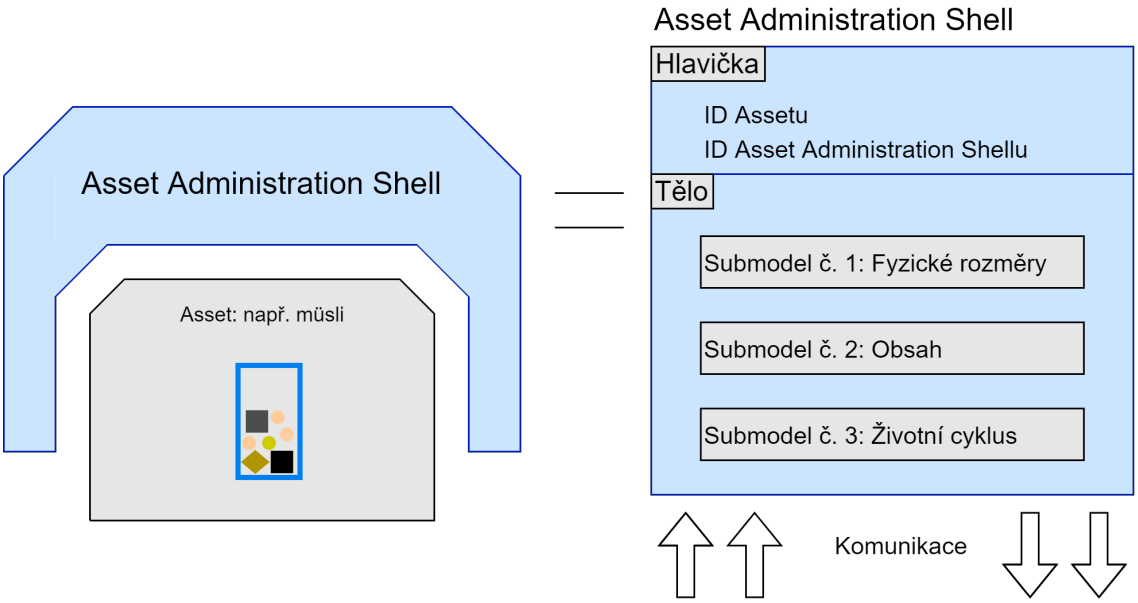
Asset Administration Shell je digitální reprezentace assetu. Jedná se o vylepšení digitálního dvojčete, kde AAS má navíc daleko rozsáhlejší možnosti komunikace. Skládá se z hlavičky a těla, jež je rozděleno na submodely, každý obsahující data pro danou skupinu[17]. Jeho základní schéma je k dispozici na obrázku 3.2. Lze ho taktéž rozdělit na jeho aktivní a pasivní část, kde aktivní část se zabývá komunikací a pasivní pak samotným uložením dat v příslušných submodelech. Jejich důkladnější rozbor je dále rozebrán v příslušných sekcích 3.1.1 Aktivní AAS a 3.1.2 Pasivní AAS. Taktéž je k dispozici příklad AAS skládajícího se ze submodelů a jeho příslušných standardů na obrázku 3.1.



Obr. 3.1: Příklad AAS skládajícího se ze submodelů a příslušných standardů.

Z důvodů zmíněných v sekci 2.1 Digitální dvojče (*Digital Twin*) je patrné, že tvorba detailního AAS je náročná. Z tohoto důvodu se i zde, stejně jako u digitálního dvojčete, uplatňuje princip, kde vytváříme AAS pouze v limitovaném měřítku a pouze pro ty části, u kterých nám jeho případné benefity převyšují pořizovací náklady. To jest důvod, proč je tato kapitola rozdělena do dvou částí. První se zabývá

AAS dle předních odborníků v oboru, další pak zkrácené formě AAS, která je daleko efektivnější za cenu menší univerzálnosti.



Obr. 3.2: Základní schéma AAS.[1]

3.1 Teoretické pojetí AAS

Aktuální informace ohledně AAS jsou dostupná především od dvou uskupení pocházejících z Německa. Tato uskupení jsou sponzorovaná tamní vládou. Prvním z nich je Platform Industrie 4.0 s reprezentanty asociací VDI, BITKOM, VDMA a ZVEI. Zabývá se tématy dle jejich 6 pracovních skupin:

- referenční architektura, standardizace a standardy;
- technologické a aplikační scénáře;
- zabezpečení síťových systémů;
- právní rámec;
- práce, vzdělávání a školení;
- digitální obchodní modely v průmyslu 4.0.

Především pak publikují články zabývající se radami, jak by se AAS mělo chovat, jak by k němu měli přistupovat firmy, jak má vůbec vypadat a podobně. Dále pak uváději samotný základ datamodelu AAS[18].

Druhou skupinou je pak Industrial Digital Twin Association (IDTA), která sdružuje 110 uskupení z 14 zemí[1]. Zde se zaměřují daleko více na praktickou stránku

[1]Údaj platný k 1.1.2024.

věci, především pak na uložení dat v submodelech a vzájemnou komunikaci AAS. Velké pozitivum je zde jejich open source GitHub na adrese `https://github.com/admin-shell-io`. Bohužel však v době psaní této práce není jejich řešení dostatečně rozsáhlé pro potřeby praktické implementace AAS s užitím PLC. Z tohoto důvodu se lze pouze inspirovat jejich základy a využít stejné programovací jazyky pro umožnění případné vzájemné kompatibility v budoucnu.

Nutno zmínit, že Platform Industrie 4.0 a IDTA spolu navzájem spolupracují a aktuální verze článků zabývající se AAS je záhodno kontrolovat u obou institucí. Jako příklad lze uvézt „Asset Administration Shell Reading Guide" z listopadu 2022[19], který zastává funkci jakéhosi rozcestníku pro studium AAS s odkazy na příslušné články. Jeho předchozí verze z ledna 2022 byla vydána v kooperaci obou institucí a verze z dubna 2021 pouze za Platform Industrie 4.0.[2]

Jejich představa AAS je dosti obšírná. Kupříkladu je schopna reagovat na požadavky z různých částí průmyslu. To jest jak z výrobní, tak i z obchodní či marketingové. Taktéž obsahuje jedinečné identifikátory jak samotných assetů, tak i AAS, a umožňuje hierarchickou strukturu submodelů a jejich filtraci. [20]

Mezi hlavní přínosy těchto uskupení je tvorba formátu souboru balíčku pro prostředí AAS v souladu s ISO/IEC 29500-2:2012 s příponou **.aasx**, obsahující data k AAS.[21] Tento typ souboru lze otevřít pomocí **AASX Package Explorer**, který je rovněž open source na adrese `https://github.com/eclipse-aaspe/aaspe`.[3] Umožňuje nahlížet a filtrovat dané submodely v grafickém prostředí. Aktuálně však pouze experimentálně jakožto demonstrace výhod AAS.

Veškerá literatura, jak již bylo nastíněno dříve, často nezmiňuje konkrétní praktické aplikace AAS v průmyslovém prostředí. Současný výzkum v této oblasti se primárně zaměřuje na AAS ve formátech JSON a XML, jak lze sledovat například v [22]. Tento přístup reflektuje snahu o standardizaci formátů pro popis AAS, což je klíčový krok pro dosažení interoperability a kompatibility mezi různými průmyslovými systémy a zařízeními. Nicméně, i přes tyto snahy, standardizace AAS zůstává stále v době psaní této diplomové práce v nedohlednu.

Navzdory tomu je důležité zdůraznit, že rozvoj a implementace standardů AAS může přinést významné výhody v oblasti průmyslového inženýrství jako je zlepšena interoperabilita, transparentnost dat a efektivnější správa průmyslových procesů. Je zřejmé, že i přes současnou absenci konkrétních standardů je důležité při vývoji a implementaci AAS brát v úvahu potenciální budoucí standardizační snahy. Z těchto důvodů je nutno postupovat obezřetně při její implementaci tak, aby byla případná budoucí standardizace snáze proveditelná.

[2] Většina autorů je totožná pro vícero verzí článků.

[3] V době psaní práce projekt migroval z `https://github.com/admin-shell-io/aasx-package-explorer`.

Samotná koncepce AAS se dá rozdělit do dvou následujících částí.

3.1.1 Aktivní AAS

Komunikace mezi jednotlivými AAS je řízena aktivní částí těchto entit. Tato aktivní část má za úkol udržovat a spravovat přesně definovaný jazyk AAS, jak je specifikováno v normě VDI 2193-1/2. Jde o komplexní slovník, který popisuje termíny a koncepty používané v rámci průmyslu 4.0. Tento slovník obsahuje definice a vzájemné vztahy mezi různými aspekty průmyslového prostředí, čímž je umožněna jednotná interpretace a porozumění informacím v rámci digitálních dvojčat a dalších průmyslových procesů.

Důležitou součástí aktivního AAS je i definice struktury zpráv a protokolu sémantické interakce. Tyto prvky umožňují efektivní výměnu informací mezi různými AAS a dalšími prvky průmyslového ekosystému. Struktura zpráv určuje formát a obsah dat, které jsou v rámci komunikace využívány, zatímco protokol sémantické interakce zajišťuje dodržování stanovených pravidel a sémantiky při výměně těchto dat.[23]

Zabezpečení aktivního AAS je další klíčovou součástí jeho fungování. Bezpečnostní aspekty jsou detailně popsány v článku nazvaném "Security der Verwaltungsschale"[24] (Bezpečnost správy administrativních skořepin), který je publikován v německém jazyce. Tento článek se zabývá různými aspekty bezpečnosti v kontextu AAS, včetně ochrany před neoprávněným přístupem, zajištění integrity a důvěrnosti dat a dalších bezpečnostních opatření nezbytných pro provoz a správu průmyslových systémů v rámci konceptu průmyslu 4.0.

3.1.2 Pasivní AAS

Pasivní AAS slouží ke správě informací o jednotlivých assetech v průmyslovém prostředí. Jeho hlavní funkcí je uchovávat a poskytovat detailní informace tak, aby bylo možné tyto informace efektivně využít v rámci průmyslových procesů a aplikací.

Konceptuálně je pasivní AAS rozděleno na dvě hlavní části: hlavičku a tělo, jak je patrné již ze zmíněného obrázku 3.2. Hlavička obsahuje klíčové metainformace o assetu jako je identifikátor, název, typ, umístění a další relevantní údaje. Tělo pak obsahuje samotná data o assetu jako jsou jeho vlastnosti, stav, historie, provozní parametry a další informace důležité pro jeho správu a monitorování.

Další důležitou charakteristikou pasivního AAS je jeho schopnost hierarchické organizace dat pomocí submodelů. Tyto submodely umožňují strukturované uspořádání informací o aktivu do logických skupin, což usnadňuje přehlednost a správu dat v rozsáhlejších průmyslových prostředích.

Pasivní AAS je konstruován s ohledem na efektivní výměnu informací s ostatními AAS a dalšími prvky průmyslového ekosystému. Tím výrazně přispívá k zajištění

interoperability a propojitelnosti průmyslových systémů, což je klíčový faktor pro dosažení plných výhod digitalizace a automatizace v rámci průmyslu 4.0.

3.2 AAS pro PLC

Z důvodů zmíněných v předešlé sekci je patrné, že implementace AAS do PLC je dosti problematická. Je tedy nutné představu AAS upravit tak, aby bylo možné danou implementaci provést. Tato sekce čerpá především z autorovy bakalářské práce[1], kde jsou zmíněny veškeré ústupky ve větším detailu.

V autorově bakalářské práci byl vybrán OPC UA protokol pro umožnění vzájemné komunikace mezi žadatelem služeb (produktem) a poskytovatelem služeb (PLC) ve formě OPC UA metod včetně vlastního standardu zasílání zpráv pro Vysoké učení technické v Brně. Veškeré komunikaci taktéž bylo umožněno, aby byla zašifrovaná v rámci požadavků na průmyslovou bezpečnost, na což dále naráží kapitola 7 Implementace žadatelů služeb.

Uložení dat ve formě submodelů bylo taktéž silně limitováno vlastnostmi PLC, kde není vhodné ukládat jinak nepotřebná data, či odkazy na ně. Výsledky práce byly velice uspokojivé, neboť bylo možné volat OPC UA metody a mít přístup ke sdíleným proměnným pomocí klienta ve formě aplikace **UAExpert**. Bohužel však stále nebylo možné realizovat dané řešení v praxi, neboť daná problematika vysoce převyšovala rozsah bakalářské práce.

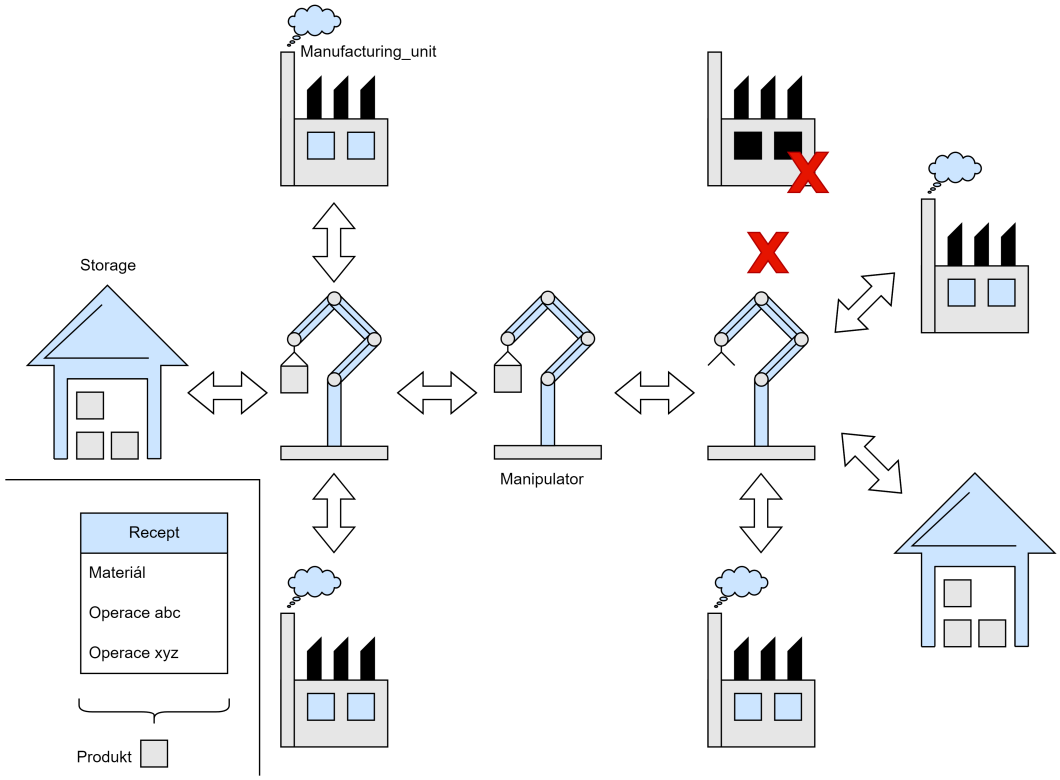
Princip inovativní implementace AAS v PLC, kterou navrhuje autor této práce si lze představit na ilustrativním obrázku 3.3. Jedná se o implementaci s decentralizovanou produkcí, kde dochází k porušení klasické průmyslové pyramidy (viz. obrázek 2.2 v předchozí kapitole). Rovněž jsou zde rozděleny implementace AAS dle toho, zda je daná entita žadatelem či poskytovatelem služeb. Všichni poskytovatelé služeb jsou zařízení, stroje ve výrobě, PLC s OPC UA servery a všechny produkty jsou ve formě OPC UA klientů, tedy žadatelů služeb s vlastní *„inteligencí"*[4]. Tito žadatelé služeb také disponují recepturou, kterou se snaží vykonat u jednotlivých poskytovatelů služeb.

Tento inovativní přístup přináší řadu výzev. Jednou z klíčových výzev je optimalizace alokace zdrojů a koordinace mezi různými entitami v systému, aby bylo dosaženo efektivního a spolehlivého výrobního procesu. To zahrnuje i adaptaci na dynamické změny v prostředí výroby[5] a schopnost rychle reagovat na nepředvídané

[4]Nejedná se o inteligenci v pravém slova smyslu, jakou by měla aplikace neuronové sítě, ale o seznam pravidel, který vnějšímu pozorovateli může připadat inteligentní.

[5]Především na to, zda je dané zařízení volné či obsazeno jiným produktem.

události, jako jsou poruchy zařízení nebo změny v požadavcích zákazníků[6]. Další výzvou je řízení komunikace a interakce mezi jednotlivými členy systému tak, aby bylo dosaženo optimálního výkonu a minimalizovány ztráty způsobené časovými prodlevami nebo konflikty využití zdrojů. Důležité je také zajistit, aby aplikace AAS nebyla příliš finančně nákladná. V neposlední řadě je nutné zajistit bezpečnost a spolehlivost systému, aby bylo minimalizováno riziko výrobních selhání a potenciálních hrozeb kybernetických útoků.



Obr. 3.3: Asset Administration Shell v produkci.

[6]Zde si lze představit přidání nové várky produktů, která má bližší termín výroby či větší profitabilitu za dřívější výrobu.

4 Plánování produkce

Jak již bylo nastíněno v sekci 3.2 AAS pro PLC, řešení Asset Administration Shell v PLC dle [1] není vhodné aplikovat do průmyslu z důvodu chybějící rezervační fronty a s ní spojené prioritizace výrobků. V této kapitole je uvedeno autorovo řešení spolu s odůvodněním příslušných kompromisů jak pro hardware a software, tak i pro samotné implementace Asset Administration Shell v průmyslu.

4.1 Požadavky AAS na buffer operací

S ohledem na zmíněné požadavky v sekci 3.1 Teoretické pojetí AAS je nutno vylepšit stávající systém rezervace strojů pro žadatele služeb. Není možné, aby daná zařízení nevykonávala žádnou činnost z důvodu časově neefektivního čekání na produkty. Je nutné přidat fronty, do kterých by se produkty průběžně zapisovaly a jejich akce by byly prováděny dle možností daných strojů. V rámci decentralizace je vhodné, aby tyto fronty a jejich průběžná správa byly zařazeny přímo do jednotlivých PLC tak, aby bylo co možná nejvíce umožněno tzv. Plug and Produce dle [25]. Pro umožnění prioritizace služeb s vyšším cenovým ohodnocením je nezbytné, aby dané fronty umožňovaly zařazování produktů dle jejich priorit. Tyto prioritní fronty musí být co do velikosti dosti obsáhlé, aby nedocházelo často k jejich plným stavům, což by mohlo mít za následek ve formě výrazného zhoršení optimalizace cenově a/nebo časově přívětivé produkce.

4.1.1 Chování prvků v PLC

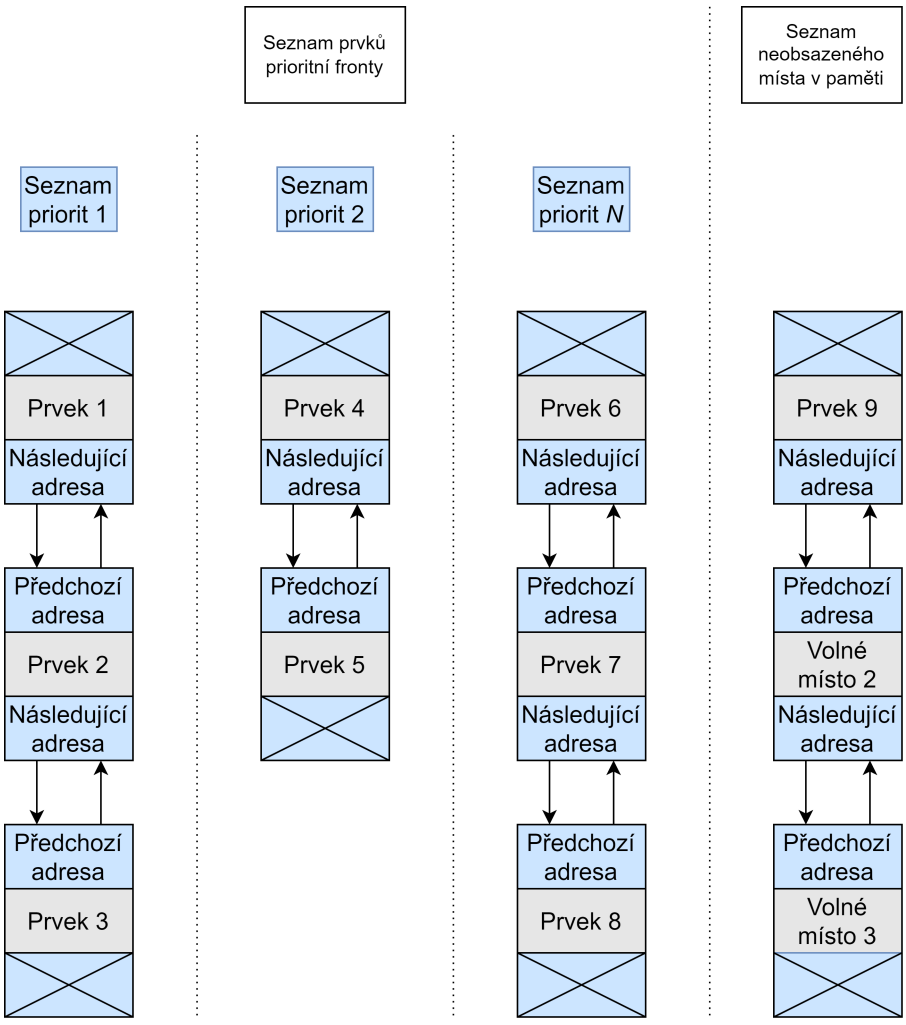
Prvek se nejdříve zaregistruje do prioritní fronty u příslušného PLC včetně parametrů jeho požadované operace. Až bude ve sféře vlivu daného stroje, tak se jeho akce vykoná dle jeho priority a následně bude vymazán z prioritní fronty. Z tohoto důvodu je nezbytné, aby prioritní fronty v PLC umožňovaly následující funkce:

- **get** - funkce vrátí prvek dle pozice;
- **push** - prvek se přidá do prioritní fronty dle své priority;
- **remove** - smazání prvku dle jeho ID;
- **find** - datová struktura navrátí pozici prvku dle jeho ID;
- **clear** - smazání všech prvků ve frontě.

4.1.2 Volba datové struktury

Zásadním problémem při snaze implementovat prioritní fronty do PLC je fakt, že PLC systémy v obecné rovině nepodporují dynamické alokování paměti. S přihlédnutím na tuto skutečnost jsou zde dvě možnosti implementace.

První je ve formě oboustranného spojového seznamu[1], kde každý prvek obsahuje adresu na prvek následující. Pro umožnění oboustranného vyhledávání se zde nachází i prvek předchozí. Avšak v rámci omezení PLC je adresa každého prvku pevně daná a při každém přidání nového prvku je provedeno vyhledání z neobsazených adres datového bloku ve smyslu, že se hledá nejbližší prázdný prvek. Tento problém by bylo možné obejít tím, že prázdné prvky budou ve vlastním spojovaném seznamu, aby se předešlo zdlouhavému hledání dalšího prázdného místa pro nově příchozí prvky. Z toho plyne, že po odstranění prvku z prioritní fronty by se jeho otisk na paměťovém místě vynuloval a tento prvek by se po odstranění z prioritní fronty přidal do seznamu neobsazeného místa v paměti, jak je znázorněno na obrázku 4.1. Taktéž by bylo nutné v rámci efektivnějšího vykonávání funkcí **get**, **remove** a **push**

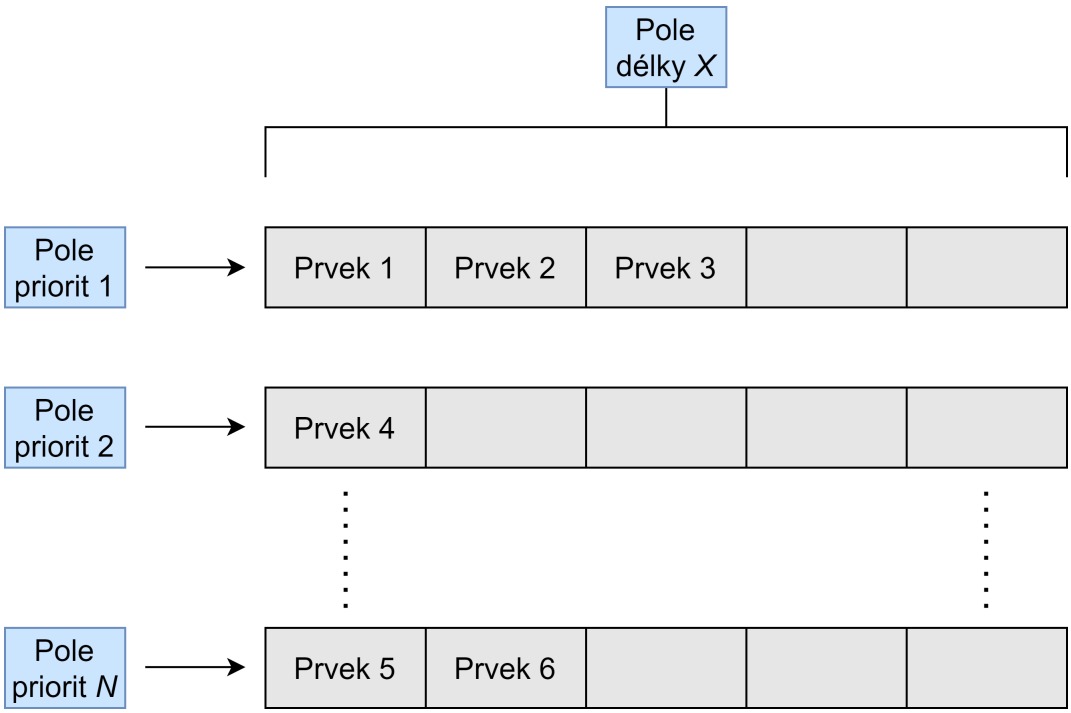


Obr. 4.1: Schéma oboustranného spojovaného seznamu bez dynamické alokace paměti.

[1]V angličtině známý pod označením 'double linked list'.

implementovat pro každou prioritu svůj vlastní oboustranný spojovaný seznam tak, aby se prohledávala pouze ta část seznamu, která obsahuje dané priority.

Druhou možností je fronta s poli priorit[2], kde je předdefinováno X polí o N délkách. Zde se do každého pole zapisují pouze prvky s příslušnou prioritou. Její funkční schéma je zobrazeno na obrázku 4.2.



Obr. 4.2: Schéma fronty s poli priorit.

V rámci paměťové efektivity umožňuje spojovaný seznam využití veškerého datového prostoru, avšak výraznou část zabírají adresace následujících a předchozích prvků. Na druhou stranu u fronty s polem priorit nelze datový prostor plně využít. Jestliže je totiž dílčí pole plné, tak neumožňuje zapsat další prvek s danou prioritou, i když ostatní pole nejsou plně obsazena. Přestože v tabulce 4.1 mají funkce v obou implementacích totožné výpočetní náročnosti v Bachmann - Landauově notaci[3], tak je výhodnější právě implementace pomocí polí, neboť je u nich předpoklad menší konstanty výpočetních náročností z důvodu eliminace přechodů dle adres, což umožňuje přímou indexaci prvků.

Z důvodů zmíněných v této subsekci bylo rozhodnuto, že implementace pomocí fronty s poli priorit je lepší jak po stránce implementační, tak i lehce po stránce časové náročnosti algoritmu. Jediná nevýhoda tohoto řešení je v problému při obsazenosti pole s danou prioritou. Avšak řešení tohoto nedostatku lze v podstatné míře

[2]V angličtině známá pod označením 'Bucket queue'.
[3]Též známá pod označením *Big O* či *omikron* notace.

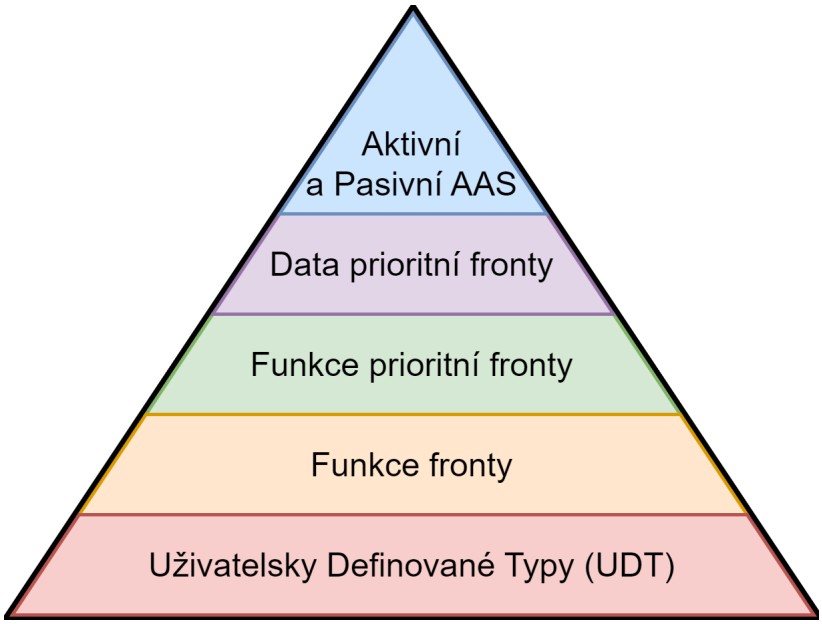
přesunout na stranu žadatele služeb, kde by se produkt musel zaregistrovat s odlišnou prioritou. Tato možnost je preferována, neboť žadatelé služeb nejsou limitováni časovými ani paměťovými omezeními v takové míře jako PLC systémy provádějící výrobu.

Tab. 4.1: Časové náročnosti funkcí oboustranného spojovaného seznamu a polí pro prioritní fronty ve stálých strukturách.

Funkce	**Časová náročnost**
get	$\mathcal{O}(n_{priority})$
push	$\mathcal{O}(1)$
remove	$\mathcal{O}(n_{priority})$
find	$\mathcal{O}(n_{priority})$
clear	$\mathcal{O}(1)$

4.2 Aplikace prioritních front do PLC

Z důvodů zmíněných výše se tato práce zabývá aplikací ve formě fronty s poli priorit. Tato sekce objasňuje její zdárnou implementací do PLC, jejíž hierarchickou strukturu si lze představit na obrázku 4.3.



Obr. 4.3: Schéma hierarchické struktury prioritních front v PLC.

Nejdříve je pro správné pochopení zmíněné implementace nutné uvést využité datové typy využívané na vyšších úrovních, což jsou: **typeProductInQueue**, který je popsán v tabulce 4.2, dále **typeArrayOfProducts**, jež je popsán v tabulce 4.3 a **typePQueueData**, který je popsaný v tabulce 4.4.

Tab. 4.2: Popis UDT typeProductInQueue týkající se prioritních front.

typeProductInQueue		
Název	**Datový typ**	**Jeho význam**
id	Dint	ID žadatele služeb.
material	Int	Druh materíalu žadatele služeb.
operation	Int	Požadovaná operace.
op. parameters	Wstring	Parametry operace odděleny znakem **-**.

Tab. 4.3: Popis UDT typeArrayOfProducts týkající se prioritních front.

typeArrayOfProducts		
Název	**Datový typ**	**Jeho význam**
product	Array[0.."SubQueueCountSmall"] of "typeProductInQueue"	Pole produktů ve frontě
staticQueueVariables	Struct	Struktura proměnných pro udržování fronty.
- statFirstItemIndex	Int	Index nejstaršího zápisu v bufferu.
- statNextEmptyItemIndex	Int	Index následujícího volného místa pro zápis v bufferu.
- statElementCount	DInt	Počet prvků v bufferu

Na další úrovni jsou pak funkce klasické fronty, které jsou uvedeny v tabulce 4.5. Tyto základní funkce klasických front jsou dále využívány na úrovni funkcí prioritních front, které jsou následně uvedeny v tabulce 4.6. Toto je umožněno díky přístupu k datům prioritní fronty.

Veškerá data prioritní fronty jsou uložena v instanci **typePQueueData** (tj. PQueueData), jež je uvnitř datového bloku **Queues**. Zde je rovněž uložena instance **typeArrayOfProducts** (tj.ReportedProductsQueue), která obsahuje infor-

Tab. 4.4: Popis UDT typePQueueData týkající se prioritních front.

typePQueueData		
Název	**Datový typ**	**Jeho význam**
buffer	Array[0.."QueueCountSmall"] of "typeArrayOfProducts"	Listy všech dílčích front v prioritní frontě.
isEmptyBuffer	Array[0.."QueueCountSmall"] of Bool	Indikace plných front.
isFullBuffer	Array[0.."QueueCountSmall"] of Bool	Indikace prázdných front.
queueProductCount	Dint	Celkový počet prvků v prioritní frontě.
emptyItem	"typeProductInQueue"	Konstanta, jak má vypadat prázdný prvek.
activeItemPosition	DInt	Pozice aktivního prvku v produkci.

mace o prvcích v jeho sféře vlivu. Na ni jsou však volány pouze funkce klasické fronty z očividných důvodů.

Na vrcholu pyramidy je pak Aktivní a Pasivní část AAS. Toto znázorňuje fakt, že jak OPC UA metody, tak sdílené proměnné mají přímý přístup k datům prioritní fronty.

Tab. 4.5: Seznam funkcí klasické fronty v PLC.

Název funkce	**Význam funkce**
Queue_clear	Funkce vymaže z fronty veškeré jeho prvky.
Queue_find	Funkce nalezne ve frontě prvek dle jeho ID.
Queue_get	Funkce vrátí prvek ve frontě dle jeho pozice.
Queue_push	Funkce přidá do fronty prvek.
Queue_remove	Funkce smaže z fronty prvek dle jeho pozice.

Tab. 4.6: Seznam funkcí prioritní fronty v PLC.

Název funkce	**Význam funkce**
PQueue_clearAll	Funkce vymaže z prioritní fronty veškeré jeho prvky.
PQueue_clearPriority	Funkce vymaže z prioritní fronty veškeré jeho prvky s danou prioritou.
PQueue_find	Funkce nalezne v prioritní frontě prvek dle jeho ID.
PQueue_get	Funkce vrátí prvek v prioritní frontě dle jeho pozice.
PQueue_push	Funkce přidá do prioritní fronty prvek dle jeho priority.
PQueue_removeById	Funkce smaže z prioritní fronty prvek dle jeho ID.
PQueue_removeByPosition	Funkce smaže z fronty prvek dle jeho pozice.

5 AAS pro PLC ve formě OPC UA metod

Jak již bylo zmíněno v předešlých sekcích, tato práce navazuje na bakalářskou práci[1], v níž byl expertně vytvořen návrh OPC UA prostředí a s ním související OPC UA metody v PLC spolu s pasivními daty. Tato kapitola se zabývá aktualizovanými OPC UA metodami a pasivními daty spolu s odůvodněním jejich změn, především však s ohledem na aplikace prioritních front.

5.1 Volba PLC

V rámci návaznosti na předešlou práci se bude i nadále pracovat s PLC od firmy Siemens. Avšak v této práci se oproti té předešlé bude pracovat s vyšší řadou S7-1500, a to ze dvou podstatných důvodů.

Prvním z nich je, že aplikace prioritních front do PLC vyžaduje o poznání vyšší náročnost na paměťová úložiště v programu. Aplikace prioritních front do PLC jsou ovlivněny požadavky na integrovanou pracovní paměť pro data. Z tohoto důvodu je k dipozici tabulka 5.1 s přehledem jejich maximálních velikostí dle různých modelů CPU. Jedná se pouze o orientační tabulku pro čtenáře, neboť každý model CPU může mít několik typů s odlišnou pracovní pamětí pro data, a tudíž se pro budoucí typy jmenovaných modelů mohou lišit.

Tab. 5.1: Tabulka limitací pracovních pamětí u standardních Siemens PLC řady S7-1500.

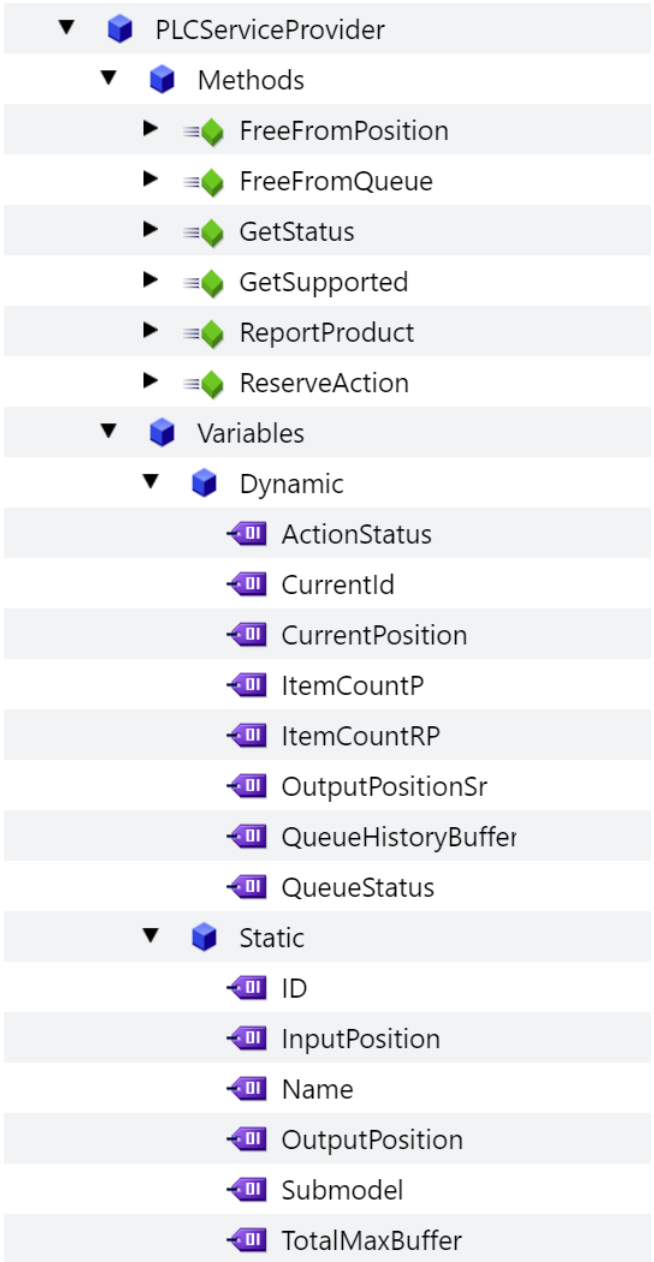
CPU Model	**Integrovaná pracovní paměť pro data [Mb]**[1]
1511-1 PN	1-1,5
1513-1 PN	1,5-2,5
CPU 1515-2 PN	3-4,5
1516-3 PN/DP	5-7,5
1517-3 PN/DP	8
CPU 1518-4 PN/DP	60

Druhým důvodem je umožnění simulovat PLC pomocí programu SIMATIC S7-PLCSIM Advanced V4.0 od firmy Siemens pro řady S7-1500. V rámci praktické implementace totiž bude nutné vytvořit rozsáhlou síť PLC tak, aby bylo možné otestovat výsledné řešení této práce, což by bylo dosti komplikované vzhledem k požadavkům na hardware.

[1]Platné k 31.12.2023.

5.2 Rozlišení jednotlivých částí AAS v PLC

Každé AAS v PLC obsahuje 6 metod, které zajišťují jeho aktivní část, a dále 8 dynamických a 6 statických[2] proměnných jež zajišťují pasivní část AAS. Jejich komunikační prostředí v OPC UA je znázorněno na obrázku 5.1 v prostředí SiOME.



Obr. 5.1: OPC UA prostředí pro PLC.

[2]Přestože jsou proměnné nazvány jakožto statické, tak jsou nativně volány klientem stejně jako dynamické proměnné. Případnými optimalizacemi v této problematice se autor práce nezabývá.

5.2.1 Rozbor OPC UA metod

Všechny metody[3] mají jeden vstupní a jeden výstupní string[4] pro zajištění větší univerzálnosti, kde jsou jednotlivá slova oddělena znakem /. Pasivní data jsou taktéž ve formě stringů. Dané metody jsou:

- **FreeFromPosition**
  Metoda umožňuje žadateli služeb upozornit zařízení, že se již nenachází v jeho sféře vlivu a tím pádem neobsazuje jeho výstupní pozice, což zařízení umožní tuto pozici zaplnit jiným produktem. Má pouze jeden vstupní parametr, a to **ID žadatele služeb**. Výstup je ve formě **Success** v případě úspěšného ohlášení či příslušný kód chyby. Vybrané kódy chyb jsou v tabulce 5.2.
- **FreeFromQueue**
  Metoda umožňuje žadateli služeb uvolnit se z dané fronty. Má pouze jeden vstupní parametr, a to **ID žadatele služeb**. Výstup je ve formě **Success** v případě úspěšného odregistrování. V opačném případě je zde příslušný kód chyby.
- **GetStatus**
  Metoda má jeden vstupní parametr, a to **ID žadatele služeb**. Její odpověď je ve formě **position:Číselné pozice prvku ve frontě**, případně příslušný kód chyby. Žadatel služby si tak může kdykoliv zjistit, kolik prvků je aktuálně před ním, případně zdali je vůbec zařazen v dané frontě.
- **GetSupported**
  Slouží k evaluaci nabízených služeb. Oproti stejnojmenné metodě v autorově bakalářské práci obsahuje větší škálu parametrů. Její vstupní parametr je ve tvaru **ID žadatele služeb/ID materiálu/ID operace/parametry operace odděleny znakem pomlčky, tj. -**. Výstup této metody je ve tvaru **support:hodnota1_position:hodnota2**, kde:
  - **hodnota1**:
    Jedná se o bezrozměrné číslo v rozmezí od 0 do 100, které vyjadřuje míru podpory dané operace zařízením. Hodnota 0 znamená, že zařízení danou operaci nepodporuje, zatímco hodnota 100 značí, že ji podporuje maximální možnou mírou. Toto číslo umožňuje produktu vyhodnotit úroveň podpory požadované operace. Poskytovatel služby pak tuto úroveň reflektuje v čase, který je potřeba k provedení dané služby. Vyšší hodnota znamená kratší dobu trvání služby.
  - **hodnota2**:

[3] Nejedná se o parametry funkčních bloků metod v PLC, nýbrž o parametry OPC UA metod.

[4] V PLC je nutno v rámci úspěšného napojení na Server Interface pracovat s datovým typem **WSTRING**.

Jedná se o pozici v prioritní frontě poskytovatele služeb, kde by se produkt ocitl, zaregistroval by se u daného zařízení. Slouží k tomu, aby si žadatel služeb mohl vybírat poskytovatele služeb nejen na základě podpory daných operací, ale i na zatížení daných strojů.

Toto rozdělení na podporu a zatížení zařízení umožní daleko lepší rozhodování pro žadatele služeb v porovnání s modelem navrženým v autorově bakalářské práci [1].

- **ReportProduct**

  Je nutné, aby PLC vědělo, které prvky jsou v jeho sféře vlivu. První z možností je ve formě čárových kódů, které by měl daný prvek na sobě a každý stroj by tím pádem musel obsahovat zařízení umožňující jeho čtení, což by mohlo zvyšovat další finanční náklady u případné praktické implementace AAS do PLC. Taktéž vzhledem k situaci, kdy výsledky této práce budou testovány na virtuálních PLC, by vznikal problém v chybějící simulaci samotných zařízení, které by poskytovatelům služeb hlásily jejich pozice.

  Kvůli jmenovaných důvodů bylo uplatněno druhé řešení, což je implementace této metody, která je volána žadateli služeb pro ohlášení se, že se nachází v jejich sférách vlivu. Její vstupní argument je ve formě **ID žadatele služeb** a má odpověď ve tvaru **Success** indikující úspěšné zapsání do seznamu prvků, které jsou v jeho sféře vlivu, nebo příslušný kód chyby.

  Z praktických implikací této práce vyplývá, že není nutné implementovat metodu, která by odregistrovávala produkty ze seznamu prvků, pokud budou splněny následující podmínky:

  1. Prvky budou odregistrovány automaticky poskytovatelem služeb po vykonání příslušných operací.
  2. Žadatelé služeb budou korektně volat tuto metodu pouze v případech, kdy budou skutečně v daných sférách vlivů.
  3. Případy, kdy je prvek ve sférách vlivů dvou různých zařízení, je nutno pečlivě otestovat tak, aby bylo zajištěno korektní volání této metody od žadatelů služeb.

- **ReserveAction**

  Tato metoda je sloučením metod **Give**, **Change** a **Transport** v autorově bakalářské práci. Toto sloučení proběhlo s cílem zefektivnit zavádění AAS do PLC díky standardizovanému OPC UA prostředí platnému pro všechny typy poskytovatelů služeb. Vstupní parametr je s tímto totožný v metodě **GetSupported** tj. **ID žadatele služeb/ID materiálu/ID operace/parametry operace odděleny znakem pomlčky, tj. -**. Výstupní parametr je ve formě stringu **Success** či **Failed:error code**, kde **error code** je kód chyby v hexadecimální soustavě, jak tomu je i u ostatních metod.

Až na metody **ReportProduct** a **FreeFromPosition** pracují všechny metody s prioritní frontou.[5] V praktické implementaci se může stát, že vlivem různých okolností nebude prioritní fronta korektně fungovat. Z tohoto důvodu metody **GetPosition**, **ReserveAction**, **Free** a **ReportProduct** mohou vrátit zprávu ve tvaru **Error:error code** či **Error1:error code1;Error2:error code2**, kde **error code** označuje kód chyby v hexadecimální soustavě o délce 4 znaků identifikující problémy s prioritní frontou. Její hodnoty byly voleny tak, aby odpovídaly ustáleným zvyklostem v chybových kódech firmy Siemens v PLC. Podstatné chyby jsou k dispozici v tabulce 5.2, která je vytažena z části **PQueue Tag table** nacházející se v příloze obsahující program v prostředí TIA portal v17.

5.2.2 Implementace aktivní části AAS do PLC

Aktivní část AAS je implementována ve formě OPC UA metod. Toto je zajištěno pomocí klasických funkčních bloků, které však obsahují pouze statické a dočasné proměnné. Jsou zde využity: knihovní funkce **OPC_UA_ServerMethodPre**, **OPC_UA_ServerMethodPost**, struktury **UAMethod_In-Parameters**, **UAMethod_Out-Parameters**, uživatelsky definované proměnné **typeOPCUA_Status**, **typeOPCUA_MethodHandling** a **typeArrayMessage**. Jejich podrobnější popis je v návodu [26](eng) a/nebo především v autorově bakalářské práci [1](cz). Tyto funkční bloky jsou obsaženy v bloku **Active AAS**, který je cyklicky volán dle standardních procedur.

5.2.3 Implementace pasivní části AAS do PLC

Pasivní část AAS, starající se o jeho řádný interní chod, je zajištěna funkčním blokem **Passive AAS**. Jednak jsou zde uvedeny statické proměnné **ID**, **InputPosition**, **Name**, **OutputPosition**, **ID**, **Submodel**, **Submodel** a **TotalMaxBuffer**, kde popis jejich funkcí je k dispozici v tabulce 5.4. Jednak se stará o pravidelnou aktualizaci dynamických proměnných **ActionStatus**, **CurrentId**, **CurrentPosition**, **ItemCountP**,**ItemCountRP**,**OutputPositionSr**, **QueueHistoryBuffer** a **QueueStatus**. Jejich popis je v tabulce 5.3.

Pracuje tím způsobem, že periodicky volá funkční blok **CheckAction** kontrolující průniky prvků, které jsou jak v samotné prioritní frontě, tak i v seznamu prvků jeho sféry vlivu **RPQueue**[6].

[5] Zmíněné metody pracují pouze s klasickou frontou.

[6] **RPQueue** značí Reported Products Queue, neboli fronta ohlášených prvků, aktualizována AAS metodou **ReportProduct**.

Tab. 5.2: Přehled chybových kódů v PLC ohledně prioritních front.

Jméno v programu	**Hex. hodnota**	**Význam chyby**
PQErrBufferIsEmpty	8001	Buffer je prázdný.
PQErrBufferIsFull	8002	Buffer je plný.
PQErrWrongTypeItem	8201	Datový typ parametru InOut ‘item‘ neodpovídá typu prvků pole vstupu ‘buffer‘.
PQErrWrongTypeInitialItem	8202	Datový typ vstupu ‘initialValue‘ neodpovídá typu parametru InOut ‘item‘.
PQErrItemNotFound	8301	Produkt nebyl nalezen v PQueue_find.
PQErrItemAlreadyInProgress	8302	Produkt nelze odebrat z fronty PQueue, protože je již ve výrobě.
PQErrItemAlreadyInQueue	8401	Produkt se již nachází v prioritní frontě poskytovatele služeb.
RQErrIdAlreadyKnown	8501	Produkt je již známý ve sféře vlivu poskytovatele služeb.
PQErrIndexInArrayLimits1	8601	Tag ‘statNextEmptyItemIndex‘ není v mezích pole.
PQErrIndexInArrayLimits2	8602	Tag ‘statFirstItemIndex‘ není v mezích pole.
PQErrClearBuffer	8610	Chyba při čištění bufferu v bloku ‘MOVE_BLK_VARIANT‘ - zkontroluj kód ‘subFunctionStatus‘.
PQErrReturnFirstEntry	8611	Chyba při vrácení první položky z bufferu v bloku ‘MOVE_BLK_VARIANT‘ - zkontroluj kód ‘subFunctionStatus‘.
PQErrReplaceItemByInitValue	8612	Chyba při nahrazení položky počáteční hodnotou v bloku ‘MOVE_BLK_VARIANT‘ - zkontroluj kód ‘subFunctionStatus‘.
PQErrWriteEntry	8613	Chyba při zápisu položky do bufferu v bloku ‘MOVE_BLK_VARIANT‘ - zkontroluj kód ‘subFunctionStatus‘.
PQErrMissingProduct	8701	Produkt nebyl nalezen ve frontě.

Tab. 5.3: Popis funkcí dynamických proměnných v implementovaném OPC UA modelu komunikace.

Název proměnné	**Význam proměnné**
ActionStatus	Popisuje stav zařízení. Může nabývat hodnot 'idle', 'inProgress' a 'failed'.
CurrentId	Jedná se o celočíselnou hodnotu ID žadatele služeb, který je aktuálně v procesu 'inProgress'.
CurrentPosition	Jedná se o celočíselnou hodnotu udávající pozici aktuálního produktu v prioritní frontě.
ItemCountP	Počet prvků v prioritní frontě.
ItemCountRP	Počet prvků ve fromtě **RPQueue**, tj. počet prvků ve sféře vlivu poskytovatele služeb.
OutputPositionSr	Celočíselná hodnota ID žadatele služeb, který aktuálně zabírá výstupní pozici poskytovatele služeb.
QueueHistoryBuffer	String ve formátu **'x/x/x'**, kde '**x**' jsou celočíselné ID žadatelů služeb, u kterých již byly úspěšně dokončeny požadované služby. Jedná se o plovoucí buffer, kde nalevo jsou nejstarší a napravo nejnovější prvky.
QueueStatus	Jedná se o string ve formátu '**x/x/x/.../x**' o počtu proměnných '**x**' rovnu počtu tzv. SubFront, tedy počtu možných priorit produktů. '**x**' je počet žadatelů služeb v dané SubFrontě, kde nalevo jsou SubFronty s nejvyšší prioritou a napravo nejmenší. Měli bychom kupříkladu 5 možných priorit, kde by bylo **17** prvků s nejvyšší prioritou **0** a **42** prvků s prioritou **4**, vypadala by tato proměnná **17/0/0/42/0**.

Tab. 5.4: Popis funkcí statických proměnných v implementovaném OPC UA modelu komunikace.

Název proměnné	**Význam proměnné**
ID	Proměnná obsahující celočíselné ID poskytovatele služeb.
InputPosition	Celočíselé ID vstupní pozice poskytovatele služeb. Slouží k tomu, aby žadatel služeb věděl kam má jít v případě, že se rozhodne zaregistrovat služby u daného poskytovatele služeb.
Name	Název poskytovatele služeb, např. sto2, man4, apd.
OutputPosition	Celočíselé ID výstupní pozice poskytovatele služeb. Slouží k tomu, aby žadatel služeb znal svou pozici v případě úspěšného splnění akce u daného poskytovatele služeb. Může pak oznámit manipulátoru pozici, z které ho má vyzvednout.
Submodel	Jedná se o název submodelu. Může nabývat hodnot **Storage**, **Manipulator** nebo **Manufacturing_unit**.
TotalMaxBuffer	Maximální možný počet produktů v prioritní frontě poskytovatele služeb. Jedná se tedy o součet všech subfront.

6 Informační a datový model komunikačního rozhraní

Informační model AAS specifikuje, jak jsou data a informace strukturovány v rámci komunikace mezi jednotlivými průmyslovými zařízeními a systémy. Obsahuje definice klíčových entit, jako jsou aktiva, vlastnosti, relace a metadata, a popisuje jejich vzájemné vztahy a hierarchickou organizaci. Tento model poskytuje rámec pro jednotnou interpretaci a výměnu dat mezi různými průmyslovými subjekty a umožňuje efektivní správu a manipulaci s informacemi o assetech.

Datový model komunikačního rozhraní AAS určuje formát a strukturu dat používaných při komunikaci mezi AAS a jinými průmyslovými zařízeními a systémy. Zahrnuje specifikace pro různé typy zpráv, protokoly komunikace a formáty dat, které jsou využívány při výměně informací. Tento model definuje standardizované postupy pro přenos dat, čímž pak zajišťuje interoperabilitu mezi různými průmyslovými platformami a technologiemi.

Při definování informačního a datového modelu komunikačního rozhraní AAS je klíčové zohlednit potřeby a požadavky průmyslových uživatelů a zajistit kompatibilitu s existujícími průmyslovými standardy a protokoly. Zároveň je nutné dbát na flexibilitu a rozšiřitelnost modelu, aby bylo možné efektivně reagovat na budoucí potřeby a nové technologické trendy v oblasti průmyslové automatizace a digitalizace, jak bylo zmíněno v sekci 3.1 Teoretické pojetí AAS.

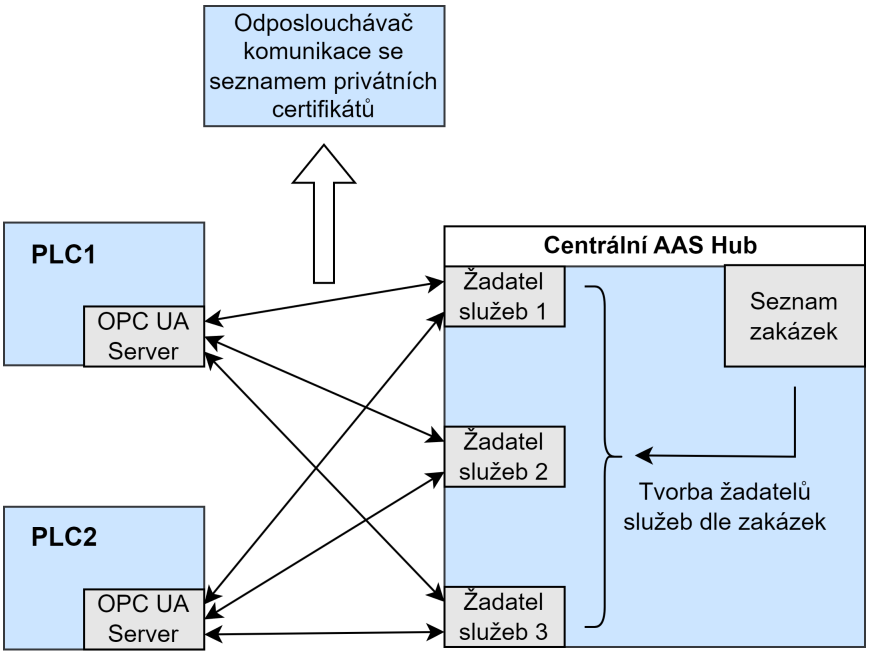
6.1 Volba vhodného modelu komunikačního rozhraní

Je nezbytné zajistit sledování výrobních procesů a rychlou reakci na případné události v průmyslovém prostředí. S tímto cílem autor práce navrhuje tři možná řešení, jež budou podrobena dalšímu vyhodnocení a případnému testování. Z těchto návrhů bude následně vybráno to nejvhodnější, které se bude jevit jako nejuspokojivější a nejefektivnější pro navazující výzkum v této problematice.

Prvním navrhovaným řešením je vytvoření tzv. Odposlouchávače, který bude mít přístup k certifikátům PLC[1] a centrálního AAS hubu. Toto řešení se vyznačuje svou finanční efektivitou, avšak současně přináší značná bezpečnostní rizika spojená se sdílením privátních certifikátů. Schéma tohoto řešení je zobrazeno na obrázku 6.1.

Druhou možností je vytvoření pozorovacího hubu, který bude sloužit k centralizovanému sběru informací o akcích zařízení. Tato strategie by umožnila sledovat

[1]Přestože se neočekává zabezpečená komunikace při následném testování z důvodu její komplikované proveditelnosti, je nutno dbát na umožnění zabezpečení komunikace a s tím související argument při výběru komunikačního rozhraní.



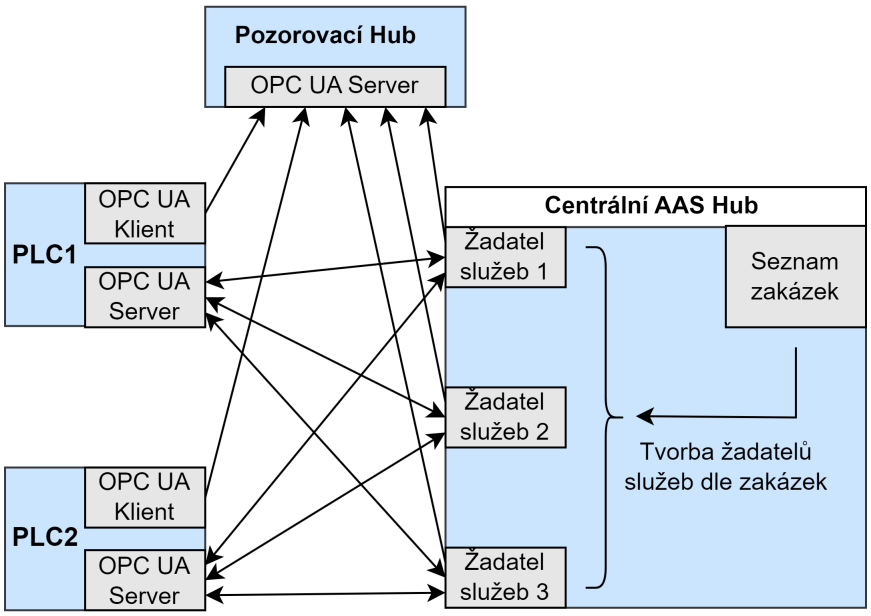
Obr. 6.1: Model komunikačního rozhraní s odposlouchávačem.

výrobní procesy a reagovat na události z jednoho centrálního místa. Avšak, aby bylo možné tento hub implementovat, vyžadovalo by to, aby PLC servery disponovaly také OPC UA klienty. To znamená, že by bylo nutné provést další investice do hardwaru a softwaru. Tato konfigurace by také prakticky znásobila komunikaci na síti. Tento nárůst komunikace může[2] potenciálně vést k problémům s přetížením a nedostatečnou propustností sítě, což by mohlo ovlivnit stabilitu celého systému. Přestože by tento přístup umožnil detailní monitorování a řízení výrobních procesů, je důležité zvážit jeho významné náklady a potenciální dopady na výkon a spolehlivost průmyslové sítě. Jeho schéma je k dispozici na obrázku 6.2.

Poslední možností, kterou autor navrhuje, je přidání úlohy centrálnímu AAS hubu, který bude zaznamenávat potřebné informace z výrobního prostředí a následně je předávat operátorům ve výrobě. Tímto způsobem se umožní monitorování výrobních procesů a rychlá reakce na případné události, přičemž se minimalizuje zátěž na průmyslové síti. Obrázek 6.3 ilustruje schéma tohoto řešení.

Implementace této strategie by umožnila operátorům v reálném čase sledovat výrobní procesy a přijímat informovaná rozhodnutí na základě aktuálních dat. Díky centralizovanému sběru informací by se také zlepšila transparentnost a efektivita výrobních operací. Navíc by tato konfigurace minimalizovala bezpečnostní rizika spojená se sdílením privátních certifikátů, jak tomu je u prvního navrhovaného řešení.

[2] Autor práce si je vědom nízkého rizika vzhledem k velkým propustnostem dnešních průmyslových sítí. Přesto však zmiňovaná možnost může přinést v tomto ohledu nejasné problémy.



Obr. 6.2: Model komunikačního rozhraní s pozorovacím Hubem.

Z těchto důvodů byla třetí varianta vybrána jako nejvhodnější. Bude tedy implementována, přestože zavedením centrálního prvku se mírně vyruší nativní myšlenka decentralizace, neboť se zde opět vytváří centrální prvek. Změna spočívá v tom, že těchto AAS hubů může[3] být více, čímž se opět zmiňovaná implementace posouvá k původním úvahám o decentralizované výrobě.

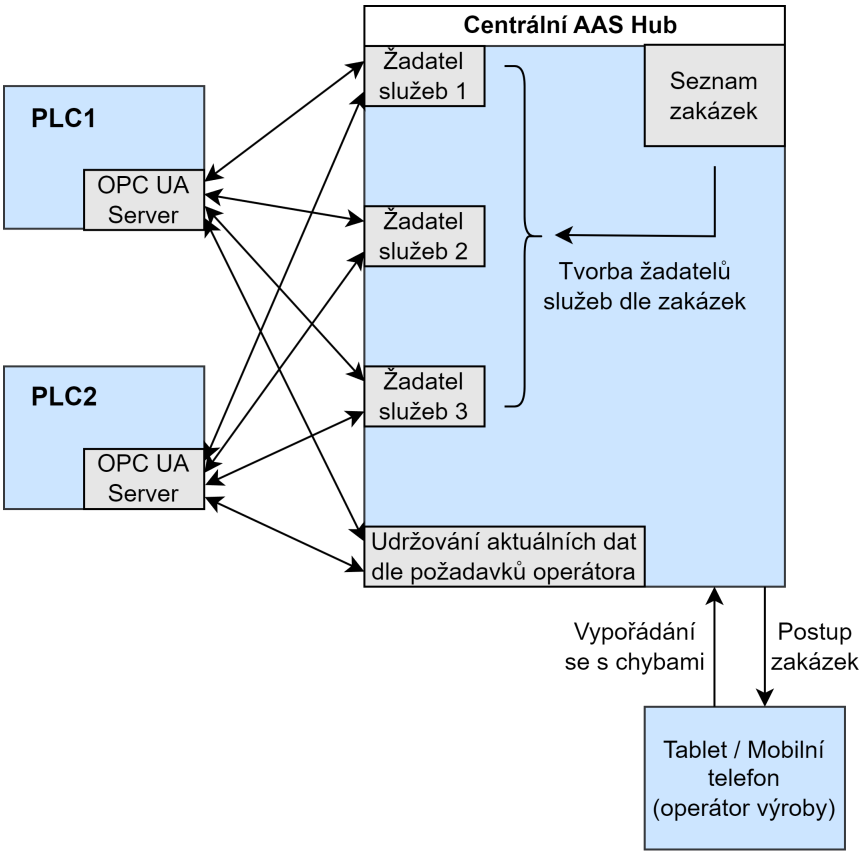
6.2 Formát komunikace

Tato sekce se zabývá informačním pohledem na samotný formát komunikace, především pak jejím sekvenčním pohledem. Popis navrhované komunikace je zde prezentován z informativních důvodů na UML diagramu.

6.2.1 Volba univerzálního datového typu

Základní popis sdílených proměnných a metod v rámci OPC UA komunikace byl již popsán v sekci 5.2 Rozlišení jednotlivých částí AAS v PLC. Je však nutné dále zdůvodnit volbu univerzálního datového typu ve formě **string** pro veškeré sdílené proměnné v OPC UA prostředí. Toto rozhodnutí je ovlivněno již zmiňovaným nedostatkem výzkumu v problematice aplikací AAS ve výrobě, kde jak již bylo nastíněno, aplikovat AAS do výroby ještě nikdo nikdy nepokoušel, a tedy ani není z čeho vycházet, pokud jde o standardy.

[3] V reálné výrobě se pak z důvodu redundance toto přímo doporučuje.



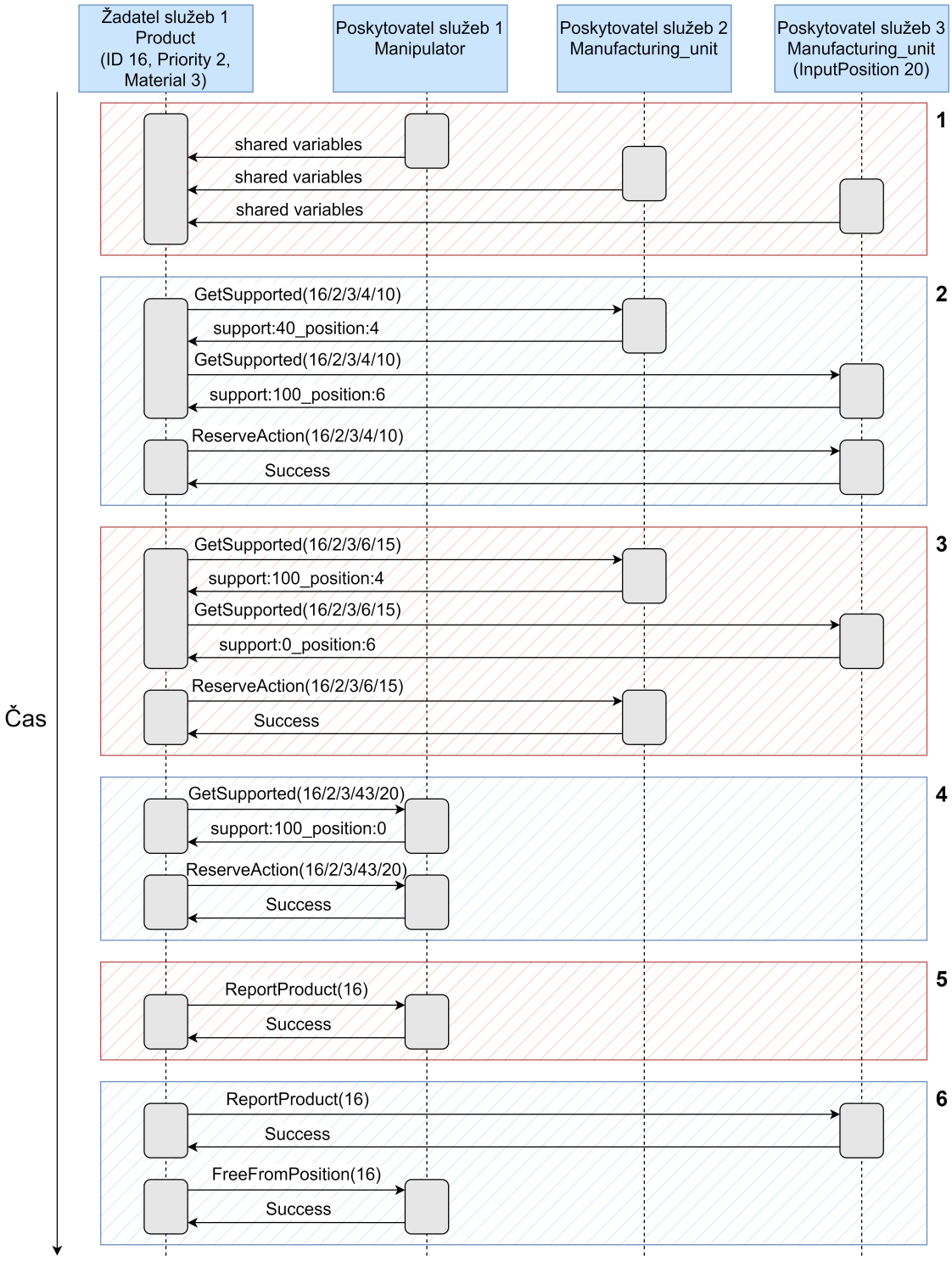
Obr. 6.3: Model komunikačního rozhraní uvnitř centrálního hubu.

Jelikož tato práce slouží pouze jako **proof of concept**, bylo po důkladné úvaze rozhodnuto pro univerzální datový typ napříč proměnnými, a to i těch, u kterých jsou v zásadě požadovány pouze cifry. Hlavním důvodem byla snaha o zachování co nejvyššího stupně univerzálnosti, aby formát komunikace byl snáze uchopitelný pro navazující výzkum v této oblasti.

Totéž je patrné i z přepracování datového modelu OPC UA serveru. Ten byl taktéž sjednocen do jednoho univerzálního, právě s oním totožným cílem: zjednodušit co nejvíce případné nasazování a s tím související úpravy aplikace tak, aby vyhovovali daným provozům. Tento přístup k volbě datového typu a datového modelu je založen na snaze o co největší flexibilitu a jednoduchost při implementaci a údržbě systému v průmyslovém prostředí.

6.2.2 Sekvence komunikace

Pro snazší pochopení komunikace se tato část práce zabývá popisem obrázku 6.4. Popis komunikace lze rozdělit do několika následujících částí:



Obr. 6.4: Sekvenční model komunikace.

1. Tato sekce ilustruje, jak žadatelé služeb[4] odebírají sdílené proměnné z OPC UA serverů. Intervaly, které určují frekvenci odesílání těchto proměnných od-

[4]Jedná se o tzv. subscribery.

běratelům, umožňují širokou škálu nastavení. Je velmi výhodné, že PLC řady S7-1500 umožňují nastavit OPC UA servery[5] tak, aby odesílaly data pouze při jejich změně nebo registraci nového odběratele, což výrazně redukuje množství informací na lince.[6]

Na první sekci obrázku 6.4 je tedy zobrazeno pouze prvotní zaslání dat odběratelům při jejich zaregistrování. Další změny v průběhu komunikace již z důvodu snahy o zachování jednoduchosti obrázku zobrazeny nejsou.

2. Žadatel služeb se dle své receptury snaží nalézt optimálního poskytovatele služeb. Musí přitom zohlednit jak podporu dané operace (support), tak i jeho případnou pozici v dané prioritní frontě (position), na základě čehož je schopen odhadnout dobu, po kterou by zde čekal na vykonání požadované operace. Požadavky zasílá pouze těm poskytovatelům služeb, u kterých má ve své interní databázi uloženo, že se jedná právě o daný typ poskytovatele.

   Po vybrání optimálního poskytovatele služeb se u něj zaregistruje pomocí funkce **ReserveAction()** s očekávanou odpovědí **Success**. Stále se však nenachází v jeho sféře vlivu, takže provedení akce je pořád blokováno funkcí **ReportProduct**. Tuto metodu produkt zasílá v této fázi pouze zařízením typu **Storage** při vytvoření AAS, neboť z logiky věci se jedná o první část jeho receptury a tedy jeho vytvoření je podmíněno pouze tím, zdali se daný materiál nachází v dané jednotce.[7] Tuto kontrolu materiálu zajišťuje nenulová část odpovědi **support**.

   Zpráva **16/2/3/4/10** metody **GetSupported** zde znamená:

   - 16 = ID žadatele služeb;
   - 2 = priorita žadatele služeb, která je dána administrátorem při jeho vzniku;
   - 3 = materiál žadatele služeb, definován jeho recepturou;
   - 4 = požadovaná operace dle receptury;
   - 10 = parametr operace dle receptury.

   V tomto případě si produkt vybral poskytovatele služeb č. 3, neboť přestože by byl v prioritní frontě o dvě místa později než u poskytovatele služeb č. 2, má daleko vyšší podporu dané operace. Detailnější zdůvodnění výběru žadatele služeb je dále uvedeno v kapitole 7 Implementace žadatelů služeb.

3. Tato část komunikace je typicky identická s částí č. 2. Žadatel služeb si objednává požadovanou službu podle následující části své receptury. Důvodem je, aby v každém okamžiku bylo definováno následující zařízení, kam půjde

[5] Zmiňované nastavení je v aktuálním SW v době psaní této práce nativní, avšak lze ho pozměnit.

[6] Samozřejmě musí mimo jiné i s definovanou periodou zasílat signál **KeepAlive**, který ujišťuje odběratele, že OPC UA server je stále aktivní.

[7] Myšleno zařízení **Storage**.

s cílem obstarat své služby. Po každém úspěšném splnění dílčí části své receptury si automaticky vyjedná následující část tak, aby byl vždy[8] zaregistrován v prioritní frontě u jednoho až dvou poskytovatelů služeb.[9]

Zde poslední část zprávy **GetSupported 6/15** opět znamena operaci číslem **6** a její parametr podle receptury číslem **15**. Žadatel služeb si vybírá poskytovatele č. 2, neboť poskytovatel služeb č. 3 danou operaci nepodporuje.

4. Nyní je produkt zaregistrován u poskytovatelů služeb a musí si zajistit transport do jejích sfér vlivů. Volá tedy metodou **GetSupported()** všechny[10] poskytovatele služeb s typem **Manipulator** s cílem najít si optimální zařízení na transport.

   Zde poslední část zprávy **GetSupported 43/20** již nereprezentuje operaci a její parametr[11], nýbrž jeho aktuální pozici (43) a pozici, na kterou se chce dostat (20). Svou vlastní pozici (43) zná ze své interní databáze[12]. Pozici 20 zná taktéž ze své interní databáze, neboť mu bylo sdělena v části 1 uvnitř parametru **InputPosition** daného poskytovatele služeb.

   Zde je nutno poznamenat, že v autorově implementaci AAS je předpoklad, že se žadatelé služeb mohou vždy pomocí sady manipulátorů dostavit na všechny pozice ve výrobě. Nedodržení této podmínky by znamenalo uvíznutí žadatele služeb na své aktuální pozici.

5. Mezi ukončením sekce 4 a začátkem sekce 5 může nastat časové okno. V uvedeném příkladu na obrázku 6.4 se již produkt nachází na pozici **43**, a tak může nahlásit manipulátoru, že se nachází v jeho sféře vlivu a může s ním začít pracovat.

6. Mezi sekcemi 5 a 6 je časové okno, během kterého žadatel služeb čeká na vyřízení služby. Poté, co produkt obdrží od manipulátoru informaci, že byl transportován na požadovanou vstupní pozici u zařízení typu **Manufacturing_unit**, nahlásí poskytovateli služeb, že se nachází v jeho sféře vlivu. Zároveň informuje manipulátor, že opouští jeho pozici, a tím umožňuje manipulátoru poskytovat své služby dalším produktům. Je nutné podotknout, že toto nahlašování je nezbytné zejména z důvodu virtualizace testování. V reálném provozu by metody **ReportProduct()** a **FreeFromPosition()** nebyly potřebné, neboť by poskytovatelé služeb byli vybaveni vhodnými čidly, která by jim poskytovala informace o polohách produktů.

   Vzhledem k tomu, že na konci sekce 6 je produkt zaregistrován pouze u jednoho

[8] Vyjímaje případy před první a během poslední části receptury.

[9] Myšleno zařízení **Manufacturing_unit** a nebo **Manipulator**.

[10] V tomto případě je zde pouze jeden poskytovatel služeb typu **Manipulator**.

[11] U zařízení typu **Storage** by poslední dva identifikátory **x/y** znamenali $\mathbf{x} = 0$ pro příjem a 1 pro výdej materiálu a **y** by byla daná pozice.

[12] Taktéž známe jako pasivní části AAS.

poskytovatele služeb typu **Manufacturing_unit** / **Storage**, následovalo by opět vyjednávání a zajištění dalších služeb podle jeho receptury, jak bylo uvedeno v sekcích 3 - 4. V případě ukončení receptury by si produkt zajistil uložení do vhodného skladu, to jest poskytovatele služeb typu **Storage**.

6.3 Zabezpečení komunikace

Jak již bylo nastíněno v autorově bakalářské práci, je nezbytné zabezpečit komunikaci mezi poskytovateli a žadateli služeb. Navržená implementace AAS do výroby je už ze své podstaty extrémně citlivá na vnější útoky. Tato sekce zmiňuje nejvýraznější rizika a doporučení k jejich minimalizaci.

6.3.1 Šifrování zpráv

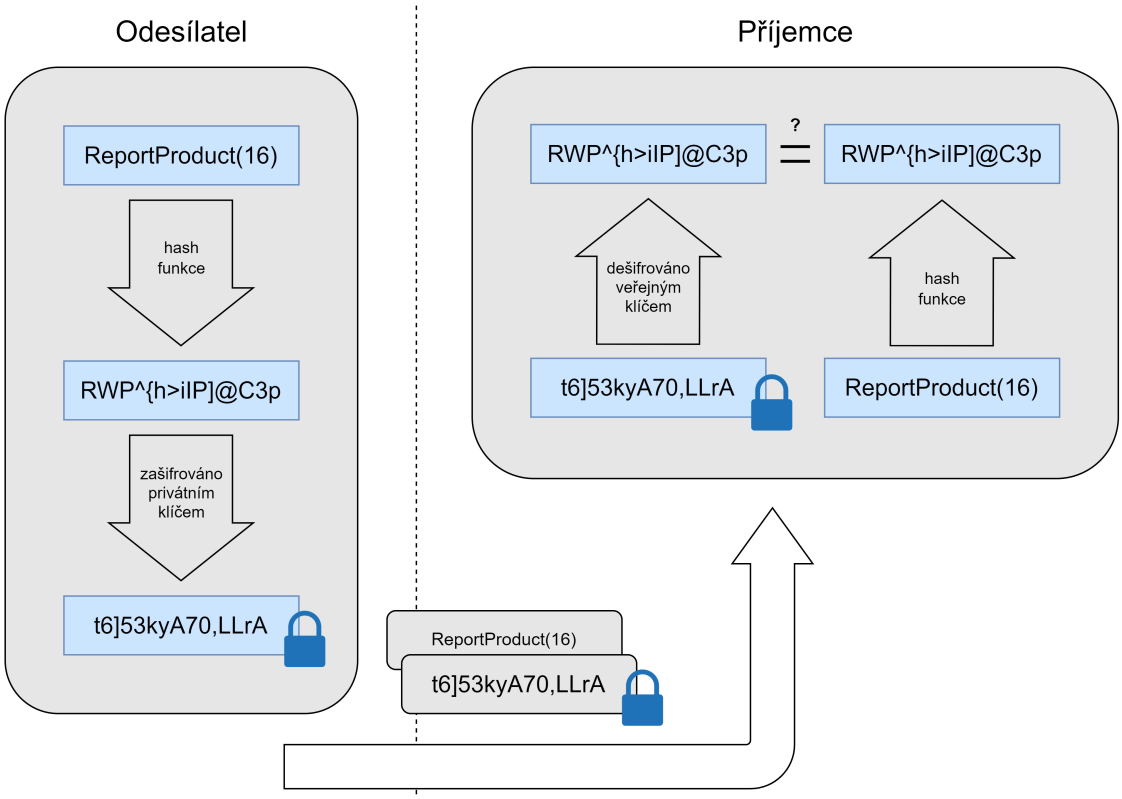
Prvním důležitým předpokladem je zabezpečení samostatných zpráv. Existují tři základní typy zabezpečení, což jsou **None**, **Sign** a **Sign&Encrypt**. Volba možnosti **None**, tedy nepoužití žádného šifrování, je v průmyslovém prostředí nepřijatelná. I když tato možnost umožňuje rychlé testování bez komplikovaného nastavení, představuje vážné riziko pro bezpečnost dat a může být zneužita neoprávněnými stranami. Útočníci by mohli odposlouchávat a manipulovat se zprávami, což může vést k narušení výrobních procesů a úniku citlivých informací.

Druhou možností je zabezpečení ve formě **Sign**, která přináší digitální podpis pro všechny zprávy, což umožňuje zařízením ověřit původ zpráv a odmítnout zprávy od neověřených zdrojů. Tato metoda je zaměřena na eliminaci hlavního problému, kterým jsou útoky třetích stran. Avšak i přesto, že digitální podpis brání neoprávněnému přístupu, neřeší problém s veřejnou dostupností zpráv. Bez dodatečné ochrany mohou útočníci stále získávat informace a analyzovat veřejně dostupné zprávy, což může vést k narušení důvěrnosti dat.

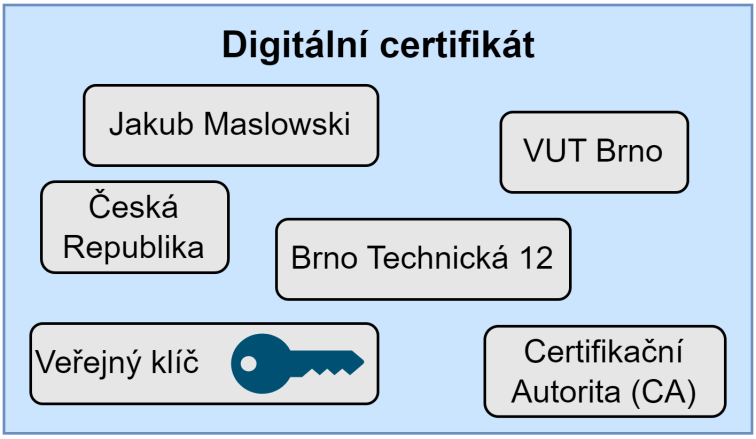
Pro lepší ilustraci problematiky je k dispozici obrázek 6.5, který popisuje, že odesílatel zasílá kromě samotné zprávy i její zašifrovaný hash. Příjemce se pak díky veřejnému klíči může přesvědčit o identitě odesílatele. Zmiňovaný veřejný klíč je součástí certifikátu odesílatele, jež ilustruje obrázek 6.6. Siemens PLC řady S7-1500 umožňují nastavit, zdali povolí přijímání neznámých certifikátů v průběhu programu. V praxi je doporučeno toto nepovolovat a předem nahrát do PLC důvěryhodné certifikáty, případně pak povolit pouze certifikáty od určité Certifikační Autority (CA).[13]

Třetí možností, **Sign&Encrypt**, přináší komplexnější zabezpečení. Kromě digitálního podpisu pro ověření původu zpráv jsou také zašifrována všechna data (tj.

[13]K dosažení důvěryhodnosti certifikátů se používají Certifikační Autority.



Obr. 6.5: Princip fungování podepisování zpráv.



Obr. 6.6: Obsah digitálního certifikátu.

zprávy). Tato kombinace umožňuje ochranu dat v průmyslovém prostředí před útoky třetích stran. Šifrování dat ztěžuje útočníkům dešifrování obsahu zpráv a chrání tak citlivé informace. Rozhodnutí o využití **Sign&Encrypt** či jiného přístupu závisí na konkrétních potřebách a bezpečnostních požadavcích jednotlivých provozů. Některé továrny mohou preferovat tuto kombinaci pro ochranu firemního tajemství, zatímco

jiné mohou preferovat ukládání citlivých informací v poskytovatelích služeb, z čehož vyplývá, že jednotlivé operace a parametry operace pak nebudou tajné a vystačí si se zabezpečením typu **Sign**.

Nutno tedy zdůraznit, že správná volba zabezpečení závisí na specifických potřebách a bezpečnostních požadavcích každého provozu. Důkladná analýza a porozumění rizikům je nezbytná pro správnou volbu implementace.

7 Implementace žadatelů služeb

Tato kapitola se zabývá implementací žadatele služeb, jeho umístěním, modelem komunikace a zabezpečením.

7.1 Volba programovacího jazyku aktivního AAS

Prvním důležitým rozhodnutím je vybrat vhodný programovací jazyk pro aplikaci reprezentující OPC UA klienta. Po zralé úvaze s vedoucím práce byl zvolen jazyk Python. Tato sekce přibližuje důvody této na první pohled atypické volby.

Python vyniká svou jednoduchou syntaxí a modulárním přístupem, což usnadňuje rozvoj a údržbu aplikací pro aktivní AAS. Jeho schopnost snadno čitelného kódu a jednoduchosti v syntaxi může přispívat k rychlému vývoji. Díky tomu mohou případní vývojáři rychle iterovat a experimentovat s různými funkcionalitami, což je klíčové pro tvorbu této **proof of concept** aplikace.

Python nabízí bohatou paletu nástrojů pro implementaci komplexních zabezpečovacích mechanismů. Nabízí možnosti šifrování komunikace typu **Sign** i **Sign&Encrypt** pomocí příslušných knihoven, což je nezbytné pro zajištění bezpečné komunikace s PLC servery a ochranu dat před neoprávněným přístupem.

7.1.1 Podpora existujících vzorů a knihoven

Python disponuje rozsáhlou knihovnou a ekosystémem nástrojů, které vývojářům umožňují snadnou integraci s existujícími vzory a knihovnami. Projekt admin-shell-io od IDTA, dostupný v repozitáři `https://github.com/admin-shell-io`, poskytuje komplexní soubor standardizovaných vzorů pro submodely pasivního AAS.

Díky této podpoře mohou budoucí vývojáři stavět na existujících vzorech a knihovnách, což snižuje náklady a čas potřebný k implementaci této aplikace. Zároveň usnadňuje dodržování osvědčených postupů a stále se rozvíjejících standardů v průmyslovém prostředí, což mimo jiné přispívá i k lepší interoperabilitě.

7.1.2 Flexibilita a snadná použitelnost pro proof of concept

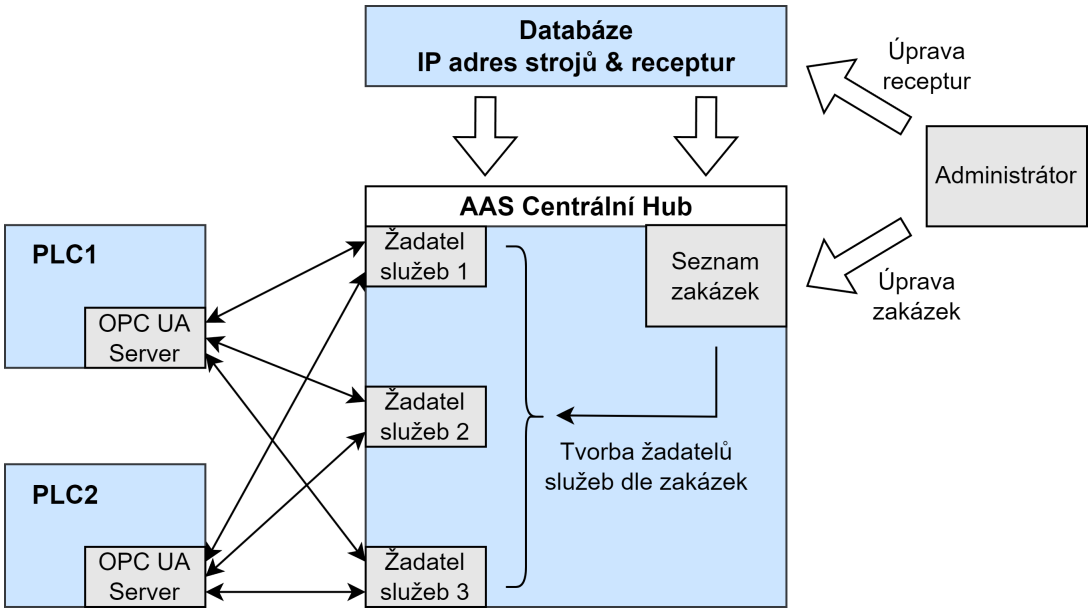
Navzdory tomu, že Python jako klient není nejrychlejším programovacím jazykem pro komunikaci s OPC UA servery, jeho snadná použitelnost a flexibilita ho činí ideální volbou pro vytváření proof of concept aplikací. V počátečních fázích vývoje je důležitější rychlý vývoj a testování nových konceptů než dosažení maximálního výkonu a optimalizace.

Python je jedním z nejpoužívanějších programovacích jazyků na světě a těší se velké podpoře komunity vývojářů. To znamená, že existuje bohatá základna online

zdrojů, tutoriálů a komunitních fór, které mohou být využity při vývoji aplikací pro AAS ve formě žadatelů služeb.

7.2 Model komunikace v OPC UA

Kvůli limitacím PLC je nutné upravit model komunikace od představ AAS tak, jak je popsán v sekci 3.1 Teoretické pojetí AAS. Je také vhodné, aby více AAS ve formě žadatelů služeb bylo na stejném serveru. Na začátku bylo vytvořeno schéma komunikace aktivního AAS, zobrazeno na obrázku 7.1.



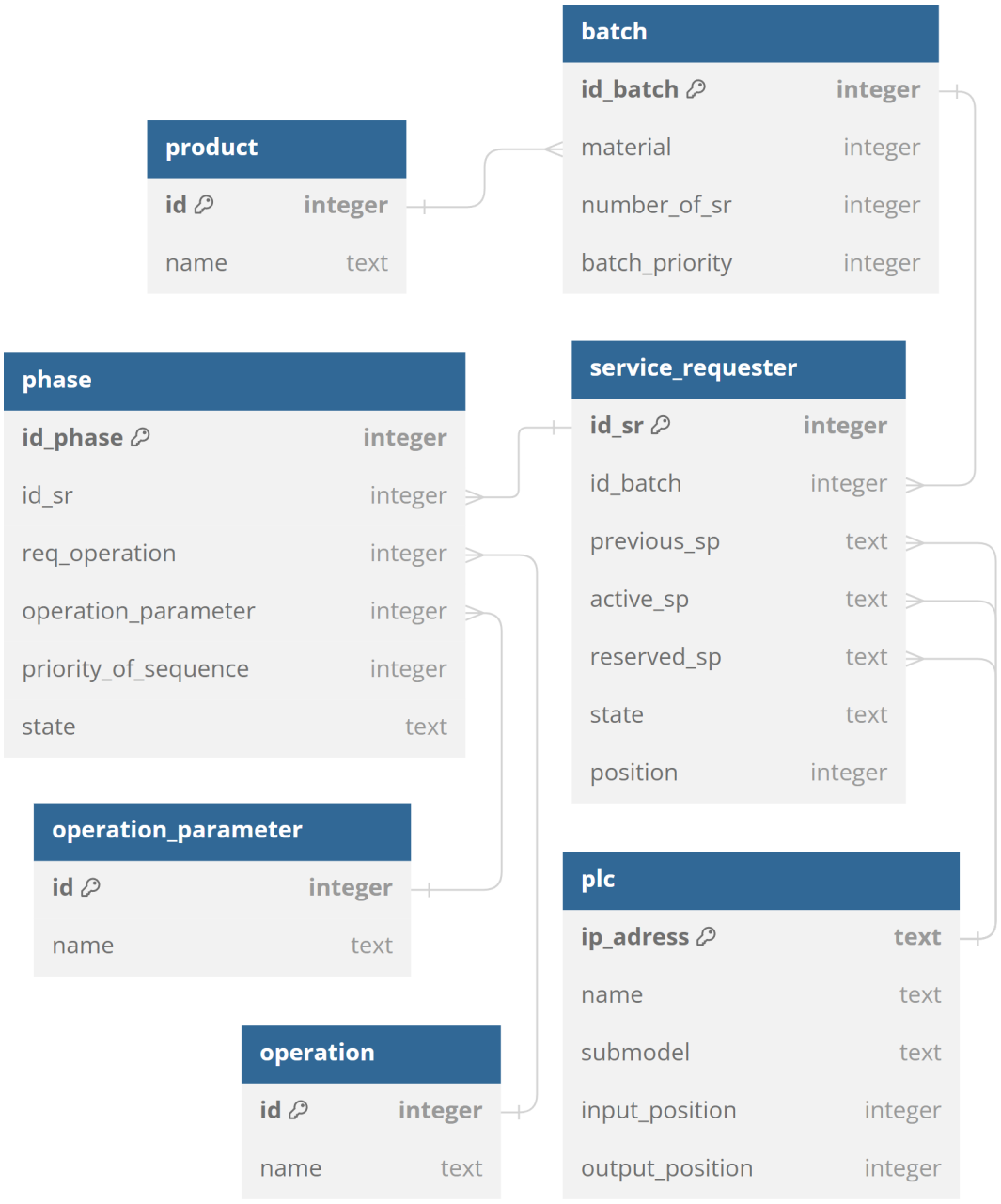
Obr. 7.1: Model komunikace žadatelů služeb.

Tento model komunikace obsahuje centrální AAS hub. Ten dle zakázek vytváří žadatele služeb, kteří mají přístup k databázi IP adres PLC a receptur. Dle nich komunikují s PLC, snažíce se splnit své receptury. Úprava samotných receptur a zakázek je pak prováděna manuálně administrátorem.

V průběhu implementace však došlo ke změně, a to sice že vyšla najevo skutečnost, že oddělovat seznam zakázek od databáze IP adres strojů a receptur je sice možné, avšak zbytečně komplikované, neboť v průběhu programu jsou tyto informace volány stejně často jako ty ze seznamu zakázek. Jednotliví žadatelé služeb totiž nejsou implementováni ve formě subprogramů, nýbrž jako části celkového AAS hubu, který cyklicky zpřístupňuje požadované informace jednotlivým žadatelům služeb.

Z tohoto důvodu byla vytvořena centrální databáze, která je zobrazena na obrázku 7.2. Z ní je patrné, že jednotlivé receptury produktů a seznam IP adres strojů

jsou přes žadatele služeb částečně propojeny. Databáze je tvořena sedmi navzájem propojenými tabulkami, kde centrálním prvkem jsou jednotlivé fáze[1], které se vykonávají dle příslušných priorit[2] a jejich stavy jsou průběžně aktualizovány přes proměnnou **state**, která může nabývat hodnot **none/inProgress/finished/failed**.

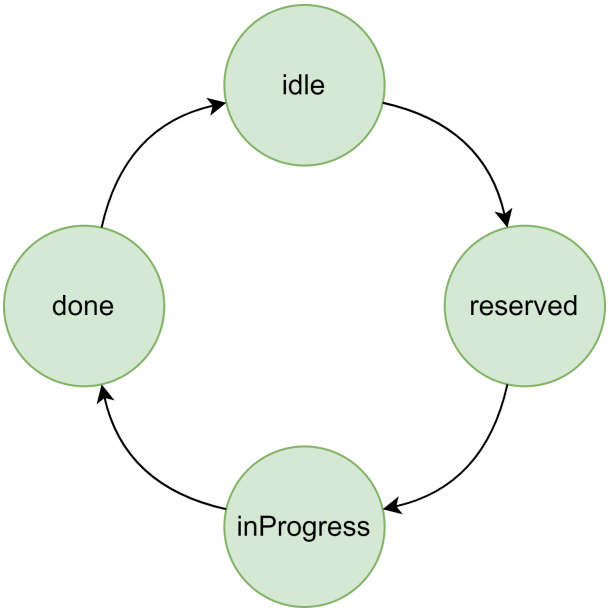


Obr. 7.2: Databáze pro žadatele služeb.

[1] V tabulce označeny jako **phase**.
[2] V tabulce označeny jako **priority_of_sequence**.

7.3 Stavy AAS ve formě žadatelů služeb

Jednotlivé stavy AAS byly výrazně pozměněny od těch z autorovy bakalářské práce. V původním návrhu byly pouze stavy **idle**, **reserved**, **inProgress** a **done**, jak je zobrazeno na obrázku 7.3. Tento návrh musel být v této práci pozměněn, především z důvodu přidání bufferu operací a s tím souvisejícího vylepšení informačního modelu komunikace.



Obr. 7.3: Stavový diagram AAS podle datovém modelu v autorově bakalářské práci.

Ve vylepšeném modelu komunikace, který je zobrazen na obrázku 7.4, je stavů sedm a mají bohatší škálu funkcionalit:

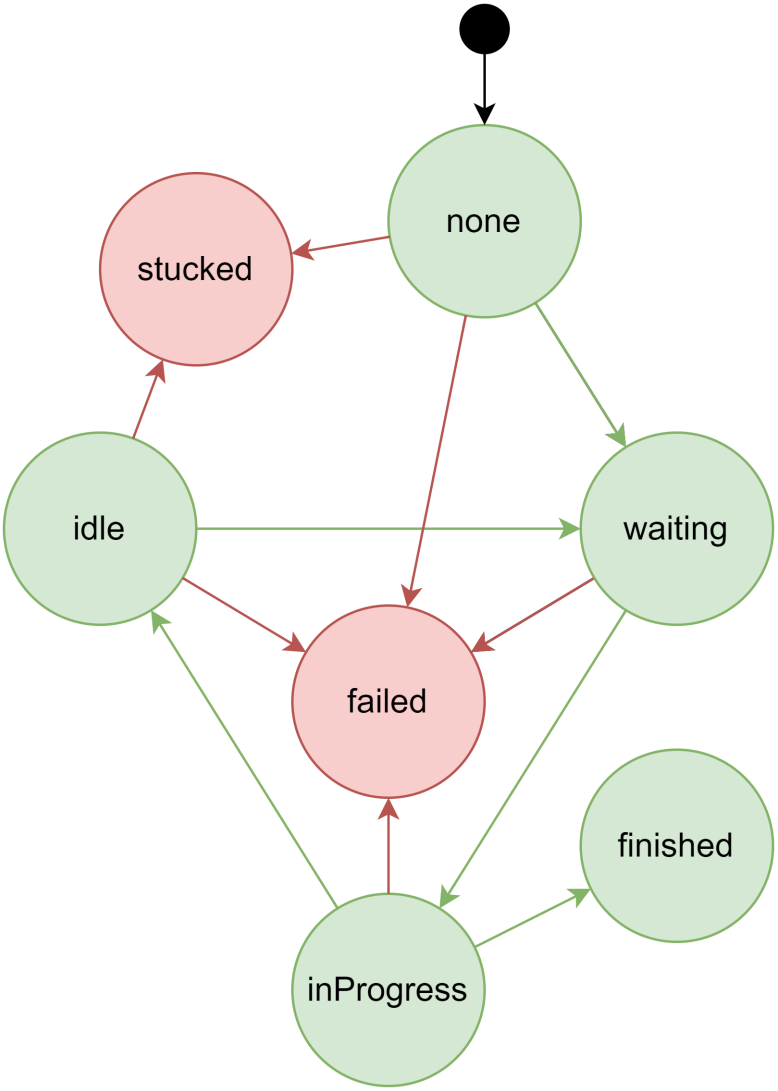
- **none**: Jedná se o prvotní stav žadatele služeb ihned po vytvoření administrátorem. V této fázi žadatel služeb poptává zařízení typu **Storage** s jeho prvotním požadavkem na materiál podle jeho receptury. Pokud tato akce proběhne úspěšně, přechází do stavu **waiting**, kde čeká v prioritní frontě daného poskytovatele služeb. Jestliže se mu však nepodaří najít žádného poskytovatele služeb, který by mu byl ochoten splnit jeho požadavek, dostává se do stavu **stucked**[3]. V případě neočekávané chyby od poskytovatele služeb přechází do stavu **failed**
- **waiting**: Produkt v tomto stavu pravidelně kontroluje sdílené proměnné **CurrentId** a **OutputPositionSr**. V případě, že díky proměnné **CurrentId** zjistí, že je aktuálně v procesu daného zařízení, přejde do stavu **inProgress**. Totéž

[3]V překladu zaseknutý.

učiní i v případě, kdy je již v proměnné **OutputPositionSr**, udávající, že již skončil na výstupní pozici daného stroje. Ve zmiňovaných případech se rovněž díky OPC UA metodě **FreeFromPosition()** odregistruje z pozice předcházejícího zařízení, čímž uvolní místo pro další produkty.[4] Žadatel služeb rovněž aktualizuje stav dané fáze na stejnou hodnotu, jakou má on sám.

- **inProgress**: Poskytovatel služeb v tomto stavu čeká, až se v proměnné **OutputPositionSr** objeví jeho ID, poté přechází do stavu **idle**. Pokud se jedná o poslední fázi, odregistruje se od aktuálního zařízení typu **Storage**. V případě úspěchu přejde do stavu **finished**, v opačném případě přechází do stavu **failed**. Stav fáze se aktualizuje stejně, s vyjímkou stavu **idle**, kde fáze přechází do stavu **finished**.
- **idle**: Pokud jde o následné přechody, tento stav je velmi podobný stavu **none**. Zde se však žadatel služeb již nachází ve výrobě. Postupuje tedy dále podle schématu znázorněného na obrázku 7.1. Stejně jako v případě stavu **none**, může přejít do stavu **stucked** či **failed** v případě výskytu příslušných chyb.
- **stucked**: V aktuální autorově implementaci je tento stav konečný a žadatel služeb se stává neaktivním. Toto bylo nastaveno s ohledem na budoucí testování, kde bylo žádoucí ponechat tyto stavy pro pozdější analýzu. V praxi by bylo vhodné tyto produkty ve stavu **stucked** pravidelně kontrolovat s kratší periodou, zda se již někde nenabízí požadovaná služba, případně je zařadit zpět do skladovací jednotky.
- **failed**: Tento stav je také konečný, podobně jako stav **stucked**. Jak již bylo zmíněno, produkt se sem dostane pouze v případě neočekávané chyby ze strany poskytovatele služeb, například při neúspěšném odregistrování pomocí metody **FreeFromPosition()** či neúspěšného zaregistrování se metodou **ReportProduct()**. Tyto stavy mohou tedy nastat pouze v případě chyb poskytovatelů služeb, a v praxi by vždy měly upozornit operátora výroby vhodným alarmem.
- **finished**: Produkt se dostává do této fáze, pokud všechny fáze jeho receptury byly úspěšně splněny. Dostává se zde vždy od zařízení typu **Storage** po úspěšném založení do skladu. Každý produkt v autorově implementaci má uložení do skladu zapsáno v receptuře jakožto poslední fázi s **operation** = 1; **operation_parameter** = 0 a **priority_of_sequence** = 10. Toto umožňuje snadnou kontrolu počtu fází produktu v průběhu programu.

[4] Zde si lze představit, že produkt ve stavu **waiting** stále zabírá místo u manipulátoru.



Obr. 7.4: Vylepšený stavový diagram AAS žadatelů služeb.

7.4 Programová část žadatelů služeb

Tato část práce se zabývá především programové části práce v Pythonu reprezentující AAS client hub.

Program jménem **serviceRequester.py** reprezentuje žadatele služeb ve formě centrálního AAS client hubu. Principiálně je jeho chování popsáno na obrázcích 6.4 a 7.4 spolu s popisy v příslušných sekcích.

7.4.1 Knihovny

V rámci implementace bylo využito několik knihoven. Ty aktivní jsou uvedeny v tabulce 7.1. Dále jsou v kódu, který se nachází v příloze této práce, doporučené

knihovny **cryptography** a **xml**. První zmiňovaná se využívá v rámci navrhnutého řešení zašifrované komunikace, která však není využita z důvodu efektivního testování. Druhá se pak doporučuje využít pro lepší práci s .xml soubory, které byly vygenerovány pomocí SiOME.[5]

Tab. 7.1: Použité knihovny v Python kódu a jejich využití.

Knihovna	**Popis**
opcua	Knihovna pro komunikaci pomocí protokolu OPC UA. Používá se pro komunikaci s PLC a jako jediná z této tabulky není standardní knihovnou a musí se doinstalovat. Toto se v jazyce Python provádí jednoduše příkazem **pip install opcua** v příkazovém řádku.
time	Standardní knihovna Pythonu pro práci s časem. Používá se z ní funkce **sleep**, využívaná pro zdržení procesoru jednak z důvodu úspory výpočetního výkonu, jednak z důvodu simulace delších prodlev od PLC v komunikaci.
shutil	Knihovna pro manipulaci s soubory a adresáři. Používá se pro kopírování, přesunování a mazání souborů, v tomto případě databáze a logů.
sqlite3	Knihovna pro práci se SQLite databází. SQLite je lehká a snadno použitelná relační databáze.
random	Standardní knihovna Pythonu pro generování náhodných čísel. Používá se pro generování receptur žadatelů služeb, neboť při testování nejsou k dispozici přímé požadavky daných provozů.
sys	Standardní knihovna Pythonu pro interakci se systémem. Používá se pro ukončení programu v případě nedostupných poskytovatelů služeb.
logging	Knihovna pro logování událostí v Pythonu. Používá se pro záznam informací z jednotlivých poskytovatelů služeb i aktualizací žadatelů služeb v průběhu provádění programu.

7.4.2 Třídy

Z důvodu navrhované implementace nebylo zapotřebí vytvářet velké množství tříd. Zde byla vytvořena pouze třída **Plc**, jejíž struktura je totožná s tabulkou **plc** v centrální databázi. Byla vytvořena pro efektivní práci s daty v rámci programu. Druhou

[5]Tyto doporučení slouží pro snazší navázání na tuto práci v budoucnu.

třídou pak je **SubHandler**, která je specifická v rámci udržování kontaktu s poskytovateli služby a jejich daty.

7.4.3 Napojení na OPC UA rozhraní

Vzhledem ke skutečnosti, že v této práci se namísto využívání aplikace **UAExpert** vytváří vlastní OPC UA klient v jazyce Python, bylo nutné provést částečnou standardizaci OPC UA prostředí. Zde byla opět snaha využít program **SiOME** pro jeho úpravu, jak již bylo nastíněno v sekci 5.2 Rozlišení jednotlivých částí AAS v PLC a na obrázku 5.1 na stránce 30.

Přestože jsou veškeré proměnné, metody i objekty[6] v OPC UA rozhraní jinak pojmenovány[7], jejich identifikace probíhá pomocí tzv. **NodeId**. Tyto **NodeId** musí být jedinečné. V prostředí SiOME se identifikátory přiřazují nativně a nelze je zde měnit. Jiné aplikace, které by toto umožňovaly, autor v době psaní práce nenašel. Z tohoto důvodu byly úpravy identifikátorů provedeny manuálně v čistém .xml souboru, jak je uvedeno v tabulce 7.2.

Tab. 7.2: Napojení proměnných a metod na OPC UA rozhraní.

Jméno	**NodeId**	**Jméno**	**NodeId**
FreeFromPosition	ns=1;i=7000	ItemCountRP	ns=1;i=6104
FreeFromQueue	ns=1;i=7001	OutputPositionSr	ns=1;i=6105
GetStatus	ns=1;i=7002	QueueHistoryBuffer	ns=1;i=6106
GetSupported	ns=1;i=7003	QueueStatus	ns=1;i=6107
ReportProduct	ns=1;i=7004	ID	ns=1;i=6200
ReserveAction	ns=1;i=7005	InputPosition	ns=1;i=6201
ActionStatus	ns=1;i=6100	Name	ns=1;i=6202
CurrentId	ns=1;i=6101	OutputPosition	ns=1;i=6203
CurrentPosition	ns=1;i=6102	Submodel	ns=1;i=6204
ItemCountP	ns=1;i=6103	TotalMaxBuffer	ns=1;i=6205

Kromě nich je při volání metod nutné znát i **NodeId** rodiče (tj. umístění metody) a dětí (tj. vstupních a výstupních parametrů metody). **NodeId** dětí je možné jednoduše zjistit pomocí knihoven Python kódu. Byly zde dvě hlavní varianty implementace:

1. Mít v aplikaci uloženo pouze **NodeId** nejvyššího rodiče (viz. objekt **PLCServiceProvider** na obrázku 5.1) a veškeré volání metod a proměnných zjišťovat pomocí dotazů na jména jeho potomků. Tato metoda je nejuniverzálnější,

[6]Funkcionalitu objektů si lze představit jako složky.

[7]Mohou mít i stejné názvy, což však znepřehledňuje jejich prohlížení.

avšak taktéž nejnáročnější na implementaci a samotný výpočetní výkon.[8]

2. Mít v aplikaci uloženy **NodeId** veškerých potřebných entit. Toto řešení výrazně ubírá na univerzálnosti, neboť by bylo zapotřebí namapovat metody na specifické OPC UA prostředí. Podstatnou výhodou je však nulifikace rizika spojeného s případnou vyšší výpočetní náročností aplikace.

Po důkladné konzultaci s vedoucím této práce bylo rozhodnuto pro variantu č. 2, neboť se jedná o bezpečnější variantu co se případných rizik týče. Zmiňované **NodeId** jsou tedy zakomponovány v rámci samotného programu. Napojení objektů na OPC UA rozhraní je rovněž k dispozici v tabulce 7.3, z které se v této implementaci využívá pouze **NodeId** objektu **Methods**.

Tab. 7.3: Napojení objektů na OPC UA rozhraní.

Jméno	**NodeId**
opc.tcp://192.168.0.1[9]	ns=1;i=5000
PLCServiceProvider	ns=1;i=5001
Methods	ns=1;i=5002
Variables	ns=1;i=5003
Dynamic	ns=1;i=5004
Static	ns=1;i=5005

Je důležité poznamenat, že v aplikaci **serviceRequester.py** nejsou využity všechny metody a proměnné z OPC UA prostředí. Metody a proměnné na PLC byly vytvořeny s cílem poskytnout širokou škálu nástrojů pro aplikace s AAS. Některé jsou tedy určeny primárně pro snazší ovládání a sledování AAS v manuálním režimu. Zde si lze představit technika či operátora ve výrobě, kteří mají odlišné chování než AAS client hub. Další možností využití těchto metod a proměnných jsou pak případné navazující aplikace, které budou vyžadovat více dat z pasivního AAS jednotlivých poskytovatelů služeb.

V programu nebyly využity metody **GetStatus()** a **FreeFromQueue()**. Žadatel služeb si pamatuje svou pozici v rámci databáze, a proto není nutné během běhu programu přímo zjišťovat jeho stav pomocí metody **GetStatus()**. Rovněž nemá důvod odregistrovat se od již zaregistrovaných poskytovatelů služeb, a proto metodu **FreeFromQueue()** také nevyužívá. Nicméně obě metody by mohly být užitečné pro budoucí rozšíření kódu.

Taktéž nebyly využity dynamické proměnné **CurrentPosition**, **ItemCountP**, **ItemCountRP** a **QueueHistoryBuffer**. Tyto proměnné jsou předpokládány pro

[8] Z hlediska zvýšeného výpočetního výkonu se jedná o neznámou proměnnou.

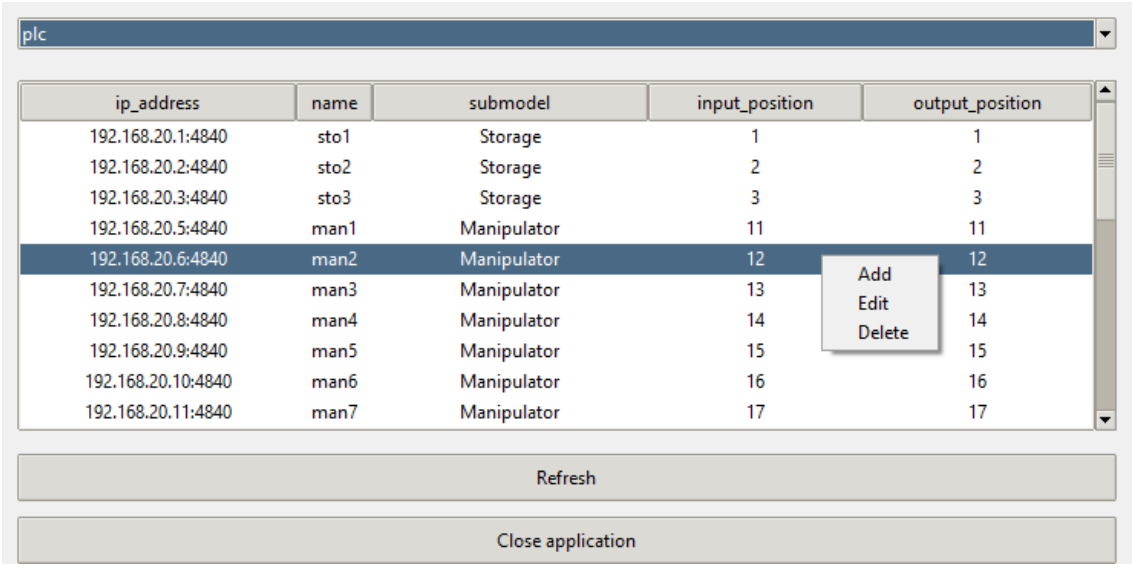
[9] Jedná se **namespace** daného prostředí. Nemá vliv na propojení v PLC.

především manuální využití ve výrobě. Statické proměnné nebyly vůbec využity, neboť jejich hodnoty se ze své podstaty nemění a je vhodné mít je již lokálně uložené v databázi.

Celkově se tedy v aplikaci využily 2/3 metod, 1/2 dynamických proměnných a žádná ze statických proměnných. Toto ale vůbec nevadí, neboť skutečnost, že OPC UA server nabízí vícero metod a proměnných má nulový vliv na jeho výkonnost a zanedbatelný vliv na jeho paměť.

7.5 Grafické uživatelské rozhraní AAS client hubu

Komunikační rozhraní navrhnuté aplikace **serviceRequester.py** je realizováno pomocí databáze **orders.db**. Tato databáze umožňuje externí úpravy zakázek a s tím související počet žadatelů služeb. Přestože to nebylo součástí zadání této práce, z důvodu snazšího pochopení přístupu k oněm datům autor práce navíc vytvořil i aplikaci **ordersGUI.py**[10]. Tato aplikace umožňuje uživatelům provádět uvedené změny a zároveň sledovat aktuální stav plnění zakázek. Její obrazovka je zobrazena na obrázku 7.5. Je důležité zdůraznit, že cílem této aplikace není plnohodnotně nahradit nástroje operátora či administrátora, ale spíše demonstrovat možnosti přístupu k datům a sloužit jako ukázka **proof of concept**. Tímto způsobem lze plynule navázat na další výzkum v této oblasti třetími stranami.



Obr. 7.5: Snímek aplikace ordersGUI.

[10]Obě aplikace byly taktéž vyexportovány do .exe souborů v příloze této práce.

8 Výsledky práce

Tato kapitola se zabývá nasazením AAS ve virtuální výrobě včetně diskuze výsledků.

8.1 Nastavení implementace AAS během testování

Jak již bylo naznačeno, celá tato práce je silně generalizována. Z tohoto důvodu jsou příslušná nastavení AAS popsány v této sekci. V praxi by tato nastavení byla silně ovlivněna konkrétními provozy. V průběhu testování nebylo provedeno zabezpečení komunikace, tedy bylo zde zabezpečení typu **None**.

Úkolem testování bylo zjistit nedostatky implementace AAS ve výrobě. Proto byla provedena specifická nastavení, která nebyla zaměřena na vytvoření ideální výroby, ale na vytvoření prostředí s mnoha problémy. Cílem bylo analyzovat chování AAS a identifikovat případné nedostatky v samotné architektuře AAS pro budoucí výzkum.

8.1.1 Nastavení poskytovatelů služeb během testování

Bylo rozhodnuto pro vytvoření sítě PLC většího rozsahu, aby byly patrné případné problémy s propustností sítě. Počet jednotlivých typů poskytovatelů služeb byl následující: 3x **Storage**, 8x **Manipulator** a 18x **Manufacturing_unit**, což dohromady představuje 29 PLC. Doby trvání jednotlivých operací závisely na typu poskytovatele služeb a na jeho podpoře dané služby. Tyto časy jsou přehledně uvedeny v tabulce 8.1.

Dále byly výrazně individualizovány nabídky služeb jednotlivých PLC s cílem dosáhnout co nejbohatší variace. Toto bylo provedeno pomocí proměnných v 1. řádku tabulky 8.2. Tyto proměnné jsou ve formátu **XXXYYYZZZ**, kde:

- **XXX** může nabývat hodnot:
  - **Mat** pro **material**.
  - **SuA** pro **supportA**, tj. operace nebo vstupní pozice manipulátoru.
  - **SuB** pro **supportB**, tj. parametr operace nebo výstupní pozice manipulátoru.

  Jedná se tedy o poslední tři identifikátory ve zprávě **ReserveAction** nebo **GetSupported**.
- **YYY** může nabývat hodnot **Low** a **High**. Definuje, zda se jedná o dolní či horní limit daných parametrů.
- **ZZZ** může nabývat hodnot **100** nebo **60**, což určuje, do jaké míry je daná operace podporována.

Tabulka 8.2 tedy zobrazuje nastavení podpory jednotlivým operacím od poskytovatelů služeb, kde **sto1-3** označuje zařízení typu **Storage**, **man1-8** označuje zařízení typu **Manipulator** a **mix1-18**[1] označuje zařízení typu **Manufacturing_unit**. Tyto podpory se navzájem překrývají. Například PLC s názvem **man1** umožňuje pohybovat pouze s prvky z materiálu **1-4** z a do pozic **21-60** s podporou **20**, neboť je zde 2x odpočet **-40** u vstupní i výstupní pozice, což dle nastavení z tabulky 8.1 odpovídá devíti sekundám.

IP adresy jednotlivých PLC včetně dodatečných informací ohledně poskytovatelů služeb, které rozšiřují tabulku 8.2, jsou k dispozici v příloze **plcList.xlsx**.

Tab. 8.1: Časy potřebné pro vykonání operací v rámci testování AAS na jednotlivých submodelech.

	Podpora		
Submodel	100 %	60 %	20 %
Storage	2,5 s	3,5 s	4,5 s
Manipulator	5 s	7 s	9 s
Manufacturing_unit	10 s	14 s	18 s

8.1.2 Nastavení žadatelů služeb během testování

Žadatelé služeb byli stejně jako PLC silně individualizováni. Databáze produktů a operací jsou k dispozici v tabulkách 8.3 a 8.4. Testování bylo provedeno na 40 žadatelích služeb, definovaných podle tabulky 8.5. Všichni žadatelé služeb měli ve svých recepturách tři operace a návrat do skladovací jednotky. Parametry operací i jejich počty byly voleny náhodně, vždy však s rostoucím ID. Například tedy nemohla nastat operace **packaging** před **painting**, nebo **painting** před **cutting**. Parametry operace byly zvoleny náhodně v rozmezí od 1 do 20. Z obou programů **serviceRequester.py** i **ordersGUI.py** byly vytvořeny .exe soubory včetně ikon pro snazší spouštění na systémech Windows během testování.

8.2 Testování v reálném čase

Kvůli výpočetní náročnosti virtuálních PLC bylo testování AAS provedeno v prostorách Fakulty elektrotechniky a komunikačních technologií VUT v Brně na 8 po-

[1] Zkratka **mix** je odvozena z původního pojmenování tohoto submodelu (**Mixer**) v prvotní verzi AAS pro PLC, který byl představen v článku **Asset Administration Shell for PLC** na EEICT 2022, dostupným z `https://doi.org/10.13164/eeict.2022.49`.

Tab. 8.2: Parametry PLC při testování.

Název PLC	MatLow100	MatHig100	MatLow60	MatHig60	SuALow100	SuAHig100	SuALow60	SuAHigh60	SuBLow100	SuBHigh100	SuBLow60	SuBHigh60
sto1	1	2	3	4	0	0	0	0	1	1	0	0
sto2	3	6	1	2	0	0	0	0	2	2	0	0
sto3	1	6	0	0	1	1	0	0	3	3	0	0
man1	1	4	0	0	0	0	21	60	0	0	21	60
man2	1	4	0	0	1	3	21	60	1	3	21	60
man3	1	4	0	0	21	40	41	60	21	40	41	60
man4	1	4	0	0	41	60	21	40	41	60	21	40
man5	5	6	1	4	0	0	21	60	0	0	21	60
man6	5	6	1	4	1	3	21	60	1	3	21	60
man7	5	6	1	4	21	40	41	60	21	40	41	60
man8	5	6	1	4	41	60	21	40	41	60	21	40
mix1	1	2	0	0	1	2	0	0	0	100	0	0
mix2	3	4	1	2	1	2	0	0	0	100	0	0
mix3	5	6	3	4	1	2	0	0	0	100	0	0
mix4	0	0	1	6	0	0	1	2	0	100	0	0
mix5	1	2	0	0	3	4	0	0	0	100	0	0
mix6	3	4	1	2	3	4	0	0	0	100	0	0
mix7	5	6	3	4	3	4	0	0	0	100	0	0
mix8	0	0	1	6	0	0	3	4	0	100	0	0
mix9	1	2	3	4	5	5	6	6	0	100	0	0
mix10	5	6	3	4	5	5	6	6	0	100	0	0
mix11	1	2	3	4	6	6	5	5	0	100	0	0
mix12	5	6	3	4	6	6	5	5	0	100	0	0
mix13	3	4	0	0	1	1	0	0	0	100	0	0
mix14	3	4	0	0	2	2	0	0	0	100	0	0
mix15	3	4	0	0	3	3	0	0	0	100	0	0
mix16	3	4	0	0	4	4	0	0	0	100	0	0
mix17	3	4	0	0	5	5	0	0	0	100	0	0
mix18	3	4	0	0	6	6	0	0	0	100	0	0

čítačích se softwarem **S7-PLCSIM Advanced V4.0** podle nastavení popsaném v sekci 8.1 Nastavení implementace AAS během testování.

Tab. 8.3: Zaplnění tabulky materiálů databáze orders.db během testování.

Id	Název
1	A1
2	B1
3	A2
4	B2
5	A3
6	B3

Tab. 8.4: Zaplnění tabulky operací databáze orders.db během testování.

Id	Název
1	cutting
2	drilling
3	sanding
4	painting
5	labeling
6	packaging

Tab. 8.5: Seznam jednotlivých dávek AAS pro testování.

Dávka	Materiál	Priorita	Počet produktů	Čas přidání
1	3	0	8	ihned
2	2	2	8	ihned
3	1	4	8	ihned
4	4	0	8	2 minuty po startu
5	6	1	8	2 minuty po startu

8.2.1 Výsledky

Výpočetní zátěž jednotlivých PLC z důvodu využití OPC UA metod s prioritními frontami byla dle očekávání zanedbatelná. Co se však ukázalo jako velmi uspokojivé, byla paměťová náročnost programů s ohledem na využití prioritních front. Paměťová zátěž byla pouze v řádu jednotek procent, a to i při úpravách nastavení **TotalMaxBuffer** z **100** na **10 000**, což umožnilo zaregistrovat **1 000** produktů se stejnou prioritou u žadatele služeb.

Grafické průběhy AAS během testování jsou k dispozici na obrázku 8.1. Z nich je jasně patrné správné chování AAS entit ve výrobě a prioritizace služeb od produktů s vyšší prioritou. Pro správnou analýzu je nutné zkombinovat informace z grafu 8.1 a tabulek 8.2 a 8.3.

Z makro perspektivy je zajímavá část produkce do první minuty. Tam zařízení **sto2** plnilo služby produktů z dávky **1**, neboť byl nejvhodnější pro jeho materiál (tj. **3**). Toto trvalo přibližně 20 sekund. Druhým zařízením typu **Storage** určeným pro výdej produktů bylo **sto1**. Lze pozorovat, že upřednostňoval dávku **2** a poté plynule přešel na produkty z dávky **3**, což je patrné ze skutečnosti, že ve spodním grafu nejsou viditelné žádné propady ve vytížení zařízení typu **Storage**.

Naopak z mikroúrovně je zajímavá poslední minuta na konci, kde lze vidět, jak poslední produkt z dávky 3, která měla nejmenší prioritu, postupuje mezi zařízeními **Manufacturing_unit** -> **Manipulator** -> **Manufacturing_unit** -> **Manipulator** -> **Storage**. Rovněž zde jsou na první pohled vidět rozdíly v časových úsecích u jednotlivých zařízení dle submodelů, což zase odpovídá obsahu tabulky 8.1.

Rovněž zde bylo patrné, že produkty dávaly přednost vyšší podpoře (support), než době trvání z hlediska pozice v prioritní frontě. Toto lze sledovat především na žadatelích služeb z dávky **3**, kteří místo toho, aby si zaregistrovali služby u méně výhodného stroje, raději počkali, až se jim uvolní ten s nejvyšší kompatibilitou. Výsledek je o to přesvědčivější, že produkty z dávky **3** měly nejnižší počet jednotlivých operací. Toto je patrné ze skutečnosti, že poté, co všechny produkty opustily zařízení **Storage**, tak celkový postup dávky již přesáhl 35%, zatímco u dávek **1** a **2** to bylo pouze pod 20%.

Je zde nutno zdůraznit, že celková malá vytíženost strojů na spodním grafu je důsledkem tohoto nastavení. Zde dokonce zařízení man1, man5, mix4 a mix8 nebyly vůbec využity, neboť žádný z poskytovatelů služeb je nevyhodnotil jakožto optimální poskytovatele daných služeb, což je opět patrné z detailní analýzy obsahu tabulky 8.1.

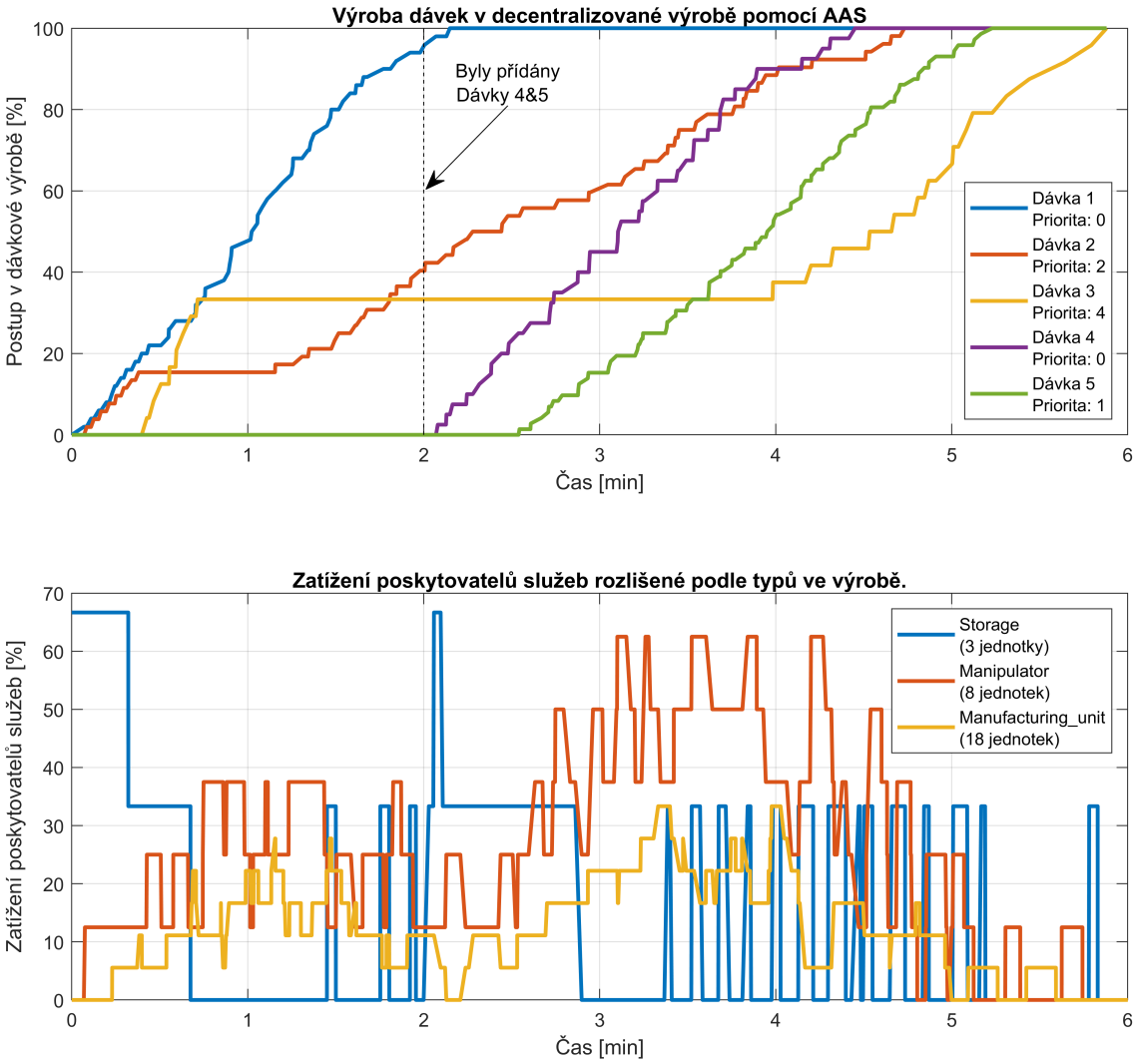
8.3 Realizovatelnost v průmyslu

Jak již bylo řečeno, hlavním cílem této práce bylo otestovat možnost praktického využití AAS. Výsledky v předcházející sekci zcela nepochybně tezi nasazení AAS do průmyslu potvrdily. Nejenom že se prvky ve výrobě chovaly přesně dle očekávání, ale i výpočetní náročnosti na PLC i AAS client hubu byly zanedbatelné.

Aplikovatelnost zjištěných poznatků do velkovýroby je v době psaní této práce téměř zanedbatelná. Existuje zde však prostor aplikovat výsledky této práce v malém měřítku na část výroby, která je silně individualizována, tedy že každý produkt vyžaduje různé operace na odlišných strojích.

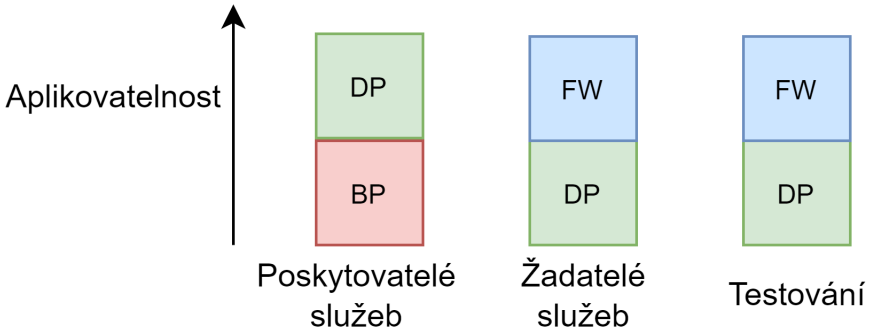
Určitě stojí za zmínku testbed **Barman** na Ústavu automatizace a měřící techniky na FEKT VUT v Brně. Původně byl na tento testbed primárně navrhnut základní model AAS v autorově bakalářské práci. V této práci se uvažovalo spíše s využitím do praxe. Nyní však po zjištění mizivých nároků na PLC v nastavené implementaci AAS přichází opět varianta aplikovat výsledky této práce právě na tento testbed pro účely demonstrace výhod AAS.

Pro porovnání výsledků této práce a s tím související náročnosti je zde obrázek 8.2. Ten znázorňuje, že v rámci autorovy bakalářské práce (BP) byla položena jakási základní stavební kostka AAS pro aplikaci ve výrobě, což nikdo nikdy předtím nezkoušel. V této práci (DP) se dokončily poskytovatelé služeb k dokonalosti



Obr. 8.1: Grafické průběhy AAS během testování.

a byly položeny základní kostky pro žadatele služeb. Stále je však nezbytné na tento výzkum dále navázat v budoucí práci (FW), kde by se dořešily aplikace operátorů ve výrobě a s tím souvisejícího testování.



Obr. 8.2: Aplikovatelnost AAS do průmyslu.

9 Závěr

V rámci této diplomové práce byla podrobně rozebrána problematika Asset Administration Shell spolu s jeho přidruženými tématy. Díky analýze v teoretické části práce byl navržen a implementován systém prioritních front do PLC ve formě polí priorit. Tato aplikace projevila nízké výpočetní a paměťové náročnosti AAS, čímž otevřela možnosti implementace i do PLC nižších řad, například S7-1200.

Ve srovnání s autorovou předchozí bakalářskou prací zde byla významně vylepšena koncepce komunikace díky jednotnému OPC UA prostředí pro všechny typy zařízení, a to včetně standardizace identifikátorů jednotlivých objektů v OPC UA prostředí.

Kromě této skutečnosti byla vyvinuta koncepce žadatele služeb ve formě centrálního AAS client hubu, jež byl vytvořen v programovacím jazyce Python. Koncepce žadatelů služeb byla otestována programem serviceRequester.exe a grafickým rozhraním aplikace ordersGUI.exe, jež slouží ke grafickému zobrazení a manipulaci s daty databáze orders.db.

Testování nového systému prokázalo slibné výsledky. Vzhledem k získaným výsledkům lze konstatovat, že navržené řešení přináší významné inovace v oblasti komunikace a integrace průmyslových zařízení. Tato práce přispívá k rozvoji poznatků v oblasti Asset Administration Shell a jeho aplikace v průmyslovém prostředí a poskytuje tak ucelený pohled na možnosti využití moderních technologií pro efektivnější provoz a správu průmyslových systémů.

Literatura

[1] MASLOWSKI, Jakub. Asset Administration Shell pro PLC. Online, bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2022. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=241426. [cit. 2023-10-27].

[2] GELERNTER, David. Mirror Worlds or the Day Software Puts the Universe in a Shoebox... How It Will Happen and What It Will Mean. UK: Oxford University Press, 1993, ISBN 978-01-95079-06-7.

[3] GRIEVES, Michael. Origins of the Digital Twin Concept. Online, PDF. 2016. Dostupné z: https://doi.org/10.13140/RG.2.2.26367.61609. [cit.2023-12-30].

[4] MAULSHREE, Singh; FUENMAYOR, Evert; HINCHY P., Eoin; QIAO, Yuansong; MURRAY, Niall et al. Digital Twin: Origin to Future. Online, PDF. Applied System Innovation. 2021, vol. 4, no. 2: 36. ISSN 2571-5577. Dostupné z: https://doi.org/10.3390/asi4020036. [cit.2023-12-30].

[5] KRITZINGER, Werner; KARNER, Matthias; TRAAR, Georg; HENJES, Jan a SIHN, Wilfried. Digital Twin in manufacturing: A categorical literature review and classification. Online, PDF. IFAC-PapersOnLine. 2018, vol. 51, no, 11, s. 1016-1022. ISSN 2405-8963. Dostupné z: https://doi.org/10.1016/j.ifacol.2018.08.474. [cit.2023-12-30].

[6] VOGEL-HEUSER, Birgit a HESS, Dieter. Guest Editorial Industry 4.0–Prerequisites and Visions. Online, PDF. IEEE Transactions on Automation Science and Engineering. 2016, vol. 13, no. 2, s. 411-413. Dostupné z: https://doi.org/10.1109/TASE.2016.2523639. [cit.2023-12-30].

[7] XU, Xun; LU, Yuqian; VOGEL-HEUSER, Birgit a WANG, Lihui. Industry 4.0 and Industry 5.0—Inception, conception and perception. Online, PDF. Journal of Manufacturing Systems. 2021, vol. 61, s. 530-535. ISSN 0278-6125. Dostupné z: https://doi.org/10.1016/j.jmsy.2021.10.006. [cit.2023-12-30].

[8] BREQUE, Maija; DE NUL, Lars a PETRIDIS, Athanasios. Industry 5.0 Towards a sustainable, humancentric and resilient European industry. Online, PDF. European Commission, Directorate-General for Research and Innovation. 2021. Dostupné z: https://doi.org/10.2777/308407. [cit.2023-01-03].

[9] HUANG, Sihan; WANG Baicun; LI Xingyu; ZHENG, Pai; Mourtzis, Dimitris et al. Industry 5.0 and Society 5.0—Comparison, complementation and co-evolution. Online, PDF. Journal of Manufacturing Systems. 2022, vol. 64, s. 424-428. Dostupné z: https://doi.org/10.1016/j.jmsy.2022.07.010. [cit.2023-01-03].

[10] LENG, Jiewu; SHA, Weinan; WANG, Baicun; ZHENG, Pai; ZHUANG, Cunbo et al. Industry 5.0: Prospect and retrospect. Online, PDF. Journal of Manufacturing Systems. 2022, vol. 65, s. 279-295. ISSN 0278-6125. Dostupné z: https://doi.org/10.1016/j.jmsy.2022.09.017. [cit.2023-12-30].

[11] GRÄßLER, Iris a PÖHLER, Alexander. Intelligent Devices in a Decentralized Production System Concept. 11th CIRP Conference on Intelligent Computation in Manufacturing Engineering. Online, PDF. Procedia CIRP. 2018, vol. 67, s. 116-121. ISSN 2212-8271. Dostupné z: https://doi.org/10.1016/j.procir.2017.12.186. [cit.2024-04-20].

[12] MANU, Suvarna; KEN, SHAUN Yap; WENTAO, Yang; JUN, Li; YEN, TING Ng et al. Cyber–Physical Production Systems for Data-Driven, Decentralized, and Secure Manufacturing—A Perspective. Online, PDF. Engineering. 2021, vol. 7, s. 1212-1223. ISSN 2095-8099, Dostupné z: https://doi.org/10.1016/j.eng.2021.04.021. [cit.2024-04-20].

[13] SCHMIDTKE, Niels; RETTMANN, Alina a BEHRENDT, Fabian. Matrix Production Systems - Requirements and Influences on Logistics Planning for Decentralized Production Structures. Proceedings of the 54th Hawaii International Conference on System Sciences. Online, PDF. HICSS. 2021, s. 1665 - 1674. ISBN 978-0-9981331-4-0 Dostupné z: http://hdl.handle.net/10125/70813. [cit.2024-04-21].

[14] VAN BRUSSEL, Hendrik. Holonic Manufacturing Systems. CIRP Encyclopedia of Production Engineering. Online, PDF. Springer Berlin Heidelberg. 2014, s. 654-659. ISBN 978-3-642-20617-7. Dostupné z: https://doi.org/10.1007/978-3-642-20617-7_6556. [cit.2024-04-21].

[15] SHOHAM, Yoav a LEYTON-BROWN, Kevin. MULTIAGENT SYSTEMS Algorithmic, Game-Theoretic, and Logical Foundations. Online, PDF. Cambridge University Press. 2009. Revize 1.1. Dostupné z: http://www.masfoundations.org/. [cit.2024-04-21].

[16] GIRET, Adriana; BOTTI, Vicente a VALERO, Soledad. MAS methodology for HMS. Lecture Notes in Computer Science. Online, PDF. Springer, Berlin,

Heidelberg. 2005, vol. 3593, s. 39 - 49. ISBN 978-3-540-31831-6. Dostupné z: https://doi.org/10.1007/11537847_4. [cit.2024-04-21].

[17] ADOLPHS, Peter; AUER, Sören; BEDENBENDER, Heinz; BILLMANN, Meik; HANKEL, Martin; REXROTH, Bosch. The Structure of the Administration Shell: Trilateral perspectives from France, Italy and Germany. Online, PDF. 2018. Dostupné z: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/hm-2018-trilaterale-coop.pdf?__blob=publicationFile&v=1. [cit.2024-01-03].

[18] GARRELS, Kai; JOCHEM, Michael a JÄNICKE, Lutz. The AAS Dataspace for Everybody: An Architecture Example for a Simple Dataspace... Online, PDF. 2023. Dostupné z: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/aas-dataspace4everybody.pdf?__blob=publicationFile&v=6. [cit.2024-01-03].

[19] GRÜNER, Sten; NEIDIG, Jörg; ORZELSKI, Andreas a POLLMEIER, Stefan. Asset Administration Shell Reading Guide. Online, PDF. Industrial Digital Twin Association. Verze 2022-11-03. Dostupné z: https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2022/12/2022-12-07_IDTA_AAS-Reading-Guide.pdf. [cit.2024-01-02].

[20] BADER, Sebastian; BARNSTEDT, Erich; BEDENBENDER, Heinz; BERRES, Bernd, BILLMANN Meik et al. Details of the Asset Administration Shell. Part 1 - The exchange of information between partners in the value chain of Industrie 4.0 (Version 3.0RC02). Online, PDF. Federal Ministry for Economic Affairs and Climate Action (BMWK). 2022. Dostupné z: https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2022/06/DetailsOfTheAssetAdministrationShell_Part1_V3.0RC02_Final1.pdf. [cit.2024-01-02].

[21] Specification of the Asset Administration Shell Part 5: Package File Format (AASX). Online, PDF. 2023. IDTA 01005-3-0. Dostupné z: https://industrialdigitaltwin.org/wp-content/uploads/2023/04/IDTA-01005-3-0_SpecificationAssetAdministrationShell_Part5_AASXPackageFileFormat.pdf. [cit.2024-01-02].

[22] BADER, Sebastian a MALESHKOVA, Maria. The semantic asset administration shell. Semantic Systems. The Power of AI and Knowledge Graphs. 15th International Conference, SEMANTiCS 2019. Online, PDF. Springer International Publishing. 2019, s. 159-174. ISBN 978-3-030-33219-8. Dostupné z: https://doi.org/10.1007/978-3-030-33220-4_12. [cit.2024-04-11].

[23] BELYAEV, Alexander a DIEDRICH, Christian. Specification "Demonstrator I4.0-Language"v3.0. Online, PDF. 2019. Dostupné z: https://www.researchgate.net/publication/334429449_Specification_Demonstrator_I40-Language_v30. [cit.2024-04-04].

[24] DÖNICKE, Nicole; GAMER, Thomas; HEER, Tobias; JÄNICKE, Lutz; JOCHEM, Michael et al. Security der Verwaltungsschale. Online, PDF. Plattform Industrie 4.0, ZWEI. 2017. Dostupné z: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/security-der-verwaltungsschale.pdf?__blob=publicationFile&v=6. [cit. 2024-04-06].

[25] HABIGER, Pascal; HILDEBRANDT, Gary; DRATH, Rainer; BARTH, Mike; FAY, Alexander et al. Plug & Produce im real-virtuellen Kontext fertigungstechnischer heterogener Anlagen. Steuerungsarchitektur und Virtuelle Inbetriebnahme Online, PDF. 18. AALE-Konferenz. Pforzheim. 2022. ISBN 978-3-910103-00-9. Dostupné z: https://doi.org/10.33968/2022.03. [cit.2024-04-13].

[26] OPC UA methods for the SIMATIC S7-1500 OPC UA server. Online, PDF. Dostupné z: https://support.industry.siemens.com/cs/attachments/109756885/109756885_OpcUa_ServerMethods_DOC_V1_2_en.pdf. [cit. 2023-12-16].

Seznam symbolů a zkratek

AAS Asset Administraton Shell

UaExpert Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V.

BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V.

BP Bakalářská Práce

CA Certificate Authority

CCPS Cyber-Physical Production Systems

CPU Central Processing Unit

DP Diplomová Práce

DT Digital Twin - Digitální dvojče

FW Future Work

HMS Holonic Manufacturing System

I4.0 Industry 4.0 - Průmysl 4.0

IDTA International Digital Twin Asociation

MAS Multi Agent System

NASA National Aeronautics and Space Administration

OPC UA Open Platform Communications United Architecture

PLC Programmable Logic Controller - Programovatelný Logický Automat

PLM Product Lifecycle Management

SiOME Siemens OPC UA Modeling Editor

UaExpert Unified Automation Expert

VDI Verein Deutscher Ingenieure e.V

VDMA Verband Deutscher Maschinen und Anlagenbau e.V.

ZVEI Zentralverband Elektrotechnik und Elektronikindustrie e.V.

Obsah přiloženého paměťového média

- `root` — kořenový adresář přiloženého archivu
  - `text`
    - `implementaceAssetAdministrationShell.pdf` — text práce
  - `programy`
    - `source` — zdrojové kódy
      - `implementaceAssetAdministrationShell.zap17` — archív TIA portalu.
      - `ordersGUI.exe` — spustitelné grafické rozhraní AAS client hubu
      - `ordersGUI.ico` — ikona aplikace
      - `ordersGUI.py` — grafické rozhraní AAS client hubu
      - `plcServiceProvider.xml` — OPC UA prostředí pro PLC
      - `serviceRequester.exe` — spustitelný program AAS client hubu
      - `serviceRequester.ico` — ikona aplikace
      - `serviceRequester.py` — program AAS client hubu
    - `testing` — složka pro soubory z testování
      - `man.log` — log zařízení typu Manipulator
      - `mix.log` — log zařízení typu Manufacturing_unit
      - `orders.db` — inicializační databáze před testováním
      - `plcList.xlsx` — excel tabulka testování PLC
      - `spUsages.mat` — zpracované data z testování
      - `srState.log` — log stavů žadatelů služeb
      - `sto.log` — log zařízení typu Storage