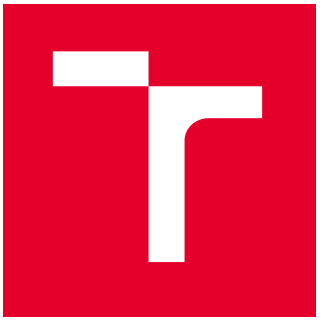


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

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

NÁVRH A SIMULACE AAS ALGORITMŮ PRO DISTRIBUOVANÉ ŘÍZENÍ VÝROBY

INDUSTRY 4.0 ASSET ADMINISTRATION SHELL DESIGN

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Prokop Tkadlec

AUTHOR

VEDOUCÍ PRÁCE — Ing. Václav Kaczmarczyk, Ph.D.

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. Prokop Tkadlec ***ID:*** 205846

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

NÁZEV TÉMATU:

Návrh a simulace AAS algoritmů pro distribuované řízení výroby

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s konceptem AAS - Asset Administration Shell pro výrobní systémy.
2. Navrhněte vhodný AAS model pro výrobní zařízení a inteligentní výrobek.
3. Proveďte rešerši softwarových simulátorů diskrétních událostí se zaměřením na open source simulátor Omnet++.
4. S využitím vhodného simulátoru vytvořte simulovanou síť zařízení s AAS datovými modely, která bude odpovídat síti Testbedu I4.0 v laboratoři.
5. Navrhněte sérii testů a proveďte tyto testy.
6. Celé řešení zhodnoťte a zadokumentujte.

DOPORUČENÁ LITERATURA:

Platform Industrie 4.0: Usage View of AssetAdministration Shell

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

Vedoucí práce: Ing. Václav Kaczmarczyk, Ph.D.

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

ABSTRAKT

Cílem této práce je vytvoření simulačního modelu, který bude odpovídat síti reálného TestBedu Barman, za pomocí zařízení s AAS datovými modely. Simulační model bude vytvořen za pomocí simulátoru OMNeT++. V práci jsou popsány pojmy průmyslu 4.0 hlavně pak AAS a jednotlivé buňky TesBedu Barman. Dále jsou popsány vytvořené AAS modely výrobních buněk a inteligentního výrobku pomocí programu AASX Package Explorer. V praktické části je popsán vytvořený simulační model TestBedu a jeho jednotlivé moduly, které byly vytvořeny. Pro simulační model jsou pak popsány dva jednoduché simulační scénáře společně s monými proměnnými ke sledování.

KLÍČOVÁ SLOVA

Průmysl 4.0, Asset, Administration Shell, AAS, TestBed Barman, OMNeT++, simulace, DES

ABSTRACT

The aim of this work is to create a simulation model that will correspond to the real TestBed Barman network, with the help of AAS data model devices. The simulation model will be created using the OMNeT++ simulator. The thesis describes the concepts of Industry 4.0, especially AAS and the individual cells of TesBed Barman. Afterwards the AAS models of production cells and intelligent products created by using the AASX Package Explorer program are described. The practical part describes the created TestBed simulation model and its individual modules that were created. Two simple simulation scenarios are then described for the simulation model together with the variables to be monitored.

KEYWORDS

Industry 4.0, Asset, Administration Shell, AAS, TestBed Barman, OMNeT++, simulation, DES,

TKADLEC, Prokop. *Návrh a simulace AAS algoritmů pro distribuované řízení výroby*. Diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2024. Vedoucí práce: Ing. Václav Kaczmarczyk, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Prokop Tkadlec
VUT ID autora:	205846
Typ práce:	Diplomová práce
Akademický rok:	2023/24
Téma závěrečné práce:	Návrh a simulace AAS algoritmů pro distribuované řízení výroby

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. Václavu Kaczmarczykovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Téma této práce se zabývá konceptem Průmyslu 4.0. Tato další revoluce v průmyslu se z rychlých výroben zaměřuje na kvalitnější, efektivnější a chytrou výrobu. Jednotlivé firmy se této premisy snaží dosáhnout pomocí stále komplexnější automatizace, která nepotřebuje přílišný zásah člověka. Snahou Průmyslu 4.0 je také decentralizovat výrobní systémy a tím přenést rozhodování na nižší úrovně řízení. Virtualizace pomocí digitálního dvojčete, která začala již během předchozí revoluce, je nyní nahrazena tzv. AAS - Asset Administration Shell ta nabízí větší možnosti komunikace, je pevně definována a umožňuje samostatně se rozhodovat. Tímto konceptem se také tato práce zabývá.

Cílem první části práce je seznámení se s konceptem AAS a následné vytvoření vhodných AAS modelů pro výrobní zařízení a inteligentní výrobek, které budou vhodné pro potřeby řízení výroby na projektu TestBed Barman, který je demonstrátorem konceptu Průmyslu 4.0. Druhá část práce si klade za cíl vytvoření simulační sítě, která bude odpovídat síti TesBedu.

V první kapitole je čtenář uveden do problematiky čtvrté průmyslové revoluce. Zároveň je seznámen s pojmem AAS a jeho základní strukturou. Dále je popsána komunikace mezi AAS a komunikace mezi assetem a AAS. Druhá kapitola se zabývá vytvářením vhodných AAS modelů pro jednotlivá výrobní zařízení a výrobku. Ve třetí kapitole je stručný popis samotného TestBedu Barman. Čtenář je seznámen s jednotlivými procesními buňkami, jejich účelem a funkcí. Čtvrtá kapitola je zaměřená na rešerši trhu se softwarovými simulátory diskrétních událostí, se zaměřením na program OMNeT++, který je podrobněji popsán. Další kapitola se zaměřuje na funkčnost jednotlivých modulů, které byly za účelem simulace vytvořeny a dalších úprav potřebných pro chod simulace. Závěr práce popisuje vytvořené simulační scénáře pro testování modelu a možné podněty pro zlepšení práce.

1 Průmysl 4.0

První myšlenky Průmyslu 4.0 byly zformovány a prezentovány na veletrhu v Hannoveru v roce 2011. Oficiální zformování a oznámení pak nastalo o dva roky později na veletrhu ve stejné městě. Základním principem každé průmyslové revoluce bylo, je a bude zjednodušení a automatizování jednoduché a monotónní práce. To má ve svém důsledku má za následek zrychlení a zkvalitnění výrobního procesu a tím i výrobku samého. Na druhou stranu také i ke zrušení méně kvalifikovaných pracovních pozic a jejich nahrazení těmi kvalifikovanějšími[11].

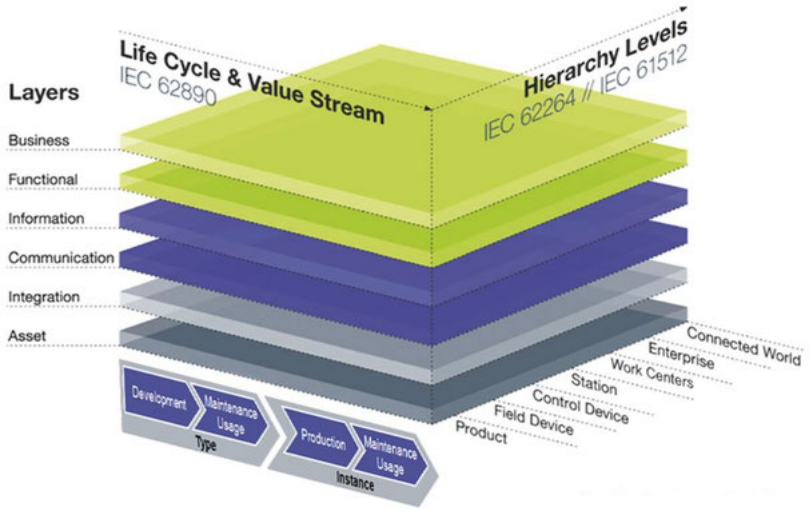
Myšlenkou je vznik takzvaných „Chytrých továren“, které jsou schopné automatizované výroby aniž by do ní zasáhl operátor. Tyto továrny se však v přesném slova smyslu zatím nevyskytují. Pokrok v oblastech mikročipů, rychlosti komunikace a možnosti ukládání dat. Dal vzniknout inteligentním prvkům ve výrobě jako například inteligentní snímače. Díky tomuto jsou formovány sítě jako Internet věcí, Internet služeb a Internet lidí, které umožňují propojení veškerých jednotlivých entit a decentralizovaný přístup v rámci celé firmy bez ohledu místo závodu nebo hranice státu[11].

Všechny vyvíjené nebo již vytvořené komponenty v rámci průmyslu 4.0, splňují obecně všechny potřebné principy. Níže jsou zjednodušeně popsány všechny základní principy průmyslu 4.0:

- Interoperabilita – schopnost kyberneticko-fyzikálních systémů, lidí, výrobků a všech systémů vzájemné komunikace
- Virtualizace – propojení a zobrazení fyzických systémů s digitálními modely
- Decentralizace – rozložení rozhodování výroby do jednotlivých operačních modulů
- Práce v reálném čase – dodržení požadavku na přenos dat, rozhodování a řízení v systémech reálného času
- Orientace na služby – preference na nabízení a využívání standardních služeb
- Modularita a rekonfigurabilita systému – možnost systému jednoduchého rozložení a složení v požadovaném tvaru a jeho možná rekonfigurace v případě potřeby přizpůsobit se aktuální situaci

1.1 Referenční Architektonický Model průmyslu 4.0

Zkráceně RAMI 4.0, patří mezi základní modely zobrazující architekturu průmyslu 4.0. Funkcí tohoto modelu je poskytnout standardizovaný rámec, který následně může zjednodušit integraci a porozumění jednotlivých standardů a příkladů použití[1]. Model se skládá ze tří os viz. Z pohledu, který vidíme na obrázku 1.1. Máme na ose



Obr. 1.1: Referenční architektonický model průmyslu 4.0 [1].

X životní cyklus výrobku, v první části je označen jako *Type*, kdy je produkt ve fázi návrhu a testování prototypů, tedy z větší části digitální. Druhá fáze označená jako *Instance* začíná jakmile produkt přejde na sériovou výrobu, pokračuje jeho údržbou a končí ukončením podpory a služeb. Osa Z znázorňuje hierarchickou škálu, která zobrazuje jednotlivé úrovně řízení tak, jak je známe z průmyslu 3.0. Poslední osa Y zobrazuje definované vrstvy, které jsou z pohledu správné funkčnosti pro Průmysl 4.0 nutné[1].

1.2 Asset

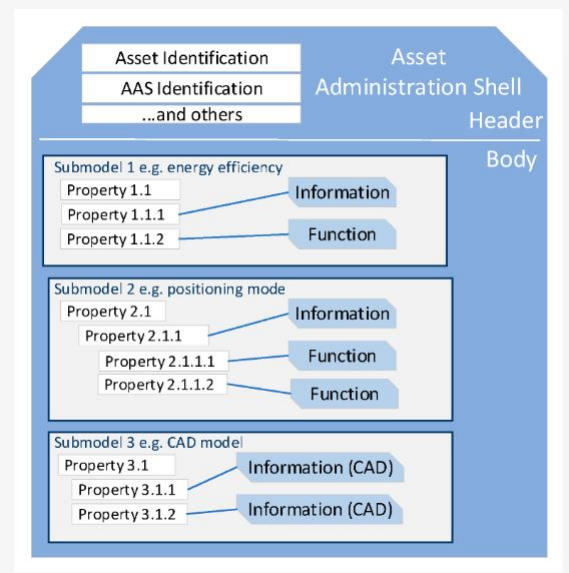
Pojem Asset můžeme do češtiny přeložit jako Aktivum. Pod tento pojem můžeme zařadit všechny fyzické i digitální prvky, které mají nebo budou mít pro daný podnik, společnost určitou hodnotu. Takovéto prvky můžou být licence, dokumentace, výrobní stroje, polotovary nebo lidský operátor. Snahou Průmyslu 4.0 je mít data o těchto assetech uložena jednotně a zároveň možnost komunikace a předávání dat mezi jedním či více assety[2].

1.3 Asset Administration Shell

Asset Administration Shell neboli zkráceně AAS je standardizovaná digitální reprezentace Assetu. Je definována podle pravidel tak, aby nebyla závislá na typu systému, objektu nebo výrobci. Tímto AAS pak můžeme popsat cokoliv od výrobku, jednoduchého nástroje, výrobní buňky až po továrnu. Kromě pasivního zobrazování

a ukládání informací o assetu, může interagovat s prostředím decentralizovaného systému tedy ostatními AAS[5].

AS se skládá z hlavičky(„Header“) a těla („Body“). Hlavička obsahuje unikátní identifikátor assetu, který obsahuje a identifikátor samotného AAS. V samotném tělu pak nalezneme submodely, které obsahují jednotlivé datové struktury a data samotná. Tyto data mohou nést informace o zařízení, jeho parametry nebo jiné nutné či jinak důležité data procesu výroby[5].



Obr. 1.2: Ukázka vnitřní struktury Asset Administration Shell (AAS) [6].

1.3.1 Pasivní AAS

Nutný základ pro AAS je jeho pasivní část, ta obsahuje veškerá informační data. Tyto data můžeme rozdělit na statická a dynamická. Statická data se s cyklem života nemění například sériové číslo, pevně dané parametry a statické vlastnosti nebo také metamodely a submodely. Opakem jsou data dynamická, tedy v čase se měnící, to mohou být například naměřené veličiny, doba provozu, opotřebování nástroje nebo momentální stav. S těmito daty následně může manipulovat aktivní část AAS a sdílet je s ostatními[8, 9].

1.3.2 Aktivní AAS

Aktivní část AAS je ta část, která je schopná komunikovat a interagovat s pasivní částí, ale zároveň i s ostatními AAS. Komunikace probíhá na základě funkcí a metod obsažených v AAS[8, 9].

1.4 Komunikace I4.0

Jedním ze základů Průmyslu 4.0 je práce v reálném čase, a proto se v komunikaci klade důraz na rychlost a robustnost. V závislosti na typu rozlišujeme komunikaci AAS s assetem a komunikaci mezi AAS. V následujících kapitolách budou oba typy popsány.

1.4.1 Komunikace AAS a fyzickým objektem

Při komunikaci AAS s fyzickým objektem je nutné vycházet z dostupnosti rozhraní a protokolu, které je možné na samotném objektu realizovat. V případě, že není možné realizovat AS přímo na assetu, je žádoucí vytvořit komunikační propojení mezi objektem a hardwarem, které dané AS obsahuje. Tento přístup bývá nutný u senzorů(nebo jejich skupin), embedded systémů nebo například u člověka, pro kterého je nutné volit specifický přístup.
Tyto komunikace jsou prováděny na pozadí a AAS k nim nemá přístup tudíž ani další AAS k ní nemají přímý přístup.

1.4.2 Komunikace mezi AAS

Při komunikaci mezi AAS se můžeme nejčastěji setkat se dvěma používanými protokoly a to OPC UA a MQ Telemetry Transport (MQTT). Oba protokoly budou v nadcházejících podkapitolách stručně popsány společně s výhodami a nevýhodami.

MQTT

Message Quering Telemetry Transport je síťový protokol fungující na principu Publisher-Subscriber. Veškerá komunikace probíhá přes centrální prvek tzv. *broker*, který každou zprávu filtruje a přeposílá na další zařízení, kterým je zpráva určená. Každé zařízení v této síti může sdílet a také odesílat neomezené množství zpráv. Tento způsob komunikace se často vyskytuje v IoT sítích a těch o malém počtu prvků[2]. Díky nutnosti centrální prvku který přeposílá jednotlivé zprávy dochází v sítích o velkém počtu prvků nebo velkém množství zpráv. Dochází k problémům s vyjednáváním služeb, zahlcení sítě a velké časové odezvy. Tento problém lze zmírnit přidáním většího počtu *brokerů*, avšak nedá se úplně vyřešit.

OPC UA

Open Platform Communications – Unified Architecture zkráceně OPC UA je otevřený komunikační standart pro průmyslové využití. V průmyslu má velkou šířku využití od vestavěných systémů po výrobní informační systémy (MES) nebo systémy

pro plánování podnikových zdrojů (ERP). OPC UA pracuje na principu klient-server, tím odpadá nutnost centrálního prvku, který by zajišťoval přenos zpráv. Absence centrálního prvku přispívá k myšlence Průmyslu 4.0 v ohledu decentralizace. V sítích o velkém počtu prvků se díky velkému množství přístupů začne prjevovat vyšší latence zpráv. Tuto nevýhodu lze redukovat použitím asistenčních prvků jako jsou místní vyhledávací servery (Local Discovery Server: LDS) nebo vícesměrovým rozesíláním zpráv (Multicast Extension: ME)[6].
V nové verzi přináší OPC UA formát komunikace Publisher-Subscriber. Takto fungující komunikace může nebo nemusí obsahovat centrální prvek. V případě použití centrálního prvku je postup komunikace podobný jako v případě MQTT. *Broker* přijatou zprávu odesílá všem, kteří ji odebírají. Pokud se centrální prvek nepoužívá je tato funkce nahrazena síťovým zařízením například switchem a vhodným protokolem. Výhodou je pak snížení latence.

1.4.3 Jazyk I4.0

Jazyk průmyslu 4.0 je definován normou VDI/VDE 2193. Tato norma se skládá ze dvou celků, její první část definuje slovník jazyku I4.0 a strukturu zpráv. Díky čemuž nedochází ke zdvojení informací a nečitelnosti zpráv. Druhá část normy definuje protokol sémantické interakce, který stanovuje způsob vyjednávaní mezi poskytovatelem služeb a žadatelem[7].

2 AAS model výrobního zařízení a výrobku

V této kapitole bude vysvětlena tvorba vhodného zjednodušeného AAS modelu pro výrobní zařízení a inteligentní výrobek. Vytvořené modely budou následně popsány.

2.1 Použitý nástroj pro tvorbu AAS

Pro vytváření AAS byl použit nástroj AASX Package Explorer. Tento program lze použít jak pro prohlížení již vytvořených AAS, tak pro vytváření nových. Mezi další výhody tohoto programu patří možnost z již vytvořeného AAS vytvořit například OPC UA server. To lze provést i s dalšími standarty, které program podporuje těmi mohou být MQTT nebo REST. Pomocí programu lze taktéž vyexportovat nebo importovat AAS z/do různých formátu, které jsou obecně podporovány dalšími programy pracujícími s AAS jako např. JSON nebo UML.

2.1.1 Vytváření AAS, submodelů a datových struktur

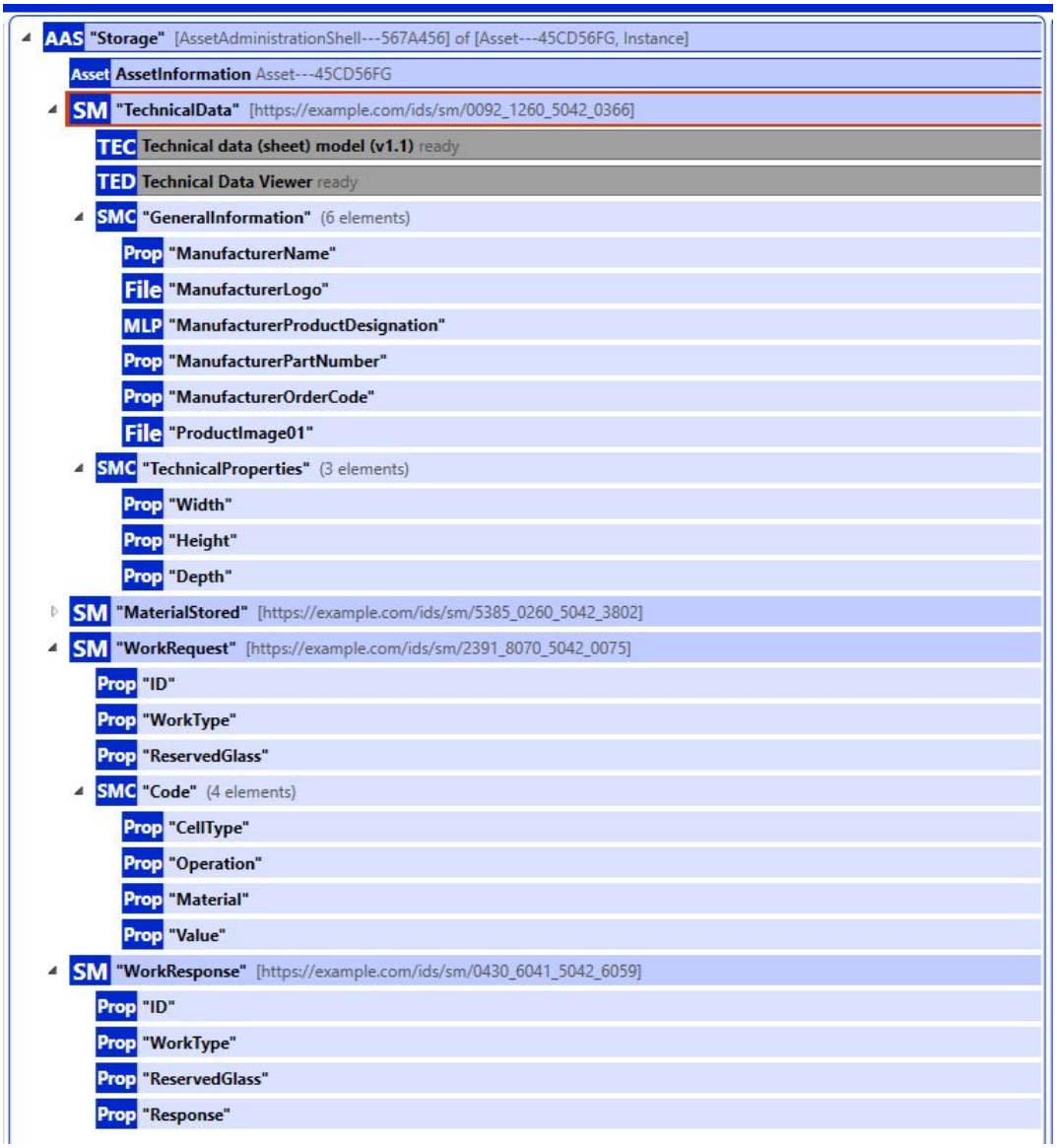
Samotné vytváření AAS v programu je jednoduché. Prvním krokem je uvedení celého programu do editovacího módu povolením *Edit* v menu *Workspace*. Následně je již možné vytvořit nové AAS čisté nebo z existujícího templatu. Ve vytvořených AAS je již možné vytvářet jednotlivé submodely čisté, které si uživatel definuje úplně sám. Nebo podobně jako u AAS může vytvořit template již obsahují předem pojmenované a rozvržené datové typy, do kterých lze vepsat hodnoty nebo je dále upravit.

2.2 Vytvořené modely AAS

Byly vytvořeny AAS modely pro jednotlivé buňky Sklad, Robot, Dopravník, Výrobní buňku a Skleničku. Protože se některé submodely opakují ve více modelech AAS budou teď popsány jednotlivé submodely.

2.2.1 Sumbodel TechnicalData

V tomto submodelu nalezneme dvě proměnné typu SubmodelElementCollection. První z nich GeneralInformation obsahuje informace o výrobci jako je název výrobce, logo, číslo součástky nebo obrázek produktu. Druhý collection TechnicalProperties obsahuje technické informace vztahující se k výrobku, proto každé AAS může obsahovat jiné proměnné, jiný počet a hodnoty. Patrný rozdíl můžeme vidět například u obrázků 2.3 a 2.4.



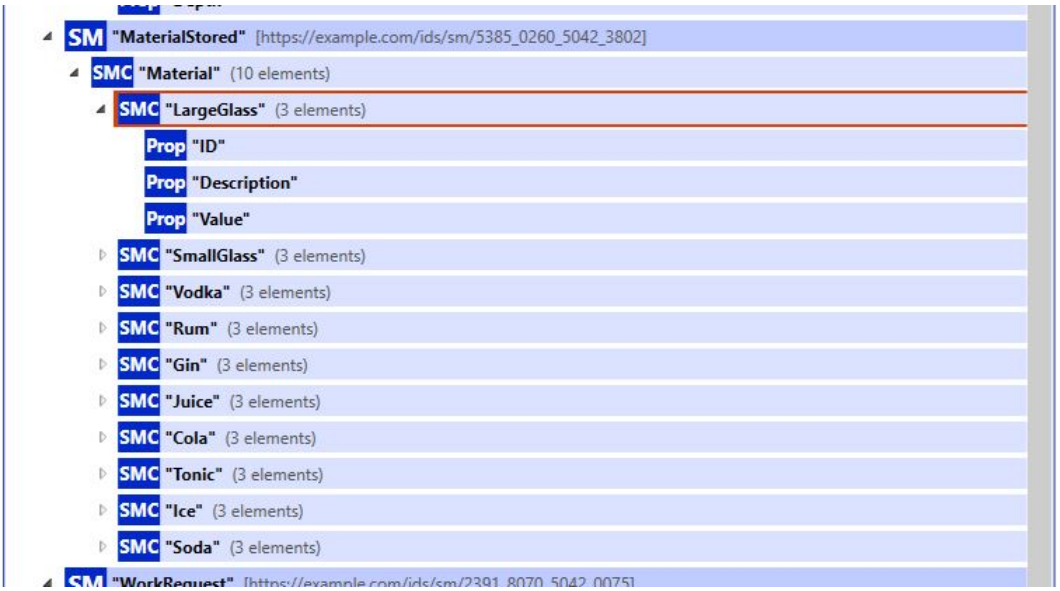
Obr. 2.1: Vytvořený Asset Administration Shell pro skladovací buňku.

2.2.2 Submodel MaterialStored

Tento submodel se nachází pouze v AAS Skladu a obsahuje informace o uloženém materiálu. Každý materiál má zadefinované ID, popis a uskladněné množství. Příklad můžeme vidět na obr.2.2

2.2.3 Submodel WorkRequest

WorkRequest obsahuje informace o žádané službě. Nalezneme zde ID práce, typ práce, číslo skleničky, která o službu zažádala a čtyřmístný kód, který určuje typ



Obr. 2.2: Vytvořený submodel obsahující informace o uloženém materiálu.

buňky, operaci, materiál a jeho množství. Jedná se o proměnou, která je využita i v simulaci a bude ještě podrobněji popsána.

2.2.4 Submodel WorkResponse

Submodel WorkResponse obsahuje prvky, které popisují vykonanou práci. Jako je typ vykonané práce, název skleničky která službu požadovala a její výsledek.

2.2.5 Submodel Recipe

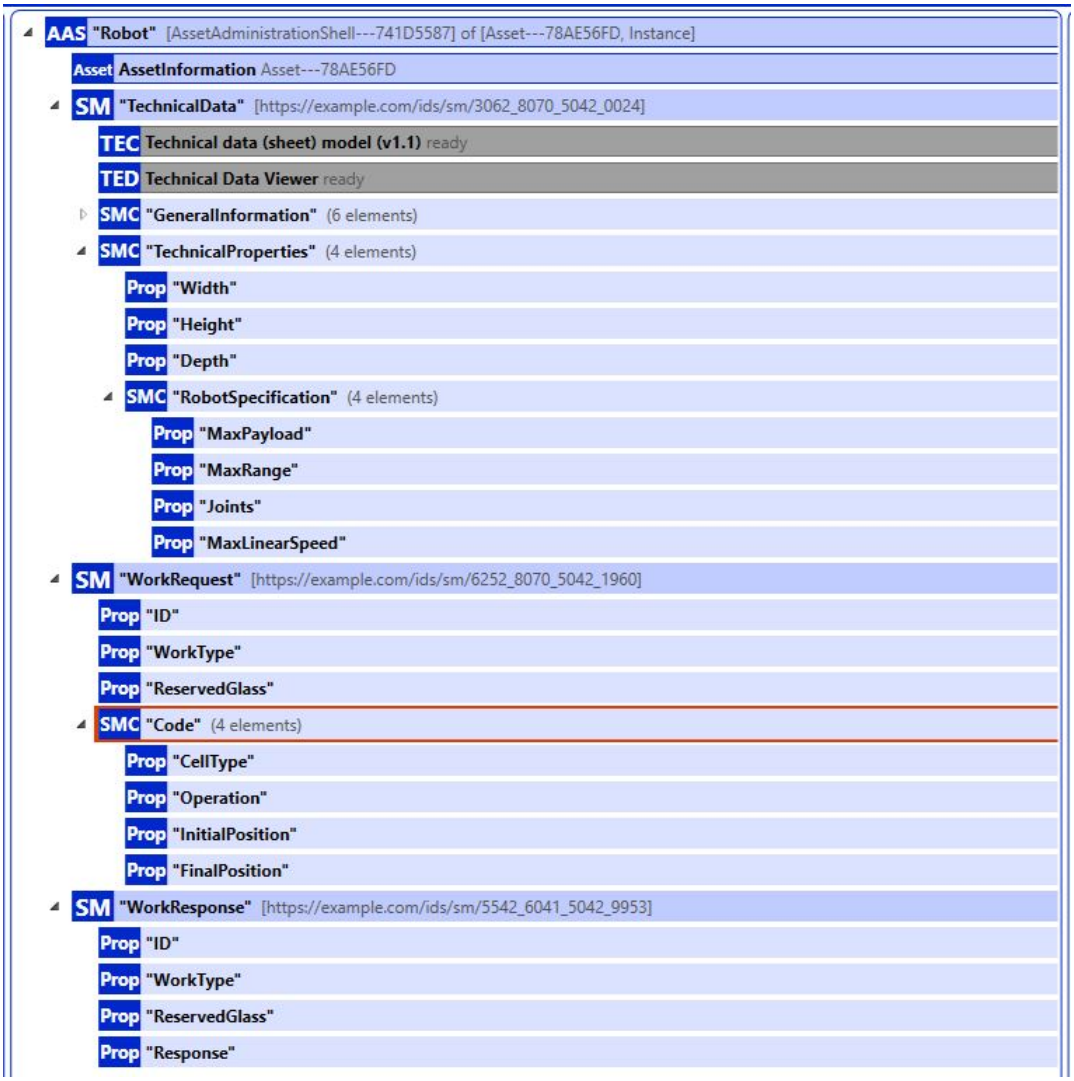
Tento submodel lze nalézt pouze v AAS Skleničky. Nalezneme zde informace o receptu, jeho identifikační číslo, popis, cenu a informace o krocích receptury.

2.2.6 Submodel Statistics

Další submodel, který se nachází v AAS Skleničky. Zaznamenává statisticky významné veličiny, které mohou ukázat na nedostatky nebo mezery ve výrobě nebo efektivitě.

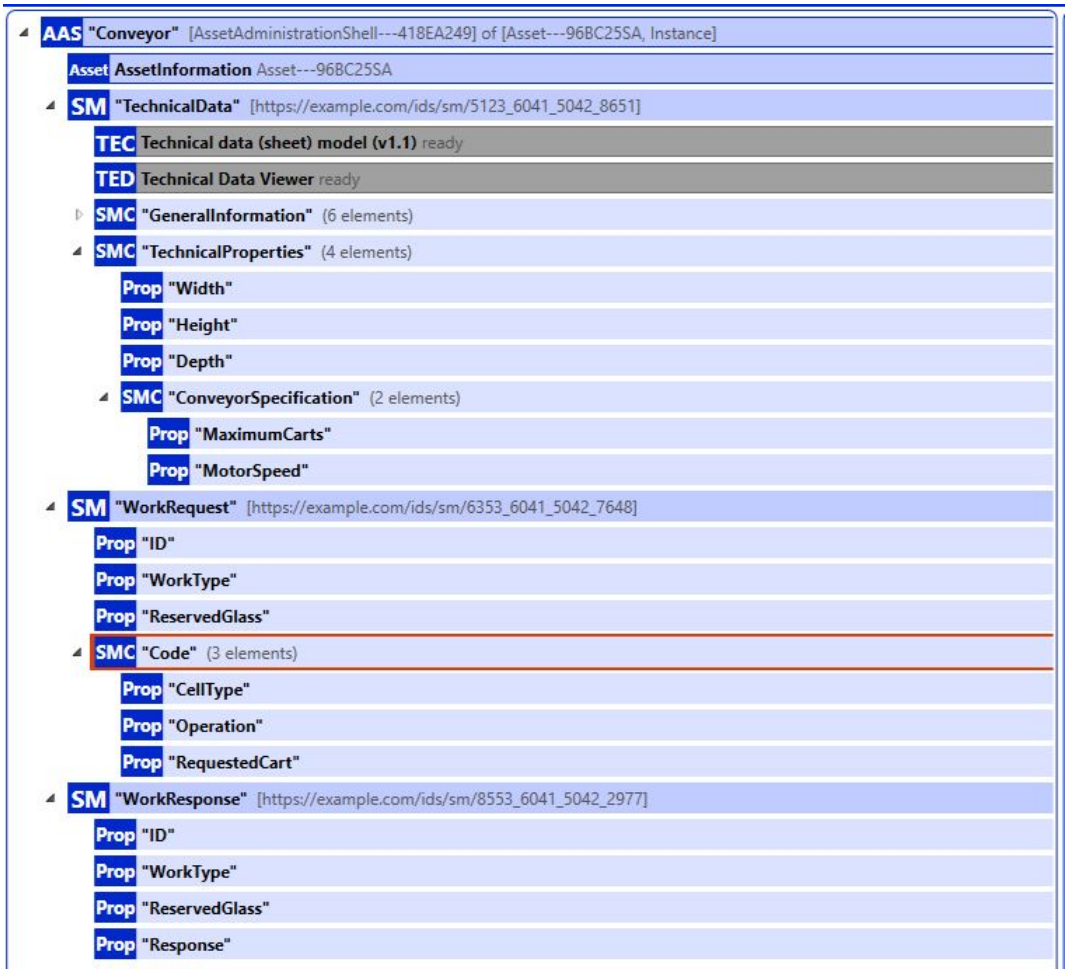
2.3 Model Barmana

Kromě výrobních buněk a skleničky bylo vytvořeno i AAS pro TestBed. Oproti výše zmíněným využívá jiné submodely, které si teď popíšeme. Submodel Nameplate

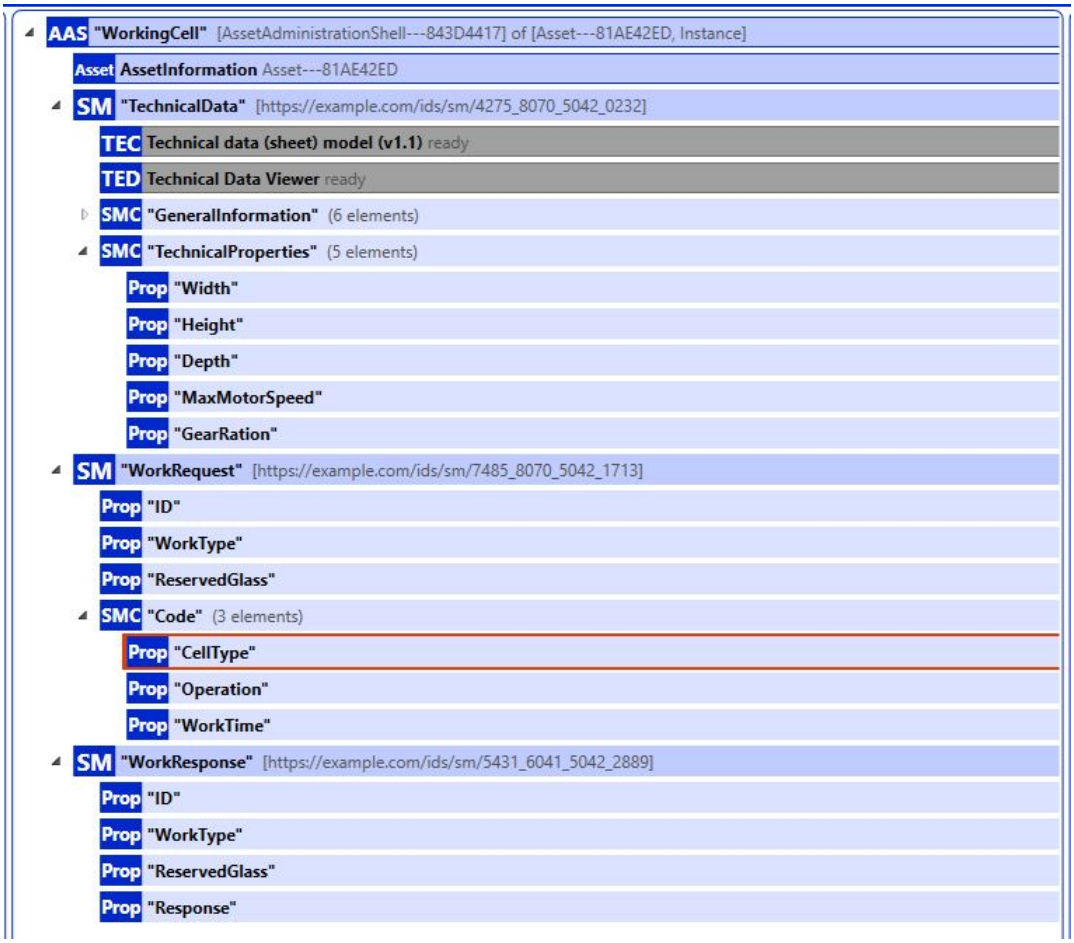


Obr. 2.3: Vytvořený Asset Administration Shell pro transportní buňku.

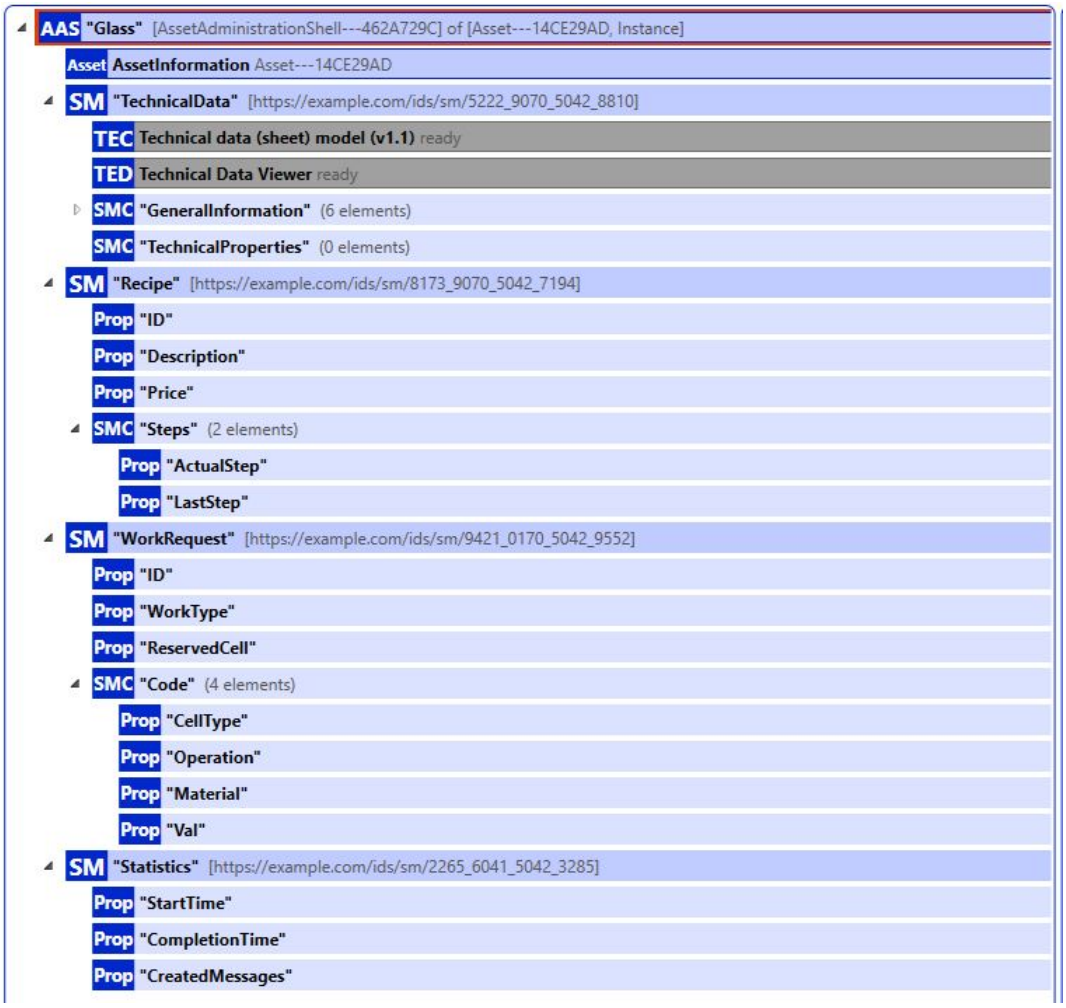
ukládá informace o výrobci, jeho adrese nebo také rok výroby a stát ve kterém byl Asset vytvořen. Druhým submodelem je BillOfMaterial ten graficky umožňuje ukázat vazby mezi vytvořenými entitami a Assety. Toho je docíleno pomocí entit kde se vkládá AAS a následně pomocí datového typu Relationship je vytvořena vazba, takto vytvořené vazby můžeme vidět na obrázku **??**. Posledním submodelem je BarmanAASRelationship zde jsou vytvořeny vazby mezi AAS.



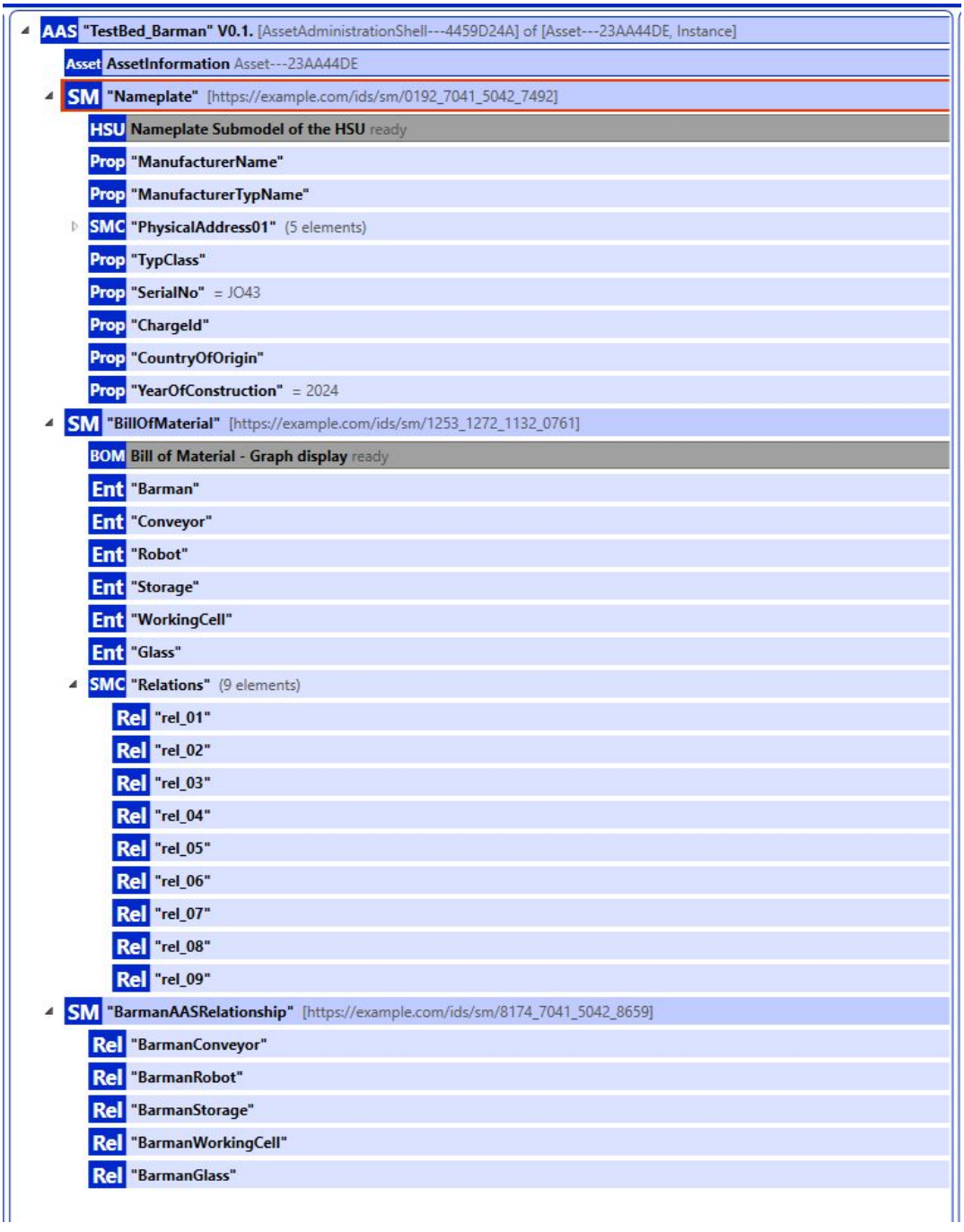
Obr. 2.4: Vytvořený Asset Administration Shell pro dopravník.



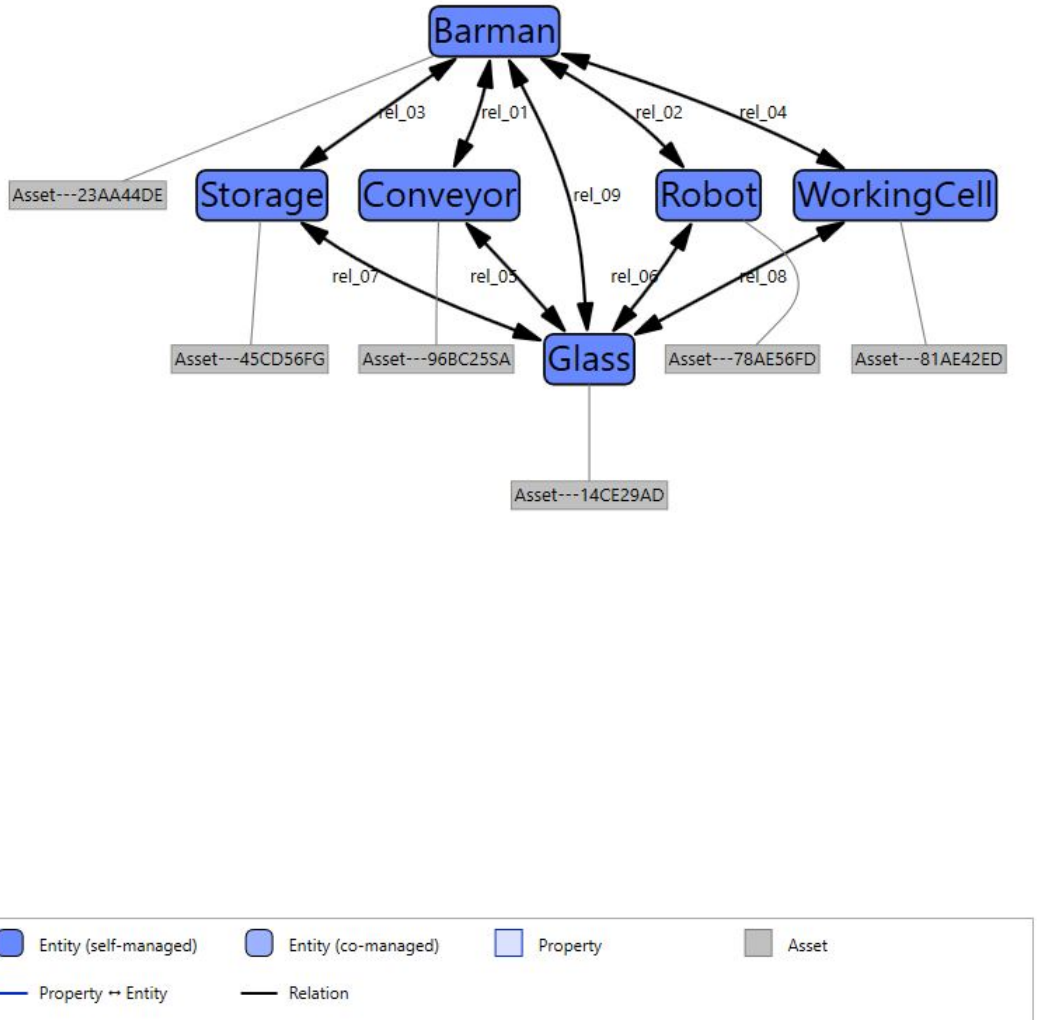
Obr. 2.5: Vytvořený Asset Administration Shell pro výrobní buňku.



Obr. 2.6: Vytvořený Asset Administration Shell pro skleničku.



Obr. 2.7: Vytvořený Asset Administration Shell pro TestBed.



Obr. 2.8: Grafické zobrazení submodelu BillOfMaterial, který umožnuje vyvářet vazby mezi entitami.

3 TestBed Barman

Tato část se zabývá TestBedem Barman a jeho fyzickým popisem. TestBed se rozkládá na pracovní ploše 2000 x 1000 mm, na které je uloženo osm autonomních procesních buněk, z těchto osmi buněk jsou dvě transportní a šest výrobních. Jednotlivé buňky jsou pak na pracovní ploše uloženy do takzvaných slotů, kdy čtyři sloty jsou umístěny na pracovním stole a dvě za ním. Umístění slotů závisí na dosahu manipulačního robota. Buňky mají dvě standardizované velikosti: malá buňka(š: 330 mm x h: 330 mm x v: 500 mm), velká buňka (š: 760 mm x h: 330 mm x v: 1500 mm). Každý slot je vybaven dvěma konektory, kde první konektor slouží pro připojení napájení a druhý pro připojení komunikačního rozhraní. Pro komunikaci se využívá rozhraní Profinet[3].
Buňky mají předem definovanou trajektorii, která je manipulačním robotem dodržena pro vložení nebo vybrání skleničky z procesní buňky.

3.1 Dopravník

Dopravní je jedna ze dvou transportních buněk. Jeho hlavní úlohou je propojení Barmana s uživatelem(obsluha). Zajišťuje tak transport hotových drinků mimo operační dosah robotu, a tím zajistit nutnou částečnou bezpečnost celého Barmana. Kromě těchto základních funkcí může dopravník sloužit i k odložení rozpracovaných drinků nebo pro odběr již prázdných sklenic. K tomuto je vybaven deseti vozíky na sklenice a je schopen detekovat jejich stav a obsah.

3.2 Scara robot

Druhou transportní buňkou je pak Scara robot, který zajišťuje přepravu sklenic mezi jednotlivými procesními buňkami a dopravníkem. Robot je vybaven efektorem díky, kterému je schopen uchopit skleničku.

3.3 Sklad sklenic

Sklad sklenic slouží pro uchovávání čistých i špinavých sklenic, které mohou být malé i velké. K tomuto účelu je vybaven dvouosým manipulátorem a čtyřmi vertikálními zásobníky. Manipulátor je umístěn v buňce tak, aby pokryl dva zásobníky na levé i pravé straně.

3.4 Sklad nealkoholických nápojů

Tato buňka se skládá ze čtyř nerezových tanků, které jsou chlazeny pomocí chladícího okruhu pracovního stolu. Funkcí buňky je dávkování většího množství tekutin jako jsou džusy, koly nebo toniky. Dávkování je zprostředkováno pomocí elektromagnetických ventilů a pulsních průtokoměrů. Pro kontrolu je na místě pro sklenici umístěn tenzometrický článek, který hlídá nadávkované množství.

3.5 Sklad alkoholu

Zásobník alkoholových nápojů slouží pro dávkování malých objemů do sklenic. K tomuto využívá lineární manipulátor na rotačním kloubu, díky kterému manipuluje se sklenicí. Může obsloužit až dvacet zásobníků, kdy každý zásobník je vybaven dávkovačem.

3.6 Shaker

Úkolem Shakeru je protřepat obsah vložené sklenice. Buňka je poháněna stejnosměrným elektromotorem, který pomocí klikové hřídele převádí rotační pohyb motoru na lineární(vertikální). Během míchání je sklenička uzavřená pomocí nerezového víčka a elektromagnetického ventilu.

3.7 Výrobník sody

V této buňce je skladována pitná voda, ze které se pomocí CO_2 vytváří soda. Samotná soda se vyrábí v dávkách, velikost dávky se odvíjí od velikosti sytící láhve od společnosti SodaStream. Dávka je odměřována a kontrolována pomocí tenzometrického článku.

3.8 Drtič ledu

Drtič ledu je uzpůsoben pro uchovávání a dávkování nadrceného ledu do sklenice. Led uchováván pomocí Peltierova chladícího systému v podobě kostek nad drtičem ledu. Drcení probíhá pomocí rotačních nožů. Dávkování požadovaného množství je měřeno pomocí tenzometrického článku.

4 Simulační software

Simulátory diskrétních událostí představují důležitý nástroj pro modelování a analýzu dynamických systémů v mnoha oblastech, od průmyslové výroby a logistiky až po telekomunikaci a počítačové sítě. Tyto programy poskytují mnoho funkcí pro vytváření komplexních simulačních modelů a jejich následnou analýzu, s cílem jejich optimalizace, nalezení rizik nebo zjištění chování. V této kapitole si některé tyto simulátory stručně představíme. Následně se více zaměříme a rozebereme open-source simulátor OMNeT++.

4.1 Network Simulator 3

Zkráceně NS3 je open-source síťový simulátor diskrétních událostí, zaměřený na výzkum a vývoj. Simulátor je napsán v jazyce C++/Python a nahrazuje předchozí verzi NS-2. Hlavním cílem NS-3 je vytvořit pevné simulační jádro, které je dobře zdokumentováno, snadno použitelné, laditelné a vyhovuje potřebám práce konfigurace systému a jeho následné analýze výsledných dat. Umožňuje modelovat různé typy sítí, včetně bezdrátových a mobilních[13].
Mezi hlavní přednosti tohoto simulátory patří rozsáhlá a aktivní komunita, která přispívá novými modely a opravuje ty, které jsou již vydané. Díky tomu, tak je i dobře vedená a zpracovaná dokumentace a to jak programu jako takového tak i příkladů. Díky tomu jsou začátky nových uživatelů s NS-3 jednodušší. Široká škála modelů a funkcí umožňuje namodelovat síť přesně podle představ programátora[13]. Jako zápory pak můžeme vidět složitější zprovoznění na operačním systému Windows. Při vytváření velkých a komplexních sítí mohou být simulace výpočetně náročné. U modelování časově náročných systémů může uživatel narazit na problém s přeskakováním některých procesních kroků nebo zpoždění během simulace. To může následně vést k nepřesným výsledkům nebo zkreslenému chování simulovaného systému[14].

4.2 Witness Horizon

V roce 2016 společnost Lanner vydala simulační program Witness Horizon, který umožňuje modelování a analýzu komplexních systémů v průmyslu, logistice a službách. Disponuje širokými modelovými prostředky a prediktivními simulačními funkcemi, které podporují rozhodování a umožňují nejlépe vyvážit poskytování služeb a jejich náklady. Witness Horizon obsahuje velké množství běžných modelovacích

prvků, které uživatel může jednoduše nakonfigurovat. Díky intuitivnímu uživatelskému rozhraní a velké dostupnosti modelů lze vytvářet systémy bez nutnosti náročného programování. Pro případy nutnosti programování nabízí aplikace vlastní programovací jazyk WITNESS Action Language, ovšem podporuje i externí knihovny napsané v běžných jazycích jako jsou C++, C# nebo VB.net. Díky robustním analytickým nástrojům lze provádět různé scénářové analýzy, pro porozumění chování systému a identifikaci možných rizik a slabých míst[12].

4.3 Plant Simulation

Jedná se o DES software vyvinutý společností Siemens. Určená k modelování, simulaci, analýze, vizualizaci a optimalizaci výrobních systémů, materiálových toků a logistiky. Je součástí balíčku Tecnomatix. Plant Simulation umožňuje vytvářet 3D modely pomocí interních knihoven nebo externích dat. Díky dědičnosti a hierarchii je možné vytvářet, simulovat a udržovat velké komplexní systémy od jednotlivých linek až po závody. Nespornou výhodou je možnost integrace mnohých komunikačních rozhraní jako například MQTT, OPCUA, Oracle SQL nebo lze jednoduše integrovat další aplikace od společnosti Siemens (např. TIA Portal, PLCSIM Advanced, SIMIT). Také disponuje analytickými nástroji pro hodnocená výkonosti systému, detekce problémových míst, využití zdrojů a strojů nebo analýzy nákladů.[15]

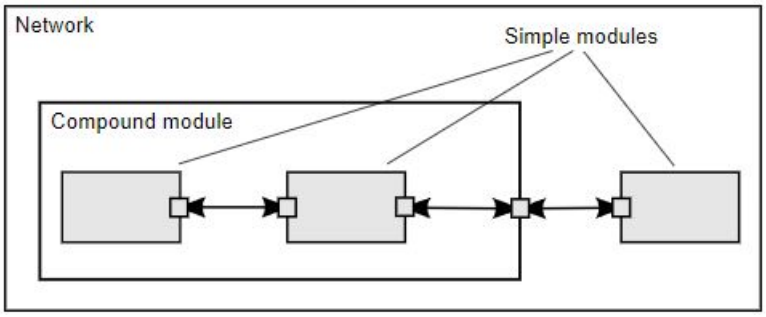
4.4 OMNeT++

OMNeT++ (Objective Modular Network Testbed in C++) je open-source modulární simulátor diskrétních událostí založený na objektech napsaných v jazyce C++ a vlastním jazyku NED. Je primárně využíván pro modelování a simulaci sítí. Ta je myšlena v širším slova smyslu ten zahrnuje drátové i bezdrátové komunikace, senzorové sítě, ad-hoc, internetové protokoly nebo modelování výkonu[16].
OMNeT++ umožňuje provádět simulaci v příkazovém jádru nebo v grafickém rozhraní s možností 2D a 3D simulace. Grafické rozhraní je založeno na Eclipse.

Simulátor můžeme spustit s grafickým rozhraním na běžných OS, Windows, Linux a macOS. Bez IDE pak OMNeT++ funguje na platformách, kde je k dispozici moderní C++ kompilátor například Docker nebo Core. Simulátor je zdarma pro akademické a neziskové účely. Při použití pro komerční užití je nutné si koupit OMNEST, který je téměř totožný s OMNeT++[16].

4.4.1 Moduly

Při modelování sítí v OMNeT++ se jednotlivé modely do sebe vnořují hierarchicky. Hierarchicky nejvýše postavený modul se nazývá systémový, tvoří tak základ pro modelovanou síť a vkládají se do něho jednotlivé prvky jako moduly. Tyto moduly mohou být dvojího typu složené a jednoduché. Složené moduly se mohou skládat z dalších složených modulů nebo modulů jednoduchých víz.4.1. Jednoduché moduly jsou nedělitelné a slouží tak, jako základní stavební kámen celé struktury.
Tyto jednoduché moduly jsou naprogramovány v jazyce C++. Jako nejjednodušší funkční prvky struktury generují a reagují na události. Složené moduly slouží pouze k seskupení více jednoduchých modelů, a tím umožňují vytváření složitějších struktur, samy ovšem nevytvářejí žádné události. Komunikace mezi jednotlivými moduly



Obr. 4.1: Zobrazení jednoduchého a složeného modulu v simulované síti[17].

je zajištěna pomocí zpráv. Tyto zprávy mohou obsahovat libovolná data nebo struktury, které jim uživatel přiřadí. Zprávy jsou přijímány nebo odesílány pomocí bran, které mohou být vstupní, výstupní nebo kombinací obou zmíněných. Samotné spojení mezi jednotlivými moduly je realizováno pomocí kanálů, kterým lze definovat specifické vlastnosti jako například zpoždění, přenosová rychlost nebo chybovost. Ty to spojení je možné vytvářet pouze na jedné hierarchické úrovni. Komunikaci mezi jednoduchými moduly, které nejsou na stejné úrovni zajišťují složené moduly[17].

4.4.2 Jazyk NED

K popisu parametrů a vlastností sítě i modulů samotných používá OMNeT++ vlastní jazyk NED (Network Description). Výstupem tohoto jazyka jsou pak soubory s koncovkou .ned. V těchto souborech uživatel definuje parametry, brány, spojení modulů. Také je možné deklarovat jednoduché moduly a případě potřeby je sestavovat do modulů složených. NED soubory je možné převádět do formátu XML a zpět bez

obavy ze ztráty dat, je tak možné vygenerovat nebo upravit NED pomocí informací z jiných systémů[17].

5 Simulace TestBedu

Simulace TestBedu Barman byla vytvořena v simulčním softwaru OMNeT++. Jak již bylo zmíněno simulační síť se tvoří pomocí modulů, které reprezentují vytvořené třídy. Jejich funkcionalitu programuje sám uživatel pomocí metod.
OMNeT++ umožňuje dvojí způsob programování a to s použitím *handleMessage()* anebo s použitím *activity()*. V druhém případě je programování podobné jako běžnému psaní funkce *main()* a zní volání funkcí a metod nebo čekání na příchozí zprávu v jakémkoliv místě kódu či zastavení jeho vykonávání. S koncem vykonávání funkce *activity()* je modul ukončen (Simulace nadále může pokračovat nebo skončit, pokud již nejsou jiné moduly, které mohou běžet).
Takovéto programování má výhody v možnosti ukládat do lokálních proměnných, není potřeba vytvářet metodu *initialize()* a styl programování je podobný klasickému v některých případech. Mezi nevýhody patří omezená škálovatelnost(vyšší nároky na paměť), pomalejší běh programu a špatná srozumitelnost kódu, kdy s rostoucí složitostí modulu se stává funkce *activity()* velikou, monilitickou funkcí.

V případě použití *handleMessage()* je funkce volána pokaždé když do modulu přijde zpráva(nastane událost). Konec simulace tedy nastane jakmile nejsou přeposílány žádné zprávy tudíž nenastávají žádné události. Pro správnou funkčnost je nutné, aby se po přijetí zprávy metoda *handleMessage()* rychle a korektně ukončila, jinak není možné přijímat další zprávy. Při tomto způsobu je modul vytvořen a je zavolána metoda *initialize()*, kde uživatel může provést základní nastavení modulu a jeho parametrů.
Z toho vyplývají určité podmínky použití.

1. V metodách nelze čekat na příchozí zprávy. Proto jsou využívány tzv.*self-message* tedy zprávy, které posílá modul sám sobě pomocí metody *scheduleAt()*, tím může uživatel vytvářet zpoždění nebo časovače.
2. Uživatel musí zajistit, aby alespoň v jednom modulu byla při inicializaci vytvořená a poslána zpráva jinému modulu nebo *self-message*. V opačném případě by simulace ihned po inicializaci skončila.

Tento způsob má dvě zásadní výhody, menší spotřebu paměti a provádí se rychleji. Nevýhodou pak je nemožnost vytváření lokálních proměnných a nutnost definovat metodu *initialize()*.

5.1 Message

Jak již bylo v předchozí kapitole napsáno, komunikace mezi moduly probíhá za pomoci zpráv. Ty jsou v programu prezentovány třídou *cMessage*, ve které jsou definovány všechny proměnné s informacemi, které by mohl uživatel potřebovat, a to

jak pro identifikaci zprávy, tak pro práci s ní. Ke zprávě je možné dynamicky v programu připojit uživatelem vytvořené objekty a ty pak společně se zprávou odeslat. Omnet++ také umožňuje vytvořit si zprávu vlastní, která se dědí z třídy *cMessage*, a v ní si nadefinovat vlastní potřebné proměnné.

V rámci OMNeTu++ platí pravidlo, že příjemce zprávy je její vlastník a musí se o ni řádně postarat, čili zprávu smazat nebo přeposlat. Nesprávně nakládání se zprávami má za následek blokování paměťových prostředků.

Pro účely simulace byla vytvořena vlastní zpráva s názvem. Definice této zprávy se provádí v souboru s příponou **.msg**. Následně pak program sám vytvoří hlavičkový a zdrojový kód se všemi nutnými metodami pro správnou funkčnost. Ve výpisu 5.1 můžeme vidět, že pro náš účel byla vytvořena zpráva s celkem čtyři proměnnými.

Výpis 5.1: Vytvořená zpráva pro TestBed Barman

```
namespace barman;

message Message {
 int source;
 int dest;
 int code[4] = 0;
 bool response;
}
```

První proměnná *source* ukládá hodnotu pozice buňky barmana nebo indexu skleničky, ze které byla zpráva poslána. Druhá proměnná *dest* je přesným opakem té první tedy ukládá cílovou pozici nebo index zprávy. Poslední proměnnou je *response* ta uchovává odpověď modulu odesílajícího zprávu.

Code

Nejdůležitějším prvkem zprávy je proměnná *code*. Jedná se o čtyřmístný kód, který určuje požadavky na službu o kterou sklenička žádá. Každá pozice má svůj význam:

1. místo určuje typ požadované buňky
2. místo definuje požadovanou operaci
3. místo může být použito jako požadovaný typ materiálu nebo jako počáteční pozice pro operaci transportu
4. místo má také dvojí využití, první možností je množství požadovaného materiálu v druhou pak konečná pozice pro transport

Pro jednotlivé pozice, typ buňky, plánovaná operace a typ materiálu byly vytvořeny proměnné enum s možnými stavy. Všechny tři tyto enumy(*position, operation, material*) jsou vytvořeny v souboru *code.h*, který je zobrazen v příloze viz.A.1.

Kind

Abychom nemuseli do vlastní zprávy přidávat přílišné množství proměnných pro určení typu zprávy. Byl využit parametr *Kind* z původní třídy *cMessage*, ten je v rámci simulace využíván pro rozeznání typu zprávy, zda se jedná o rezervační, potvrzovací, prováděcí nebo zprávy o opuštění buňky. Všechny používané typy zpráv jsou vytvořeny v souboru *code.h* viz.A.1.

5.1.1 Vytváření zpráv

O vytváření zpráv se stará metoda *CreateMessage()*. Sklenička i ostatní výrobní buňky mají metodu pro vytváření zpráv prakticky identickou. Jediným rozdílem u skleničky, je nutnost vkládat jméno zprávy jako argument při volání metody. Z principu funkčnosti všechny buňky v barmanu pouze odpovídají na příchozí zprávy skleničky proto se v tomto případě zpráva jmenuje vždy *Response*.

Výpis 5.2: Ukázka metody pro vytváření zpráv u modulu skleničky

```
Message *Glass::CreateMessage(int *code, bool response,
                    short kind, int dest, char *msgname){
    int src = glass_number;   // module index
    CreatedMessages++;

    Message *nmsg = new Message(msgname);
    nmsg->setSource(src);
    nmsg->setDest(dest);
    nmsg->setKind(kind);
    nmsg->setResponse(response);
    for(unsigned i = 0; i < 4; i++){
        nmsg->setCode(i, *code);
        code++;
    }
    return nmsg;
}
```

V ukázce kódu 5.2 můžeme vidět jak se vytváří nová zpráva, do které se následně pomocí automaticky vygenerovaných metod vkládají všechny potřebné proměnné.

5.2 Buňky Barmana

V rámci nasimulování Barmana bylo vytvořeno sedm typů buněk. Jednotlivé buňky budou blíže popsány jednotlivě. Protože všechny typy buněk vyjma skleničky mají v obecnosti podobný způsob roztřídění a reakce na přijaté zprávy, byl proto vytvořen obecný stavový diagram, který je pro svou velikost umístěn v přílohách viz.B.4.

5.2.1 Skladová jednotka

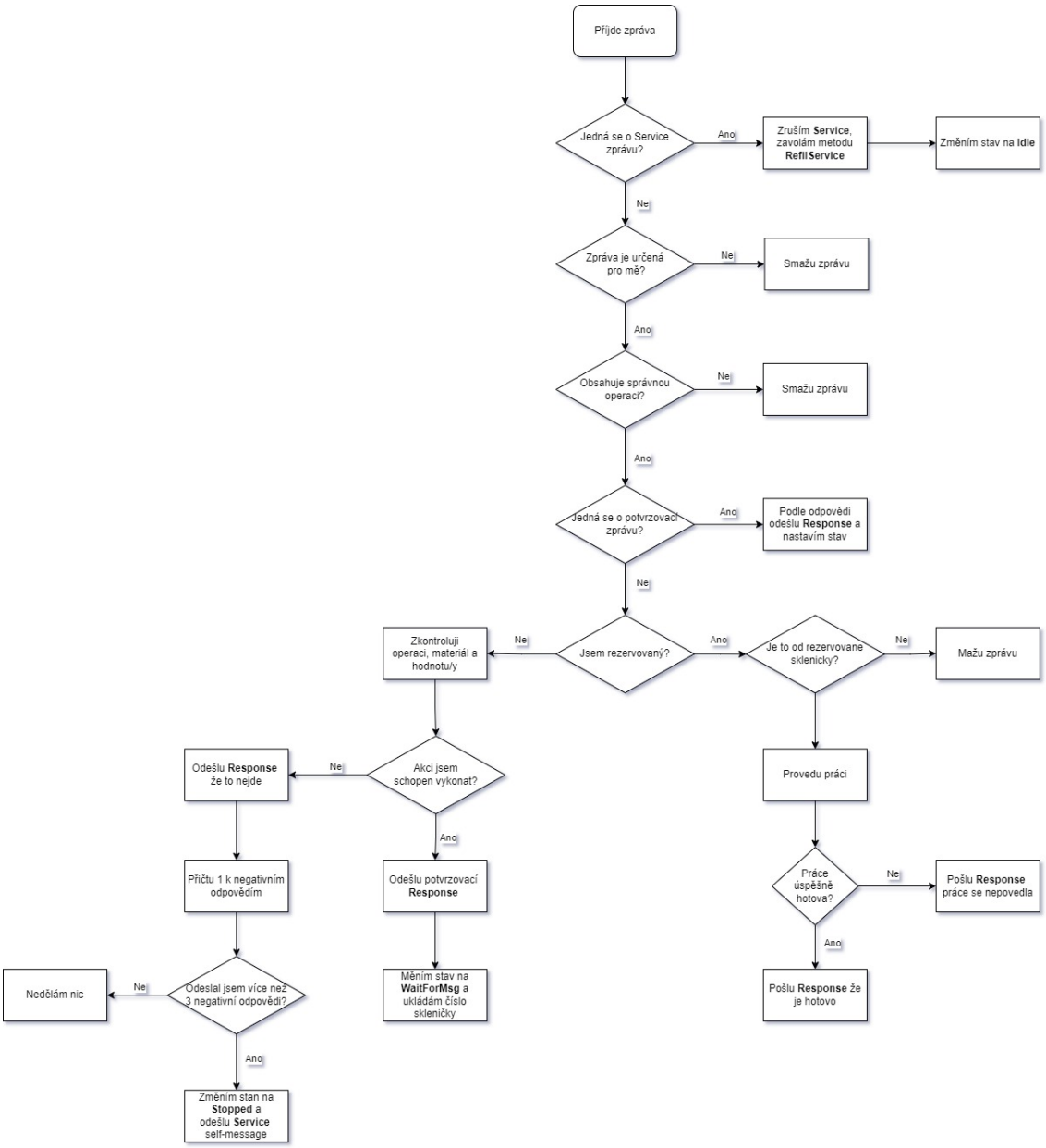
Modul skladové buňky je v modelované síti simulace nejpočetnější. Zastupuje sklad skleniček, alkoholu, nealkoholických nápojů, sody a sklad ledové tříště. Na obr.5.1 můžeme vidět stavový diagram vyhodnocení přijaté zprávy skladovým modulem. Po přijetí zprávy se nejdříve zjišťuje, zda se jedná o tzv. *self-message* (Servisní metoda) nebo zprávu z cizího modulu. Následně podle typu zprávy kontrolujeme typ požadované buňky a operaci, v případě, že je vše v pořádku proběhne kontrola zdali sklad obsahuje požadovaný materiál a podle výsledku odešle odpověď. Při následném přijetí zprávy pro vykonání služby je zkontrolován původ je-li zpráva od skleničky, která má buňku rezervovanou a obsahuje správný typ buňky, poté se již provede samotné vyskladnění požadovaného materiálu. Po vykonání práce je odeslána zpráva o jejím výsledku. Kromě rezervace a vykonání služby obsahuje modul ještě servisní metodu pro znovu doplnění zásob materiálu v případě, že je opakovaně dotazován na určité materiály, které by skladový modul měl mít naskladněn.

5.2.2 Výrobní jednotka

Výrobní buňka reprezentuje v simulaci Shaker. Na obr.5.2 můžeme vidět stavový diagram akcí po přijetí zprávy. Modul nevytváří žádné *self-message*, a proto je jeho diagram jednoduchý.

5.2.3 Robot

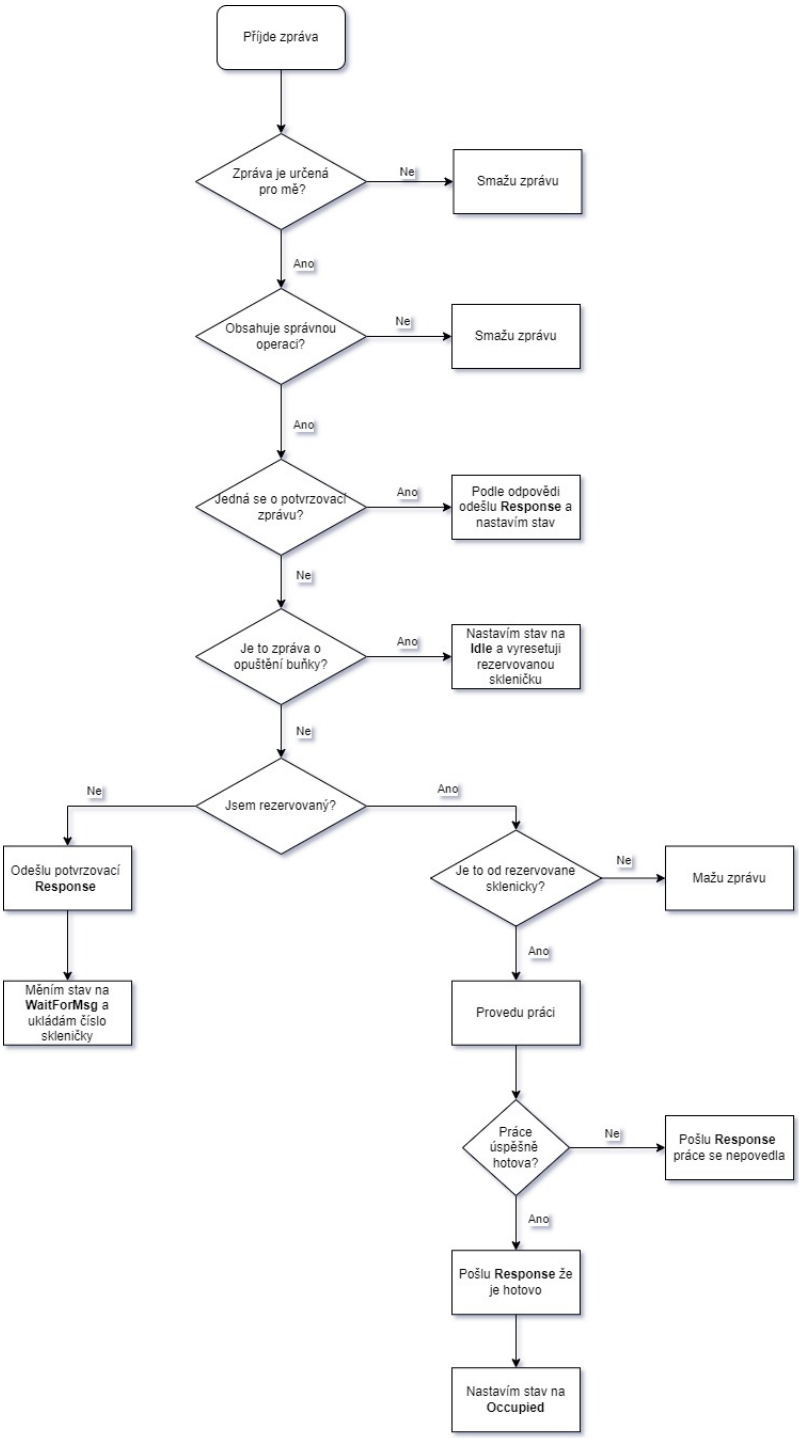
Modul robota slouží pro transport skleniček mezi jednotlivými výrobními buňkami. Jedná se o nejvíce vytíženou buňku v simulátoru. Na obr.5.3 můžeme vidět stavový automat vyhodnocení zpráv, které modul robota příjme. Po přijetí rezervační zprávy, je-li robot ve stavu *Idle* následuje kontrola zda obsahuje správný typ buňky a operace. Pokud je vše v pořádku, je odeslána odpověď a robot čeká na potvrzení nebo zrušení rezervace. Pokud buňka příjme zprávu pro vykonání služby, zkontroluje zda je zpráva od rezervované skleničky a má správný typ buňky, po kontrole je vykonána služba a odeslán výsledek.



Obr. 5.1: Stavový diagram Skladu.

5.2.4 Dopravník

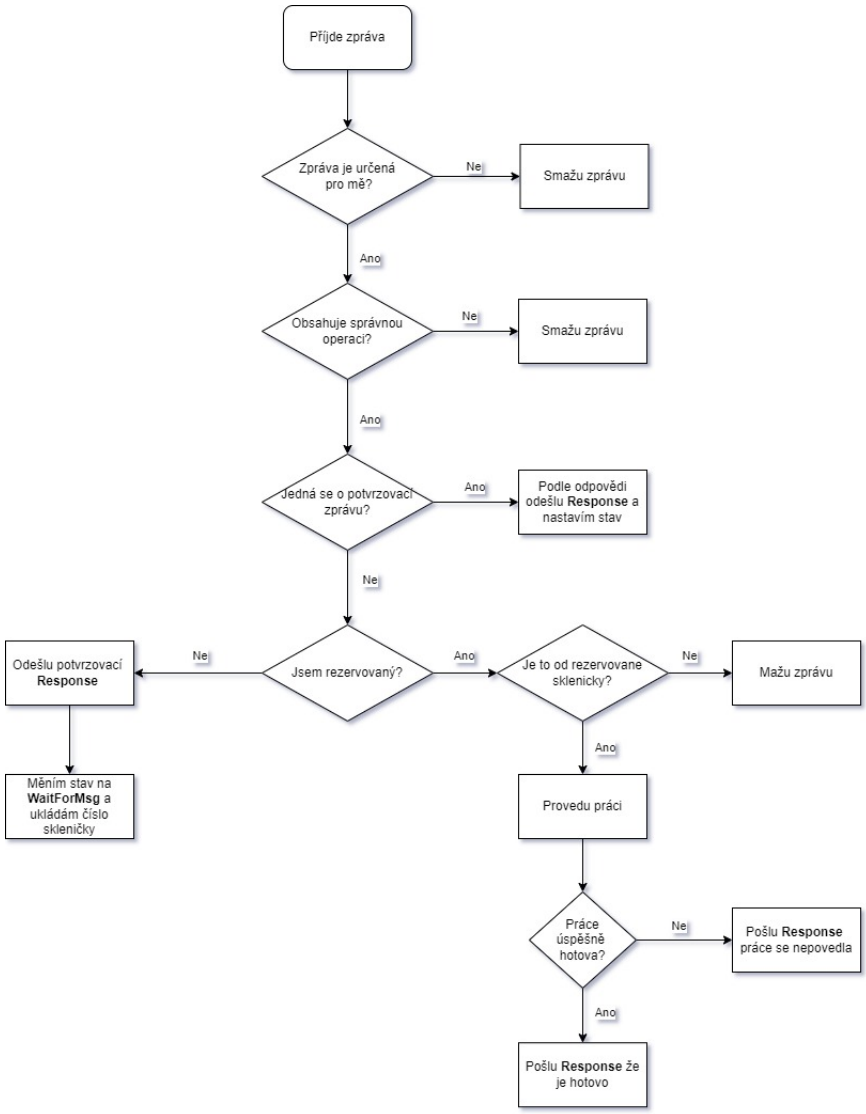
Funkce dopravníku spočívá ve výdeji dokončených drinků do rukou obsluhy nebo zákazníka. Při potvrzení rezervace skleničkou, připraví modul vozík na uložení skleničky, podle jejího čísla. Následná zpráva pro provedení služby přesune vozík mimo pracovní pole působnosti robotu, aby mohl být bezpečně odebrán z barmana. Po přijetí potvrzení vykonané služby od rezervované skleničky dopravník resetuje rezervaci skleničky a nastaví stav na *Idle*.



Obr. 5.2: Stavový diagram výrobní buňky.

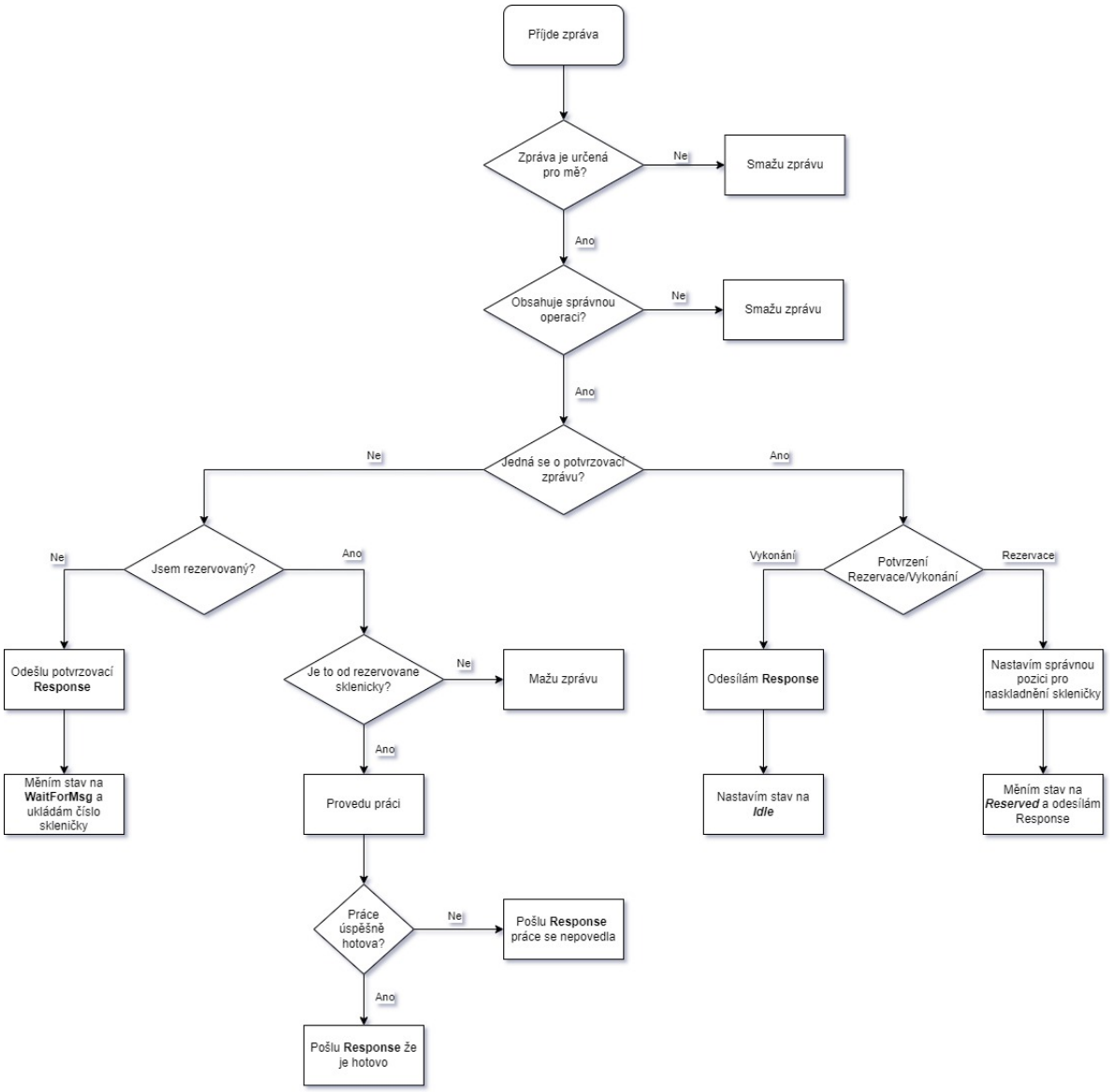
5.2.5 Mezisklad

Buňka meziskladu byla vytvořena speciálně pro simulaci a v reálném Barmanovi se nenachází. Slouží pro uskladnění rozpracovaného drinku, který z důvodu nemož-



Obr. 5.3: Stavový diagram transportu.

nosti zarezervovat a přesunout se na následující výrobní buňku, a tak splnit další krok receptu. Tím může zabírat prostor aktuální výrobní buňky jiné skleničce, která čeká na její uvolnění. Důvody proč není možné si buňku zarezervovat, mohou být následující. Za prvé buňka je ještě obsazená stávající skleničkou, která momentálně provádí výrobní operaci nebo již čeká na transport do další buňky. Druhou možností je, že samotná buňka již nemá dostatečné množství materiálu požadovaného skleničkou pro provedení služby, skleničce tak nedorazí žádná kladná odpověď.
Na obrázku 5.5 můžeme vidět, jak reaguje mezisklad na příchozí zprávu. Po přijetí rezervační zprávy a zkontrolování náležitostí, jako stav meziskladu, typ požadované buňky a operace, se ověří zda-li je volné místo v meziskladu v případě požadavku o uskladnění nebo zda se daná sklenička již nachází v meziskladu pokud chce vy-

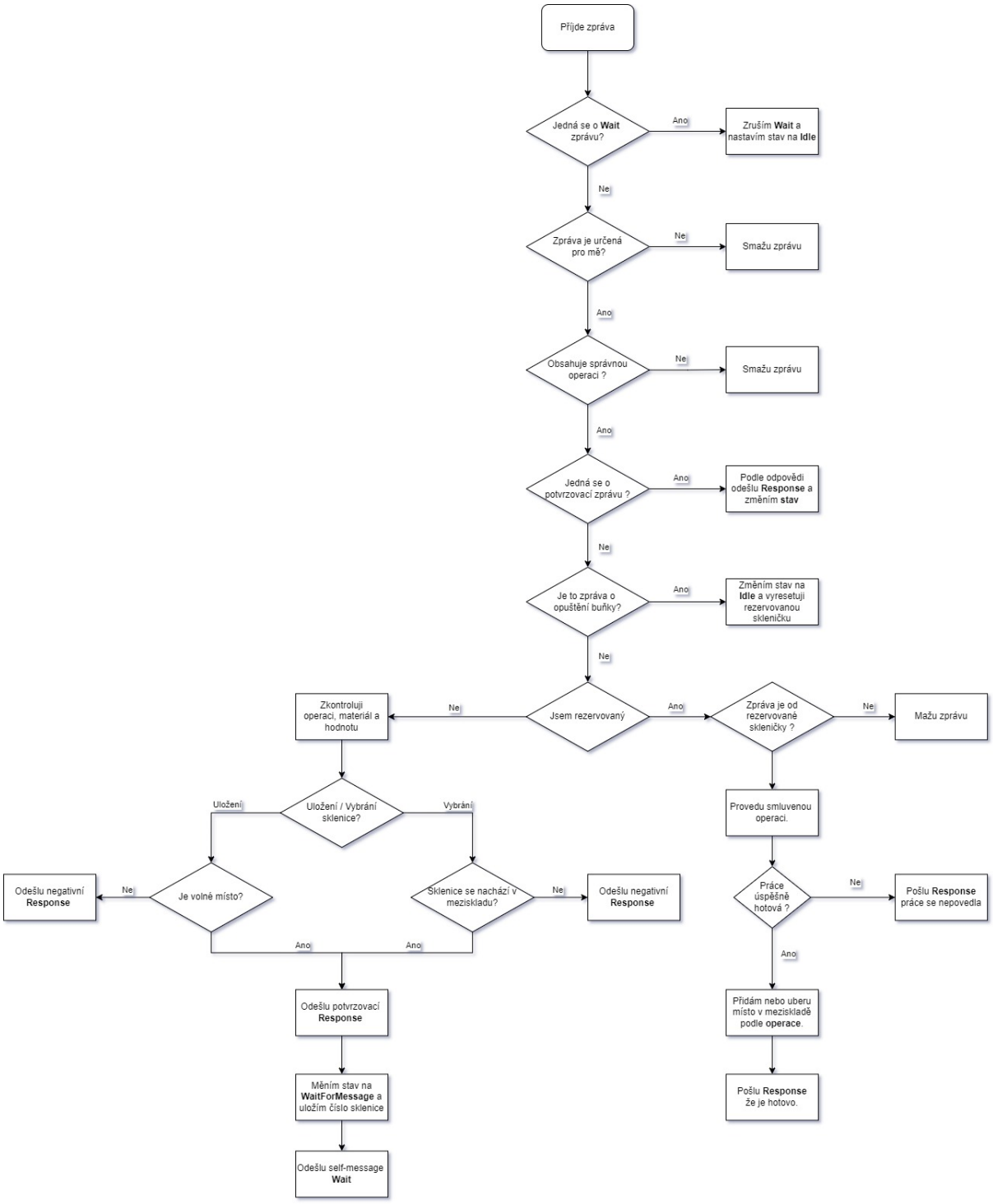


Obr. 5.4: Stavový diagram dopravníku.

skladnit. Poté je odeslána odpověď a nastaven časovač pro nutné potvrzení rezervace danou skleničkou. Pokud uplynul čas pro přijetí potvrzovací zprávy je rezervace resetována a mezisklad se vrací do stavu *Idle*, to se stane i v případě pokud dorazí potvrzení s negativní odpovědí. V opačném případě je mezisklad zarezervován a čeká na zprávu pro vykonání služby.

Po přijetí a kontrole zprávy o provedení služby je podle typu operace provedena práce naskladnění nebo vyskladnění. V případě uskladnění se mezisklad po přijetí potvrzovací zprávy vrací do stavu *Idle*. Pokud se jedná o vyskladnění, mezisklad po přijetí potvrzovací zprávy čeká na pokyn o vyskladnění. Po obdržení zprávy o

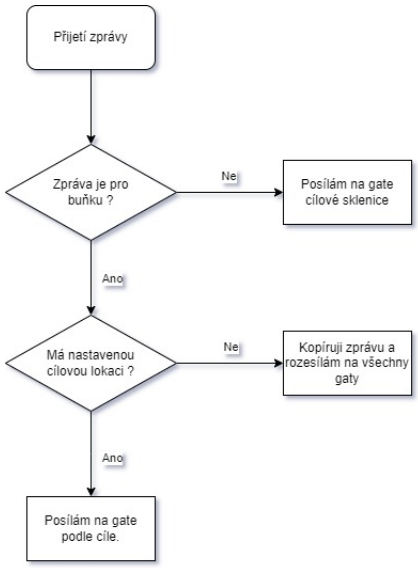
opuštění buňky se vrací do stavu *Idle*.



Obr. 5.5: Stavový diagram Meziskladu.

5.2.6 Switch

Buňka switche byla vytvořena pro zjednodušení a zredukování počtu propojení mezi jednotlivými buňkami a skleničkami. Tak aby nebylo nutné propojovat každou jednu buňku s každou skleničkou.
Funkcionalita switche je ve své podstatě velmi jednoduchá. V případě příchozí zprávy z výrobních buněk, se zpráva odešle portem pro komunikaci se skleničkami, konkrétní gate určuje proměnná *dest* ve zprávě. V opačném případě je funkce podobná. Avšak v případě že je proměnná *dest* nastavená na hodnotu -1, sklenička neví kam přesně je potřeba zprávu poslat. Switch proto zprávu nakopíruje a rozešle na všechny výrobní buňky, které jsou k němu připojeny.



Obr. 5.6: Stavový diagram switche.

5.2.7 Jednotka skleničky

Modul skleničky zaujímá v simulační síti místo aktivního prvku. Iniciuje všechny počáteční zprávy pro rezervaci nebo vykonání služby. Jak je možné vidět ze stavového diagramu viz. obr.B.1. Začíná modul skleničky inicializací nastavením potřebných proměnných do základních stavů, vytvořením zprávy pro Delay a výběrem receptu. Po nastavení parametrů jsou volány metody *Recipe()*, která zajistí nastavení počátečního kódu. Následně zjistíme zda byl nastaven parametr Delay na pozitivní hodnotu. Pokud ano naplánuje se čekání s náhodnou délkou z rovnoměrného rozložení. Tudíž první volání metody *RecipeStep()* nastává až z metody pro zpracování

zpráv *handlingMessage()*. Jestliže má parametr Delay negativní hodnotu metoda *RecipeStep()* se zavolá ihned. Tato metoda zajišťuje kompletní vykonání kroku receptu od rezervace výrobní buňky a transportu až po vykonání rezervovaných služeb.

Výpis 5.3: Inicializační metoda skleničky

```
void Glass::initialize(){                                                    1
    WATCH(CreatedMessages);                                                  2
    warehouse = false;                                                       3
    recipe_kind = intuniform(1,6);                                           4
    act_step = 1;                                                            5
    phase = 1;                                                               6
    act_position = Nowhere;                                                  7
    Stav = Idle;                                                             8
    delay = new cMessage("Delay");                                           9
    glass_number = 0;                                                        10
    if(isVector())                                                           11
        glass_number = getIndex();                                           12
                                                                             13
    Recipe();                                                                14
    if(par("Delay")){                                                        15
        scheduleAt(simTime() + uniform(0,30), delay);                        16
        InitialTime = delay->getArrivalTime().dbl();                         17
    }                                                                        18
    else{                                                                    19
        RecipeStep();                                                        20
        InitialTime = simTime().dbl();                                       21
    }                                                                        22
}                                                                            23
```

Poté co je skrze metodu *RecipeStep()* odeslána první zpráva, čeká sklenička na kladnou odpověď od výrobních buněk. V případě že nedostane žádnou kladnou odpověď zjišťuje zdali je ve výrobní buňce, kterou musí opustit aby v případě potřeby uvolnila místo. Zjistí-li, že je nucena buňku opustit odesílá rezervační zprávu do meziskladu v opačném případě nastaví Delay a čeká než znovu pošle žádost o rezervaci. Po přijetí alespoň kladné odpovědi je na výrobní buňku odeslána potvrzovací zpráva. Pokud je rezervace úspěšná je navýšena proměnná s mezikrokem a v případě potřeby se odesílá rezervace robota. V případě že nedorazí žádná kladná odpověď je nastaven Delay, který je kratší než u rezervace buňky. Jakmile je nutný transport zarezervován, nastává další mezikrok v podobě provedení transportu. Po jeho potvrzení je odeslána zpráva o vykonání služby zarezervované výrobní buňce.
Potvrzením práce ve výrobní buňce jsou dokončeny všechny mezifáze a je volána

metodou *Recipe()*, pro načtení dalšího kroku receptu. Tím se ptáme zdali jsme na konci receptu. Celý proces opakujeme dokud se nedostaneme na konec receptu. Čím je naskladnění skleničky na dopravník.
Pokud jsme se během procesu dostali do meziskladu, je mezikroková sekvence jiná. Po zarezervování výrobní buňky je potřeba zarezervovat i mezisklad samotný a následně robota. Pořadí vykonání služeb se také nepatrně mění. Nejdříve se provede vyskladnění skleničky z meziskladu, následuje přenos a končí vykonáním služby ve výrobní buňce.
Pokaždé když sklenička opouští výrobní buňku odesílá po provedení transportu na předchozí místo zprávu o opuštění buňky čímž ji uvolní.

5.3 Soubory Ned

V kapitole 4.4.2 jsme si řekli, že pomocí NED souborů definujeme parametry a vlastnosti modelované sítě a modulů samotných.
Vytvořené NED soubory modulů potřebné pro simulaci obsahují pouze definování parametrů a bran. Parametry se používají ve zdrojových souborech a jejich hodnoty se nastavují v konfiguračním souboru jedná se například o pozici v barmanu nebo v případě skladu o množství skladovaného materiálu.Jsou definovány pomocí datového typu a můžeme pro ně nastavit defaultní hodnotu. V sekci bran nastavujeme vstupy, výstupy nebo jejich kombinaci. To je provedeno definováním typu(input,output,inout) a jménem, pokud víme, že budeme potřebovat připojit na jeden port více propojení můžeme z něj udělat pole. To je výhodou v případech, kdy nevíme kolik přesně budeme vstupů/výstupů potřebovat nebo plánujeme jejich změnu v průběhu simulace. Pro ukázku je vložen NED soubor transportního modulu5.4.

Výpis 5.4: Ned soubor transportní jednotky

```
package barman;

simple Transport
{
    parameters:
        int cellpos = default(4);

    gates:
        inout out;
}
```

Samotná simulační síť je vytvořená v souboru *Barman.ned*, ten můžeme vidět zde 5.5. Abychom mohli vytvořené moduly přidat jako submoduly přidat do sítě,

je potřeba nejdříve je naimportovat. Poté již v sekci submodules můžeme přidávat jednotlivé buňky do sítě a případně upravovat jejich vizuální parametry. V sekci connections vytvořené moduly propojujeme podle potřeby. Můžeme si povšimnout, že propojení skleniček se switchem je realizováno pomocí smyčky for. Tím je garantováno, že při změně počtu skleniček budou na každém začátku simulace všechny skleničky se switchem spojeny. Pro účely jednoduchého a rychlého změnění počtu skleniček byl přidán parametr NumberOfGlasses, který určuje jejich počet. Poslední využitou sekcí je types, ve které je vytvořen tzv. kanál. Tento kanál můžeme zakomponovat do propojení jednotlivých submodulů. V jeho definici pak můžeme nastavit například zpoždění nebo ztrátovost a tím simulovat vlastnosti reálných drátových nebo bezdrátových spojení.

Výpis 5.5: NED soubor Barman definující parametry simulované sítě

```
package barman.simulations;

import barman.Glass;
import barman.Storage;
import barman.Transport;
import barman.Switch;

network barman{
    parameters:
        int NumberOfGlasses = default(2);

    types:
        channel Channel extends ned.DelayChannel{
            delay = 100ms;}

    submodules:
        Glass[NumberOfGlasses]: Glass {
            @display("p=280,60;i=block/process,gray");
        }
        Switch: Switch {
            @display("p=280,120;i=block/process,gray");
        }
        Robot: Transport {
            @display("p=280,240;i=block/process,gray");
        }
        GlassStorage: Storage {
            @display("p=160,200;i=block/process,gray");
        }

    connections:
        for i = 0..NumberOfGlasses{
            Glass[i].out <--> Channel <--> Switch.glassout++;
        }
        Switch.cellout++ <--> Channel <--> GlassStorage.out;
        Switch.cellout++ <--> Channel <--> Robot.out;
}
```

[a]

[a]Pro přehlednost byly některé řádky z výpisu vymazány.

5.4 Konfigurační soubor Omnetpp.ini

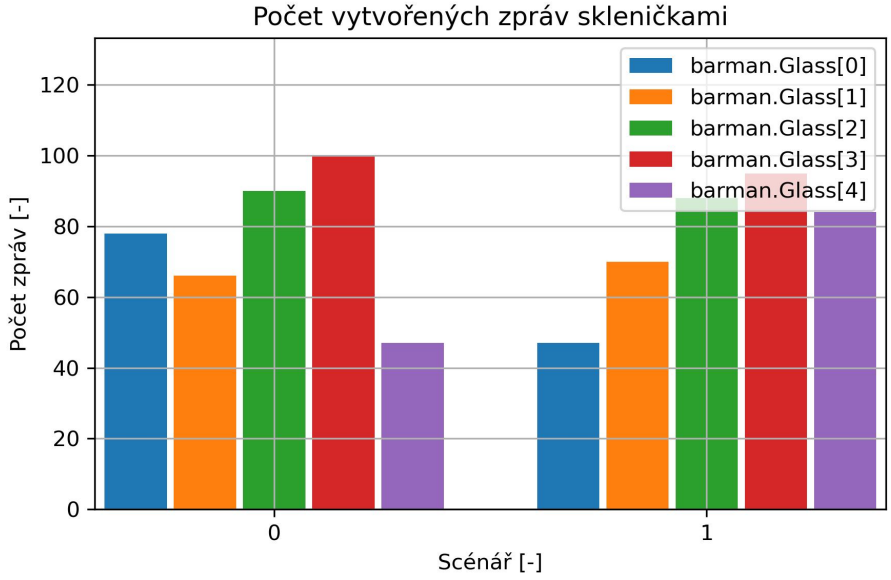
Konfigurační soubor nám umožňuje nastavovat parametry simulace a vytvářet scénáře. U těchto scénářů pak můžeme nastavovat parametry pro moduly, počet opakování simulace, omezení času simulace nebo CPU. Taktéž je možné upravit tzv. *random seed* tedy náhodné číslo, které se používá při inicializaci generátoru náhodných čísel. Při změnách *random seed* se obvykle definuje i počet opakování. Tedy simulace se provede podle definovaného počtu opakování se stejnými parametry, ale jinak generovanými náhodnými čísly.

6 Testování Simulačního Modelu

Pro otestování funkčnosti vytvořeného simulačního modelu byly vytvořeny dva scénáře, které se liší v začátku vykonávání receptur skleniček. V obou scénářích je možné nastavit a provést simulaci pro 1, 2, 5 a 8 skleniček.
V první případě je nastaven parametr Delay, ten zapříčiní vygenerování zpoždění při inicializaci skleničky. Zpoždění je generováno z normálního rozdělení s rozsahem 0 až 30 vteřin. Každá sklenička tak začne vykonávat vlastní recept v jiném časovém okamžiku, tím se snažíme simulovat postupný příchod objednávek v čase.
Druhý scénář parametr Delay nenastavuje, což má za následek okamžité vykonávání receptu všech skleniček v nulovém simulačním čase.

V průběhu simulace zaznamenáváme tři proměnné. Čas začátku vykonávání receptu, čas dokončení receptu a počet vytvořených zpráv danou skleničkou. Ze získaných časů se na konci simulace vypočítává doba, za kterou se recept stihl dokončit.

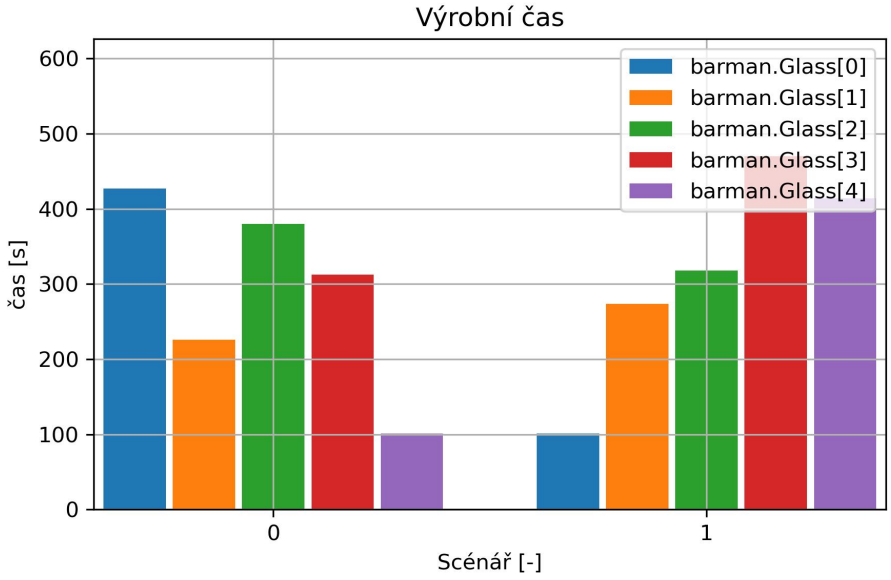
Na prvním grafu vidíme počet vytvořených zpráv pro každou skleničku zvlášť pro oba scénáře. V případě scénáře 1 je to celkem 381 zpráv, u druhého scénáře pak 384. Rozdíl je tedy zanedbatelný.



Obr. 6.1: Počet vytvořených zpráv jednotlivými skleničkami v obou scénářích.

Druhý graf nám zobrazuje výrobní časy jednotlivých skleniček. Pro usnadnění jsem z jednotlivých časů vypočítal průměr. Průměrný výrobní čas u prvního scénáře je 289,12s a u druhého 315,16s. V tomto případě je rozdíl větší, což můžeme připsat

rozdílnému začátku jednotlivých skleniček.
Největší rozdíl mezi oběma scénáři je pak patrný z obou grafů. První dokončené skleničky v obou scénářích mají shodný počet vytvořených zpráv i stejný výrobní čas, avšak v první scénáři se nejdříve dokončila sklenička s indexem 4 a v druhém scénáři pak sklenička s indexem 0. To je způsobeno rozdílným začátkem vykonávání receptu. U prvního scénáře začala čtvrtá sklenička s nejmenším zpožděním. V druhé scénáři má pak sklenička s indexem nula přednost, jelikož se její zprávy díky indexu vykonávají nejdříve.



Obr. 6.2: Výrobní časy pro jednotlivé skleničky v obou scénářích.

7 Podněty pro zlepšení

Při vypracovávání diplomové práce se naskytly možnosti jak dále rozšířit dosavadní vytvořený simulační model Barmana. Jednou z možností je implementace soutěžení služeb nabízených buňkami v Barmanovi mezi jednotlivými skleničkami. Ve stávající verzi práce se o přiřazení buňky skleničce rozhoduje pouze na základě pořadí přijatých zpráv buňkou. Tedy zprávy přijaté buňkou nejdříve jsou vybrány.
Další možnost rozšíření spočívá v nahrazení aktuální komunikace pomocí jednoduchých zpráv, standardem OPC UA díky, kterému by simulační model mohl být zpřesněn a zjednodušen.

Závěr

V úvodní kapitole byly vysvětleny pojmy z Průmyslu 4.0, jedná se zejména o referenční model RAMI 4.0, AAS a způsoby jeho komunikace. Dále byly popsány dva používané standarty komunikace mezi AAS a to OPC UA a MQTT, a bylo poukázáno na možné nevýhody či problémy a jejich možné řešení v případě implementace.
V rámci druhé kapitoly byly popsány navržené co nejvhodnější AAS modely pro výrobní zařízení a inteligentní výrobek v rámci TestBedu Barmana. Byl zde taktéž popsán programový nástroj ve, kterém se modely vytvářely, a samotný způsob vytváření AAS, submodelů a datových struktur.
Dále jsou v rámci třetí kapitoly stručně popsány jednotlivé reálné buňky TestBedu Barman s ohledem na funkcionalitu.
Čtvrtá kapitola se zaměřuje na rešerši trhu v rámci simulátorů diskrétních událostí a zaměřuje se více na open-source simulátor OMNeT++. Zvláště pak na to jakým způsobem funguje jeho simulace. V praktické části byl vytvořen simulační model reálného zařízení TestBed Barman v simulačním programu OMNeT++. Vytvořený model se skládá ze sedmi typů modulů, pět z nich bylo realizováno tak aby koncepčně a funkčně odpovídaly reálným buňkám Barman. Moduly switch a warehouse byly vytvořeny navíc, aby v případě switche zjednodušovaly propojení jednotlivých výrobních buněk a skleniček. Warehouse pak přináší možnost odložit skleničku v případě čekání na následující výrobní buňku.
Předposlední kapitola popisuje testování simulačního modelu. Byly vytvořeny dva jednoduché simulační scénáře společně s vlastnostmi skleničky, které se během simulace sledují.
V poslední kapitole jsou uvedeny možné budoucí zlepšení aktuální verze práce v rámci rozšíření a zlepšení simulačního modelu.

Literatura

[1] HANKEL, Martin a REXROTH, Bosch. *The Reference Architectural Model Industrie 4.0 (Rami 4.0)*. Online. In: ZVEI. 2015, s. 2. Dostupné z: https://www.zvei.org/en/press-media/publications/the-reference-architectural-model-industrie-40-rami-40. [cit. 2023-12-10].

[2] ARM, Jakub; BENESL, Tomas; MARCON, Petr; BRADAC, Zdenek; SCHRÖDER, Tizian et al. *Automated design and integration of asset administration shells in components of industry 4.0. Online. Sensors (Basel, Switzerland).* 2021, roč. 21, č. 6, s. 1-20. ISSN 1424-8220. Dostupné z:https://doi.org/10.3390/s21062004. [cit. 2023-12-10].

[3] KACZMARCZYK, Václav; BAŠTÁN, Ondřej; BRADÁČ, Zdeněk a ARM, Jakub. *An Industry 4.0 Testbed (Self-Acting Barman): Principles and Design.* Online. *IFAC PapersOnLine*. 2018, roč. 51, č. 6, s. 263-270. ISSN 2405-8963. Dostupné z: https://doi.org/10.1016/j.ifacol.2018.07.164. [cit. 2023-12-20].

[4] Marcon, P.; Diedrich, C.; Zezulka, F.; aj. *The Asset Administration Shell of Operator in the Platform of Industry 4.0*. Online. In 2018 18th *International Conference on Mechatronics - Mechatronika (ME)*, Brno, Czech Republic, 2018, pp. 1-5. Dostupné z: https://ieeexplore.ieee.org/abstract/document/8624699[cit. 2024-02-10]

[5] BRADÁČ, Zdeněk; MARCON, Petr; ZEZULKA, František a ARM, Jakub. Digital Twin and AAS in the Industry 4.0 Framework. Online. *IOP Conference Series: materials Science and Engineering*. 2019, roč. 618, s. 8. Dostupné z: https://doi.org/10.1088/1757-899X/618/1/012001. [cit. 2024-02-13].

[6] Salafia, Marco Giuseppe a CAVALIERI, Salvatore. Insights into Mapping Solutions Based on OPC UA Information Model Applied to the Industry 4.0 Asset Administration Shell. Online. In: . 2020. Dostupné z: https://doi.org/10.3390/computers9020028. [cit. 2024-02-13].

[7] BELYAEV, Alexander a DIEDRICH, Christian. *Specification "Demonstrator I4.0-Language"v3.0.* Online. 2019, s. 38. Dostupné z: https://www.researchgate.net/publication/334429449_Specification_Demonstrator_I40-Language_v30. (2019).. [cit. 2023-12-15].

[8] INDUSTRIE 4.0, Platform. *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. In: . Dostupné z: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html. [cit. 2023-12-01].

[9] INDUSTRIE 4.0, Platform. *Details of the Asset Administration Shell - Part 2*. Online. 2021, s. 165. Dostupné z: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part2_V1.html. [cit. 2023-12-01].

[10] Adolphs, P.; Auer, S.; Bedenbender, H.; aj.: *Structure of the Administration Shell: Continuation of the Development of the Reference Model for the Industrie 4.0 Component*. Online. ZVEI and VDI, status report, Duben 2016,[cit. 2024-02-17]. Dostupné z: https://www.academia.edu/35536663/Structure_of_the_Administration_Shell_Continuation_of_the_Development_of_the_Reference_Model_for_the_Industrie_4.0_Component.pdf

[11] ZEZULKA, František; VESELÝ, Ivo a BRAUN, Vlastimil. *Úvod do problematiky a základní modely Industry 4.0*. Online. In: SystemOnline. 2001. Dostupné z: https://www.systemonline.cz/rizeni-vyroby/uvod-do-problematiky-a-zakladni-modely-industry-4.0.htm. [cit. 2023-12-05].

[12] LANNER GROUP LT. *Witness Horizon*. Online. Lanner. 2024. Dostupné z: https://www.lanner.com/en-gb/technology/witness-simulation-software.html. [cit. 2024-04-30].

[13] *What is NS-3?* Online. NS-3. 2011. Dostupné z: https://www.nsnam.org/about/. [cit. 2024-05-01].

[14] ARSLAN, Serhat. *Introduction to NS3 Network Simulator 3*. Online. 2022. Dostupné z: Youtube, https://youtu.be/Jv_swgcykjQ?si=N-8mofiLAIUdglS2/. [cit. 2024-05-01].

[15] SIEMENS. *TECNOMATIX Plant Simulation*. Online. Siemens. 2024. Dostupné z: https://plm.sw.siemens.com/cs-CZ/tecnomatix/products/plant-simulation-software/. [cit. 2024-04-30].

[16] OMNET++. *What is OMNeT++?*. Online. OMNeT++. 2001. Dostupné z: https://omnetpp.org/intro/. [cit. 2024-04-30].

[17] VARGA, András a , OpenSim Ltd. *OMNeT++ Simulation Manual*. Online. 6.0.3. 2024. Dostupné z: https://omnetpp.org/documentation/. [cit. 2024-03-10].

[18] OMNET++. *OMNeT++ Tutorials and Technical Articles*. Online. 2019. Dostupné z: https://docs.omnetpp.org. [cit. 2024-02-20].

Seznam symbolů a zkratek

AAS	Asset Administration Shell - Administrativní obálka aktiv
ERP	Enterprise Resource Planning - Plánování podnikových zdrojů
GUID	Globally unique identifier - Globálně jedinečný identifikátor
I4.0	Industry 4.0 - Průmysl 4.0
ID	Identification - Identifikace
IoT	Internet of Things - Internet věcí
LDS ME	Local Discovery Server with Multicast Extension
MQTT	MQ Telemetry Transport - Telemetrický transport MQ
MES	Manufacturing Execution System - Výrobní informační systém
OPC UA	Object Process Control Unified Architecture - Jednotná architektura řízení objektových procesů
UML	Unified Modeling Language - Unifikovaný modelový jazyk
URI	Uniform Resource Identifier - Jednotný identifikátor zdrojů
UUID	Universal Unique Identifier - Univerzální jedinečný identifikátor
RAMI 4.0	Reference Architectural Model Industry 4.0 - Referenční architektonický model Průmyslu 4.0
XML	Extensible Markup Language - Rozšířitelný značkovací jazyk
ZVEI	Zentralverband Elektrotechnik und Elektroindustrie - Asociace německých výrobců elektrických a elektronických produktů

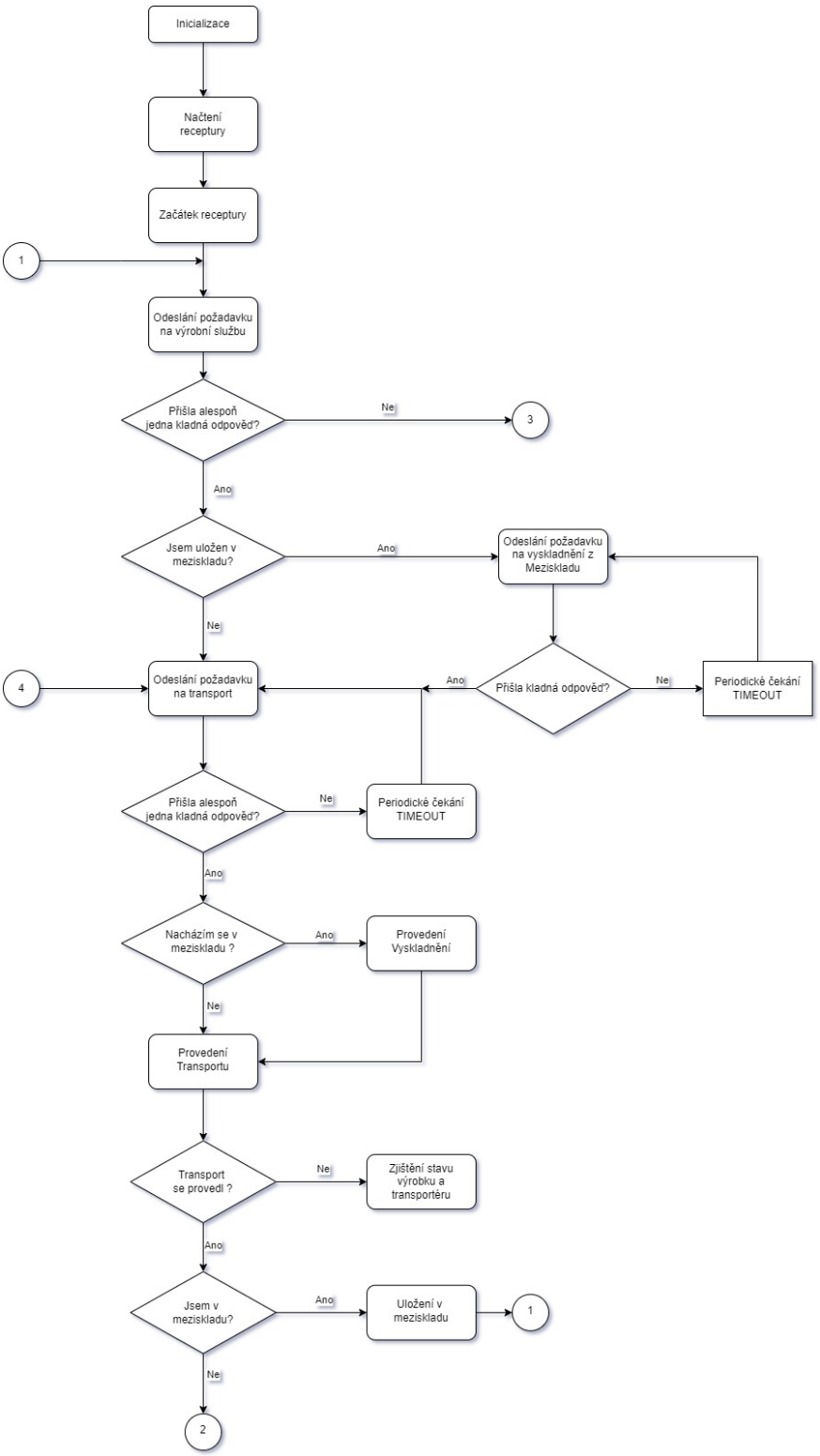
Seznam příloh

A Hlavičkový soubor code.h

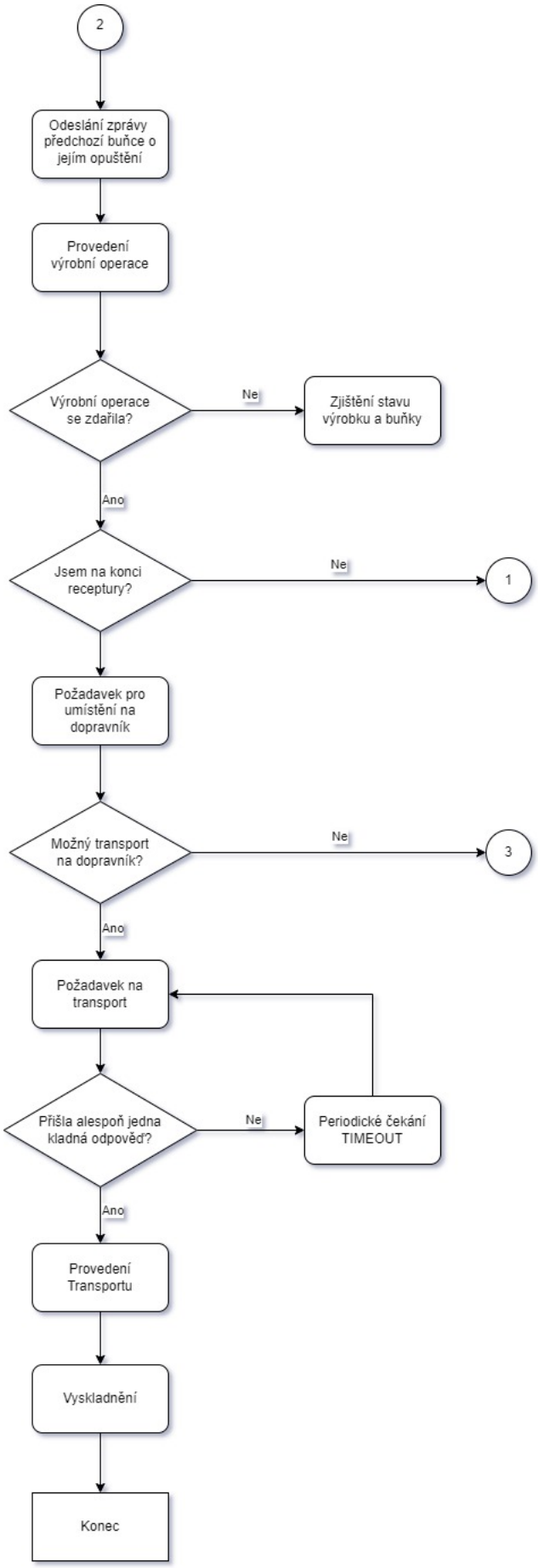
Výpis A.1: Hlavičkový soubor code.h

```
                                                                    1
// Defined variables for message Kind's                             2
const short StorageBid = 1;                                         3
const short WorkBid = 2;                                            4
const short TransportBid = 3;                                       5
const short ResponseBid = 4;                                        6
const short ConfirmBid = 5;                                         7
const short StorageRes = 11;                                        8
const short WorkRes = 12;                                           9
const short TransportRes = 13;                                      10
const short ResponseRes = 14;                                       11
const short ConfirmRes = 15;                                        12
const short ResponseResConf = 16;                                   13
const short Execute = 21;                                           14
const short ResponseExecute = 22;                                   15
const short ConfirmExecute = 23;                                    16
const short ResponseExeConf = 24;                                   17
const short CellLeaved = 31;                                        18
                                                                    19
// Code of cell type                                                20
enum position   {Conveyor, Robot, Storage, Shaker,                  21
                  Warehouse, Nowhere = 99};                         22
                                                                    23
// Code of operation                                                24
enum operation  {Transport, Store, Unstore, Mixing};                25
                                                                    26
// Code of material                                                 27
enum material   {LargeGlass, SmallGlass, Vodka, Rum, Gin,           28
                  Juice, Cola, Tonic, Ice, Soda};                   29
```

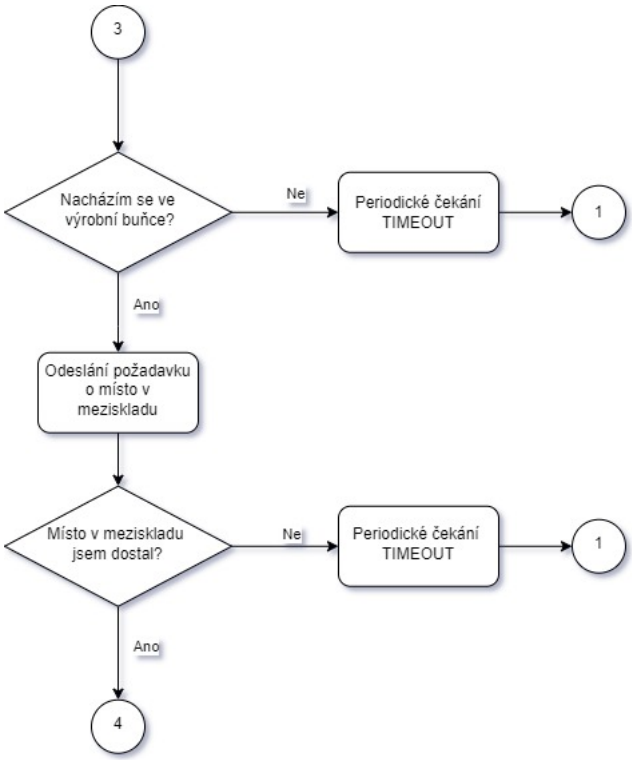
B Stavové diagramy



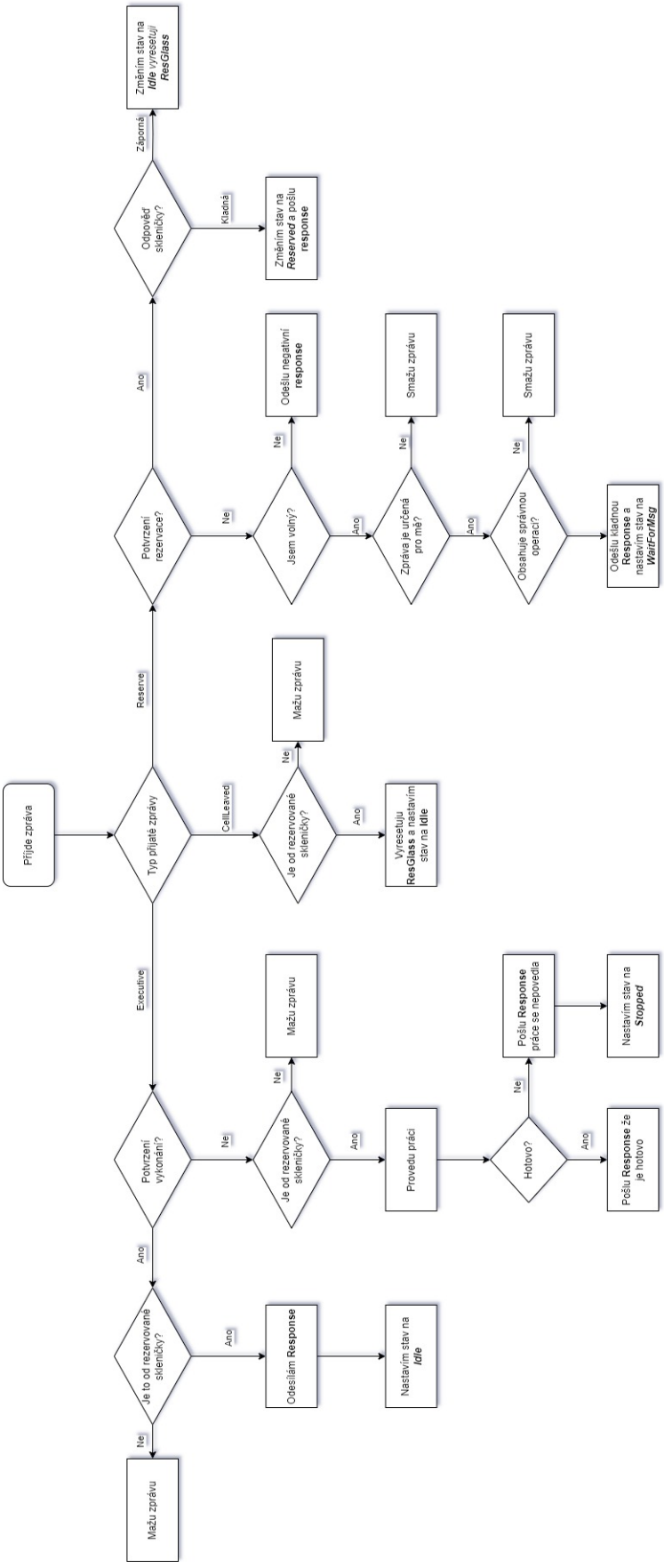
Obr. B.1: Stavový diagram Sklenice část 1.



Obr. B.2: Stavový diagram Sklenice část 2.



Obr. B.3: Stavový diagram Sklenice část 3.



Obr. B.4: Stavový diagram obecného vyřízení zpráv u výrobních buněk.

C Obsah elektronické přílohy

- `/` ..... kořenový adresář přiloženého archivu
  - `barman` ..... Složka se soubory pro simulátor Omnet++
    - `.settings`
    - `out`
    - `simulations` ..... Složka obsahující soubory pro nastavení simulace
      - `results` Složka s výslednými daty po simulaci `.3 src` Složka se zdrojovými soubory pro Simulátor
    - `.cproject`
    - `.nedfolders`
    - `.oppbuildspec`
    - `.project`
    - `Makefile`
  - `diagramy` ..... Složka s vytvořenými diagramy
    - `Stavovy_diagram`
    - `Work_Flow`
  - `AASX` ..... Složka obsahující vytvořené AAS modely