



UNIVERSITY OF NOVI SAD  
FACULTY OF TECHNICAL SCIENCES



**Пристап спецификацији и  
генерисању производних процеса  
заснован на инжењерству вођеном  
моделима**

**A Model-Driven Approach to the  
Production Process Specification and  
Generation**

Ph.D. Thesis

Supervisors:  
Dr. Vladimir Dimitrieski, Assistant Professor  
Dr. Ivan Luković, Full Professor

Ph.D. Candidate:  
Marko Vještica

Novi Sad, 2023

Marko Vještica: *A Model-Driven Approach to the Production Process Specification and Generation*, © 2023

Serbian Title:

Приступ спецификацији и генерисању производних процеса заснован на инжењерству вођеном моделима

Supervisors:

Dr. Vladimir Dimitrieski, Assistant Professor

Dr. Ivan Luković, Full Professor

Location:

Novi Sad

Date:

2023

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА<sup>1</sup>

Врста рада:	Докторска дисертација
Име и презиме аутора:	Марко Вјештица
Ментори (титула, име, презиме, звање, институција)	др Владимир Димитриески, доцент, Универзитет у Новом Саду, Факултет техничких наука др Иван Луковић, редовни професор, Универзитет у Београду, Факултет организационих наука
Наслов рада:	Приступ спецификацији и генерисању производних процеса заснован на инжењерству вођеном моделима
Језик публикације (писмо):	енглески
Физички опис рада:	Унети број: Страница 253 Поглавља 10 Референци 265 Табела 8 Слика 51 Листинзи 21 Графикона 2 Прилога 1
Научна област:	Електротехничко и рачунарско инжењерство
Ужа научна област (научна дисциплина):	Примењене рачунарске науке и информатика
Кључне речи / предметна одредница:	Моделовање производних процеса; Извршавање процеса; Индустрија 4.0; Наменски језици; Трансформације модела; Инжењерство вођено моделима
Резиме на језику рада:	У овој дисертацији представљен је приступ спецификацији и генерисању производних процеса заснован на инжењерству вођеном моделима, у циљу повећања флексибилности постројења у фабрикама и ефикаснијег разрешавања изазова који се појављују у ери Индустрије 4.0. За потребе формалне спецификације производних процеса и њихових варијација у амбијенту Индустрије 4.0, креиран је нови наменски језик, чије моделе рачунар може да обради на аутоматизован начин. Креирани језик има могућност моделовања производних процеса који могу бити независни од производних система и тиме употребљени у различитим постројењима или фабрикама, али и производних процеса који су специфични за одређени систем. Како би моделе производних процеса зависних од конкретног производног система било могуће на аутоматизован начин трансформисати у инструкције које ресурси производног система извршавају, креиран је генератор инструкција. Такође су креирани и генератори техничке документације, који на аутоматизован начин трансформишу моделе производних процеса у документе различитих типова. Употребом предложеног приступа, наменског језика и софтверског решења доприноси се увођењу фабрика у процес дигиталне

<sup>1</sup> Аутор докторске дисертације потписао је и приложио следеће Обрасце:

5б – Изјава о ауторству;

5в – Изјава о истоветности штампане и електронске верзије и о личним подацима;

5г – Изјава о коришћењу.

Ове Изјаве се чувају на факултету у штампаном и електронском облику и не кориче се са тезом.

	<p>трансформације. Како фабрике у ери Индустрије 4.0 морају брзо да се прилагоде новим производима и њиховим варијацијама, неопходно је динамички водити производњу и на аутоматизован начин слати инструкције ресурсима у фабрици, у зависности од производа који се креирају у конкретном постројењу. Тиме што је у предложеном приступу могуће из модела процеса аутоматизовано генерисати инструкције и послати их ресурсима, доприноси се креирању једног динамичког окружења у савременим фабрикама. Додатно, услед великог броја различитих производа и њихових варијација, постаје изазовно одржавати неопходну техничку документацију, што је у предложеном приступу могуће урадити на аутоматизован начин и тиме значајно уштедети време пројектаната процеса.</p>
Датум прихватања теме од стране надлежног већа:	26. 1. 2023.
Датум одбране: (Попуњава одговарајућа служба)	
Чланови комисије: (титула, име, презиме, звање, институција)	<p>Председник: др Соња Ристић, редовни професор, Универзитет у Новом Саду, Факултет техничких наука  Члан: др Миладин Стефановић, редовни професор, Универзитет у Крагујевцу, Факултет инжењерских наука  Члан: др Славица Кордић, ванредни професор, Универзитет у Новом Саду, Факултет техничких наука  Члан, ментор: др Иван Луковић, редовни професор, Универзитет у Београду, Факултет организационих наука  Члан, ментор: др Владимир Димитриески, доцент, Универзитет у Новом Саду, Факултет техничких наука</p>
Напомена:	



**KEY WORD DOCUMENTATION<sup>2</sup>**

Document type:	Doctoral dissertation
Author:	Marko Vještica
Supervisors (title, first name, last name, position, institution)	Dr. Vladimir Dimitrieski, Assistant Professor, University of Novi Sad, Faculty of Technical Sciences Dr. Ivan Luković, Full Professor, University of Belgrade, Faculty of Organizational Sciences
Thesis title:	A Model-Driven Approach to the Production Process Specification and Generation
Language of text (script):	English language
Physical description:	Number of: Pages 253 Chapters 10 References 265 Tables 8 Figures 51 Listings 21 Graphs 2 Appendices 1
Scientific field:	Electrical engineering and computing
Scientific subfield (scientific discipline):	Applied computer science and informatics
Subject, Key words:	Production Process Modeling; Process Execution; Industry 4.0; Domain-Specific Languages; Model Transformations; Model-Driven Engineering
Abstract in English language:	In this thesis, we present an approach to the production process specification and generation based on the model-driven paradigm, with the goal to increase the flexibility of factories and respond to the challenges that emerged in the era of Industry 4.0 more efficiently. To formally specify production processes and their variations in the Industry 4.0 environment, we created a novel domain-specific modeling language, whose models are machine-readable. The created language can be used to model production processes that can be independent of any production system, enabling process models to be used in different production systems, and process models used for the specific production system. To automatically transform production process models dependent on the specific production system into instructions that are to be executed by production system resources, we created an instruction generator. Also, we created generators for different manufacturing documentation, which automatically transform production process models into manufacturing documents of different types. The proposed approach, domain-specific modeling language, and software solution contribute to introducing factories into the digital transformation process. As factories must rapidly adapt to new products and their variations in the era of Industry 4.0, production must be dynamically led and instructions must be automatically sent to factory resources, depending on products that are to be created on the shop floor. The

<sup>2</sup> The author of doctoral dissertation has signed the following Statements:

56 – Statement on the authority,

5B – Statement that the printed and e-version of doctoral dissertation are identical and about personal data,

5r – Statement on copyright licenses.

The paper and e-versions of Statements are held at the faculty and are not included into the printed thesis.

	<p>proposed approach contributes to the creation of such a dynamic environment in contemporary factories, as it allows to automatically generate instructions from process models and send them to resources for execution. Additionally, as there are numerous different products and their variations, keeping the required manufacturing documentation up to date becomes challenging, which can be done automatically by using the proposed approach and thus significantly lower process designers' time.</p>
Accepted on Scientific Board on:	26. 1. 2023.
Defended: (Filled by the faculty service)	
Thesis Defend Board: (title, first name, last name, position, institution)	<p>President: Dr. Sonja Ristić, Full Professor, University of Novi Sad, Faculty of Technical Sciences  Member: Dr. Miladin Stefanović, Full Professor, University of Kragujevac, Faculty of Engineering  Member: Dr. Slavica Kordić, Associate Professor, University of Novi Sad, Faculty of Technical Sciences  Member, supervisor: Dr. Ivan Luković, Full Professor, University of Belgrade, Faculty of Organizational Sciences  Member, supervisor: Dr. Vladimir Dimitrieski, Assistant Professor, University of Novi Sad, Faculty of Technical Sciences</p>
Note:	

## Dedication

*I dedicate this thesis to my family, for their great patience, support, and love.*

*I want to express my gratitude to my mentors Dr. Vladimir Dimitrieski and Prof. Dr. Ivan Luković for their immense help, support, patience, and advice during my Ph.D. studies. I am thankful for their generous efforts in guiding me through the journey of research and writing the Ph.D. thesis.*

*I thank my colleagues from the Data Science and Information Systems group from the Faculty of Technical Sciences who participated in the research presented in this thesis. I am especially grateful to Prof. Dr. Sonja Ristić and Prof. Dr. Slavica Kordić for their great help throughout my research work. Their guidance vastly contributed to the research and enrichment of my knowledge.*

*I give my thanks to KEBA Group AG for their support during my scientific and practical work and for the systematic development of projects. I thank everyone in Linz and Novi Sad who participated in the Digital Factory project. I owe special thanks to Thomas Linde and Milan Pisarić for their huge support in project development and publishing scientific results.*

*I would like to thank Milica Todorović and Maksim Lalić for their help in preparing and analyzing the evaluation questionnaire, as well as all the evaluation participants for their time and valuable feedback. I would also like to thank Milica Ranisavljev for her help in proofreading various publications written during the research related to this thesis.*



# Abstract

Manufacturing is an essential activity of making products and services, with the goal of satisfying customer needs. Over time, manufacturing has become more and more sophisticated, and contemporary manufacturing is mainly done by computer-controlled systems, making a large number of the same or similar products. Such manufacturing, named mass production, is usually inappropriate for creating custom products in small numbers, as it would require changing or rearranging the existing shop floor. With emerging technologies, self-adjustable smart resources, and smart products, the fourth industrial revolution is happening. This industrial revolution, called Industry 4.0, aims to enable the production of numerous individualized products for customers, creating an environment for lot-size-one production while preserving the economic characteristics of mass production. Lot-size-one production would be hard to implement in contemporary production systems, as their rigid structure consisting of assembly lines, fixed robots, and machines cannot support the required production flexibility. Therefore, a new smart environment, consisting of human workers and smart mobile robots, emerges in the context of Industry 4.0 which requires a dynamic and automatic orchestration of production. Consequently, the number of product and process variations increases even more as customers embrace such a dynamic nature of production and thus raise their expectations. Additionally, due to the large number of variations, manufacturing documentation required by different standards and procedures increases as well. Each time a product or its process changes, the documentation of different types must be updated and new versions created. Accordingly, creating and modifying product and process variations and manufacturing documentation becomes a burdensome manual task. Also, as various products and their variations are produced in a factory, human workers must often adapt and create new products. Thus, a fast and automated knowledge transfer is needed on how to produce all these products and variations, as well as guided production to lower the time and costs required for human worker training. Therefore, new solutions are needed to support production flexibility and alleviate such Industry 4.0 challenges. One way to contribute to production flexibility is by applying Model-Driven (MD) principles in production systems that consist of smart resources. The solution we present in this thesis is based on MD principles and utilizes production process models as a single point of knowledge. These production process models are used to automatically generate executable resource instructions that are sent to smart resources, schedule their tasks in a production system, and automatically generate various manufacturing documentation. The main component of our MD solution is a novel Domain-Specific Modeling Language (DSML) named Multi-Level Production Process Modeling Language (MultiProLan), created for the Industry 4.0 domain. Such a language and the whole MD solution are created to contribute to solving the following challenges: (i) automatic and flexible production of individualized products; (ii) error handling during production; (iii) a specification of numerous product and process variations; (iv) a single point of creation and modification of manufacturing documentation in a company; and (v) training and guiding human workers to produce various products. Also, one of the main challenges of Industry 4.0 production process specification is that production process models should be machine-readable, allowing automatic execution of process steps, but also, production process models need to be independent of any production system in which they are to be executed, enabling production process models to

be used in multiple production systems. MultiProLan is a capability-based modeling language, having different levels of detail and modeling layers, allowing production processes to be specified independently of any production system and automatically transformable into executable resource instructions that are sent to the chosen production system. By introducing different levels of detail, process designers can specify production processes more easily, as they do not need to know details related to the specific production system in which production process models are to be executed. In addition, by introducing different modeling layers, process designers can choose aspects they want to be focused on while modeling production processes, lowering the number of modeling concepts they are dealing with. The main goal of the research, presented in this thesis, is to introduce factories into the digital transformation process, contribute to flexible production, and help process designers model production processes and create and modify manufacturing documentation in a more efficient manner. Such a goal is to be achieved by defining a novel methodological approach and a software solution that utilizes MD principles and a DSML to specify production processes formally.

**Keywords:** Production Process Modeling; Process Execution; Industry 4.0; Domain-Specific Languages; Model Transformations; Model-Driven Engineering

## Резиме

Производња представља кључну активност креирања предмета и услуга. Разни алати, машине и материјали користе се у производњи, која постаје све софистициранија услед креирања различитих и све сложенијих производа. Савремена производња примарно се изводи помоћу аутоматизованих машина којима управља рачунар [1,2].

Производња се може посматрати са два различита аспекта – технолошког и економског. Са технолошког аспекта, производња представља примену физичких или хемијских процеса како би се променила својства материјала, креирајући делове и производе. Производња се извршава помоћу низа операција, у којем свака операција доводи материјал корак ближе финалном производу. Са економског аспекта, производња представља прелазак материјала у делове и производе више вредности. Свака операција у производњи мења својства материјала, додајући им вредност.

Циљ производње је да буду задовољене потребе корисника тако што ће произвођачи креирати одговарајуће производе или пружити одговарајуће услуге које могу продати. Како се временом потребе корисника мењају, производи или сервиси морају такође бити континуирано унапређивани. Услед сталних промена, настају различите варијације производа које су груписане у оквиру породица производа. Свака варијација производа из породице производа разликује се од осталих варијација по једном или више коришћених делова или материјала. Такође, свака варијација производа има одговарајућу варијацију производног процеса. Настанком нове варијације производа, постојеће производно постројење у фабрици мора бити измењено или преуређено. То је најчешће изазовно јер су производна постројења и процеси који се у оквиру њих одвијају веома ригидни и скупи за измену.

Уместо честог мануелног преуређивања производног постројења, савремена постројења требало би да буду самоподесива, како би фабрике успешно одговориле на захтеве корисника за персонализованим производима. Све до краја 20. века, машине и даље нису биле потпуно независне и самоподесиве како би се прилагодили варијацијама производних процеса. Измене производних постројења биле су сложен и дуготрајан задатак који је захтевао заустављање производње док се измене не начине. Компанија би губила новац сваки пут када је производња заустављена. Стога, начин производње требало би да буде измењен како би се одговорило на захтеве корисника за великим бројем варијација производа и процеса.

Појава четврте индустријске револуције, назване *Индустрија 4.0*, утиче на измене у начину производње, употребом паметних, независних и самоподесивих ресурса у фабрици. Циљ Индустрије 4.0 је да омогући производњу великог броја различитих, персонализованих производа за кориснике (енгл. *lot-size-one production*), а да се уједно задрже економске карактеристике серијске производње. Међутим, такав начин производње доноси и разне изазове на које је потребно одговорити.

**Проблем истраживања.** Један од главних изазова Индустрије 4.0 представља креирање окружења флексибилне производње у којој је могуће производити различите производе и њихове варијације. Креирањем окружења за производњу великог броја различитих производа подстиче повећање броја варијација производа и процеса, јер корисници све више желе нове, персонализоване производе. Због тога је неопходно динамички и аутоматизовано управљати великим бројем варијација производа у флексибилној производњи фабрике.

Како би фабрика производила велик број различитих производа, неопходно је да производња тече без прекида. У савременим производним постројењима, појава грешке током производње често проузрокује заустављање производне линије како би грешка била отклоњена, правећи додатне трошкове фабрици. Један од изазова Индустрије 4.0 је спровођење јасно дефинисаних активности за отклањање ефеката грешака које настану током производње, а да при томе не дође до заустављања производње.

Додатно, услед великог броја варијација, количина техничке документације у производњи, чије се постојање захтева од стране различитих регулаторних тела и процедура, такође се знатно повећава. Сваки пут када се производ или процес производње измени, документи различитих типова морају бити ажурирани и нове верзије докумената креиране. Тиме, креирање и ажурирање варијација производа и процеса, као и техничке документације, постаје тежак мануелни задатак, што представља један од изазова Индустрије 4.0.

Услед великог броја различитих производа и њихових варијација произведених у фабрици, радници често морају да се прилагоде изменама и креирају нове производе. Такође, како се последњих година број радника у фабрикама смањује, радници често морају да учествују у производњи широког спектра производа. Неопходно је уложити пуно времена у обуку радника за потребе учешћа у изради нових производа. Додатно, искусни радници морају да уложе своје време приликом обуке нових радника, које би иначе уложили у креирање софистицираних производа у фабрици. Због свега наведеног, брз и аутоматизован пренос знања о начину производње нових производа и њихових варијација, као и вођен процес производње, неопходни су како би се смањили време и трошкови потребни за обуку радника, што такође представља један од изазова Индустрије 4.0.

Истраживачи покушавају да одговоре на наведене изазове последњих година, али још увек не постоји одговарајући приступ помоћу којег је могуће у потпуности креирати флексибилан производни систем. Један од начина како одговорити на изазове Индустрије 4.0 и допринети креирању флексибилне производње јесте употреба принципа инжењерства вођеном моделима (енгл. *Model-Driven (MD)*) у којима модели чине референтно место спецификације знања о производњи. Модели производних процеса могли би бити искоришћени да воде извршавање производних процеса на начин да се инструкције генеришу из њих и шаљу ресурсима на извршавање. Међутим, такви модели процеса садрже детаље о производном систему у којем ће бити извршени, што чини моделовање оваквих процеса изазовним, а моделе везује искључиво за један производни систем. У ери Индустрије 4.0, требало би да је могуће модел производног процеса искористити у различитим постројењима или фабрикама. Због тога је један од главних изазова моделовања производних процеса у Индустрији 4.0 да буде омогућено моделовање производних процеса на једноставан начин, да модели процеса могу лако да буду читљиви, разумљиви и искоришћени у различитим производним системима, али и да могу бити аутоматизовано трансформисани у инструкције које ресурси извршавају у одабраном производном систему.

**Предложено решење.** У овој дисертацији предложено је решење за ублажавање или отклањање поменутих изазова Индустрије 4.0, попут аутоматизоване и флексибилне производње персонализованих производа, спецификације великог броја варијација производа и процеса, аутоматизованог отклањања грешака у производњи, јединствене тачке креирања и ажурирања техничке документације у компанији, као и обуке и навођења радника приликом креирања различитих производа. Такво решење засновано је на примени принципа *MD* парадигме и креирања новог наменског језика (енгл. *Domain-Specific Modeling Language (DSML)*) названог Вишениновски језик за моделовање производних процеса (енгл.



*Multi-Level Production Process Modeling Language (MultiProLan)*) у контексту домена Индустије 4.0.

Да би се одговорило изазовима Индустије 4.0, предложено решење користи моделе производних процеса како би било омогућено аутоматизовано вођење извршавања производних процеса. Модели производних процеса трансформисани су у инструкције на аутоматизован начин, а затим такве инструкције ресурси извршавају у производном постројењу. Разни детаљи који се односе на процес производње и извршавања морају бити складиштени у моделима. На тај начин, модели производних процеса постају референтно место спецификације потребног знања у систему. Међутим, складиштењем детаља извршавања унутар модела производних процеса чини моделовање изазовно за пројектанте процеса. Због тога би требало такве моделе процеса представљати кроз различите нивое детаљности. Тиме је могуће раздвојити моделе производних процеса који су независни од производног система (*Master-Level – MasL*), а које моделују пројектанти процеса, од модела производних процеса који су зависни од конкретног производног система у којем ће бити извршени (*Detail-Level – DetL*). Како би било могуће на аутоматизован начин трансформисати *MasL* у *DetL* моделе процеса, неопходно је дефинисати скуп правила трансформисања. Интелигентни систем, попут оркестратора, може да искористи ова правила и формално специфициране моделе процеса, како би извршио трансформацију између модела на аутоматизован начин. Додатно, у оквиру предложеног система и наменског језика, креирани су и различити слојеви моделовања, који приказују или сакривају различите детаље релевантне само за одређене групе корисника.

Управо због тога што подржава различите нивое детаљности и различите слојеве моделовања, *MultiProLan* је и добио назив Вишенивовски језик за моделовање производних процеса, како би се истакла ова његова врло важна особина. *MultiProLan* раздваја *MasL* од *DetL* модела процеса, омогућавајући пројектантима процеса да буду фокусирани искључиво на кораке у процесу производње, уместо на детаље извршавања и на производни систем. Како рачунар може да обради *MultiProLan* моделе на аутоматизован начин, извршиве инструкције могу бити генерисане и послате ресурсима који ће их извршити и креирати производе. Сlike, аудио и видео записи, као и текстуални описи процесних корака такође могу бити генерисани из модела процеса на аутоматизован начин и послати радницима у производњи, како би радници знали на који начин да изврше кораке. Слањем једног по једног процесног корака радницима, укључујући слике, аудио и видео записе, омогућен је вођен процес производње и обука нових радника. Моделовањем потенцијалних грешака које могу да се догоде приликом извршавања производних процеса, као и корака за уклањања ефеката грешака, могуће је такође и разрешавање грешака током производње на аутоматизован начин. Додатно, техничка документација такође може бити генерисана и ажурирана на аутоматизован начин, када год настане нови производ или варијација производа, или се измени постојећи производ или нека његова варијација. На тај начин могуће је уклонити захтеван мануелни посао који обављају пројектанти процеса, чиме им се значајно штеди време.

У наставку резимеа, представљене су хипотезе и циљеви истраживања, очекивани доприноси, као и методологија истраживања, након чега следе резултати и будући правци истраживања.

**Хипотезе истраживања.** Узимајући у обзир наведене изазове и мотивацију истраживања, формирана је основна хипотеза истраживања:

**Хипотеза 0 ( $H_0$ ).** *Могуће је креирати решење, засновано на моделима, за спецификацију модела производних процеса који су независни од производног система и модела производних процеса који су зависни од конкретног производног система, као и решење за трансформисање таквих модела у извршиве инструкције и техничку документацију на аутоматизован начин.*

Главни задатак у истраживању, изведен из основне хипотезе, је дефинисање методолошког приступа и софтверског решења заснованих на *MD* принципима, као и

наменског језика за формалну спецификацију производних процеса, како би се допринело повећању флексибилности производње и одговорило на растуће потребе купаца. Тиме се такође помаже пројектантима процеса приликом моделовања производних процеса и креирања и ажурирања техничке документације. Додатно, формално специфицирани модели производних процеса могу бити обрађени од стране рачунара на аутоматизован начин чиме се омогућава аутоматизовано генерисање извршивих инструкција и техничке документације из модела процеса.

Наредне четири хипотезе изведене су из хипотезе  $H_0$  како би боље били представљени различити аспекти те основне хипотезе. Потврђивањем или одбацивањем изведених хипотеза, биће потврђена или одбачена хипотеза  $H_0$ .

**Хипотеза 1 ( $H_1$ ).** *Могуће је креирати наменски језик за потребе моделовања производних процеса са свим детаљима неопходним да би се на аутоматизован начин генерисале инструкције из модела процеса које су извршиве на ресурсима.*

Производни процеси требало би да буду моделовани на кохерентан и концизан начин употребом наменског језика, чиме модели процеса постају читљиви и обрадиви од стране рачунара. Додавањем неопходних детаља у моделе процеса који су потребни за извршавање производних процеса, могуће је трансформисати такве моделе у инструкције извршиве на ресурсима на аутоматизован начин. Такве инструкције могуће је послати у производно постројење како би биле извршене од стране одговарајућих ресурса.

Модели производних процеса који садрже неопходне детаље за потребе аутоматизоване производње, често морају бити моделовани од стране различитих група корисника, попут инжењера процеса и инжењера квалитета. Такви модели морају укључити различите аспекте и погледе на моделовање производних процеса, попут аспеката извршавања, управљања грешкама, квалитета и сигурности. Различити аспекти и погледи требало би да буду обједињени унутар једног модела процеса, чиме се креира централно место за специфицирање знања о процесима производње, као што је наведено у наредној изведеној хипотези.

**Хипотеза 2 ( $H_2$ ).** *Могуће је представити различите аспекте производних процеса на обједињен начин, независан од производног система, и тиме омогућити поновну употребу модела процеса у различитим производним системима не губећи на читљивости и разумевању модела.*

Модели производних процеса који су спремни за генерисање инструкција постају оптерећени детаљима у вези ресурса који ће извршити процесне кораке, активностима логистике у производњи и активностима конфигурисања машина. Ручно специфицирање таквих модела производних процеса представља изазов. Пројектанти процеса требало би да специфицирају моделе производних процеса независно од конкретног производног система, тиме не узимајући у обзир ресурсе, нити транспортне и конфигурационе активности. Модели процеса независни од производног система могу бити употребљени у било ком производном систему, а такође су једноставни за читање и разумевање.

Ипак, модели процеса независни од производног система не садрже довољно информација како би било могуће трансформисати такве моделе у инструкције извршиве на ресурсима. Због тога је најпре потребно креирати моделе производних процеса зависне од конкретног производног система. То је могуће учинити обогаћивањем модела процеса независног од производног система са детаљима одабраног система у којем ће бити извршен. Такве обогаћене моделе могуће је креирати ручно од стране пројектаната процеса, што би представљало врло захтевну активност, или их је могуће креирати на аутоматизован начин од стране интелигентног система као што је оркестратор. Оркестратор представља софтверско решење које је покренуто на кластеру индустријских рачунара, а које служи да повеже ресурсе и процесне кораке, као и да распореди ресурсе у фабрици на што бољи начин [3–5]. Како би концепти моделовања који се односе на производне системе требало да буду укључени у наменски језик за моделовање производних процеса, пројектанти процеса могу

користити такве концепте да ручно измене или оптимизују моделе производних процеса зависне од конкретног производног система који су настали од стране оркестратора. Потреба да наменски језик садржи концепте који се односе на производне системе како би било омогућено ручно или аутоматизовано обogaћивање модела процеса независних од производног система описана је у следећој хипотези.

**Хипотеза 3 (H<sub>3</sub>).** *Могуће је креирати наменски језик који обухвата концепте моделовања за обogaћивање модела процеса независних од производног система са детаљима о конкретном производном систему, чиме је могуће креирати моделе процеса зависне од конкретног производног система.*

Како би модели производних процеса били независни од конкретног производног система, њихово моделовање могуће је спровести уз помоћ основних концепата инжењерства заснованом на способностима (енгл. *skill-based engineering*) [6,7]. Инжењерство засновано на способностима има за циљ да обједини начин описа способности ресурса. Сваки ресурс нуди скуп способности, док сваки процесни корак захтева одређене способности за извршавање одговарајуће активности. Повезивањем тражених способности у процесним корацима и доступних способности на ресурсима, могуће је ресурсима доделити процесне кораке које би требало да изврше. То је могуће урадити на аутоматизован начин, што је задатак оркестратора. Повезивање ресурса и процесних корака може бити представљено као трансформација модела једног типа у модел другог типа. Односно, може бити представљено као аутоматизована трансформација модела процеса независних од производног система у моделе процеса зависне од конкретних производних система.

Поред повезивања ресурса и процесних корака, оркестратор би требало да дода транспортне и конфигурационе кораке такође на аутоматизован начин, а на основу топологије и логистике производног система. Оркестратор са својим алгоритмима за повезивање и расподелу ресурса није део овог истраживања и коришћен је као готово софтверско решење, без улажења у детаље његове имплементације. Више информација о оркестратору могуће је пронаћи у претходном истраживању у којем је учествовао аутор дисертације [4,5]. Аутоматизованим обogaћивањем модела процеса независних од производних система, знатно се олакшава задатак пројектаната процеса, који могу да креирају моделе процеса независне од производних система и на једноставан начин их припреме за аутоматизовано генерисање инструкција. Како би производни процеси били извршени, њихови модели би требало да буду трансформисани у инструкције извршиве на ресурсима, што је описано у наредној хипотези.

**Хипотеза 4 (H<sub>4</sub>).** *Могуће је на аутоматизован начин генерисати инструкције извршиве на ресурсима и техничку документацију различитих типова из модела производних процеса.*

Употребом генератора инструкција са правилима трансформисања из модела у текст, модел производног процеса зависног од одабраног производног система могуће је трансформисати на аутоматизован начин у инструкције извршиве на ресурсима. Генерисане инструкције могу бити послате ресурсима да их изврше, чиме се испуњава основна сврха креирања модела производних процеса – да омогуће аутоматизовано вођење извршавања производних процеса у производном систему. Како су рачунари у могућности да интерпретирају моделе процеса на аутоматизован начин, техничка документација такође може бити генерисана из модела процеса. Ова трансформација помаже пројектантима процеса да елиминирају ручно креирање и ажурирање техничке документације. Услед аутоматизованог генерисања техничке документације, могуће је уштедети време пројектаната процеса, док грешке настале приликом ручног прављења документације могуће је избећи.

**Циљ истраживања.** Ова дисертација требало би да омогући формулацију таквог приступа који ће у ери Индустије 4.0 обезбедити динамичко управљање производњом и аутоматизовано извршавање производних процеса на основу модела процеса. Обезбеђењем динамичког управљања производњом и аутоматизованог извршавања производних процеса

могуће је допринети увођењу фабрика у процес дигиталне трансформације, што представља главни циљ спроведеног истраживања. Главни циљ истраживања могуће је остварити достизањем следећих критичних фактора успеха:

- Обезбедити да један модел производног процеса може бити употребљен у више различитих производних система, креирањем модела производних процеса који су независни од производног система. Тиме би пројектанти процеса били у могућности једноставније да моделују процесе производње, без потребе да поседују знање о детаљима производног система у којем би процес био извршен.
- Ослободити пројектанте процеса мануелног задатка креирања модела процеса специфичних за производних систем, односно обезбедити трансформисање модела производног процеса независног од производног система у модел производног процеса зависног од конкретног производног система на аутоматизован начин. Ипак, потребно је пружити могућност ручне измене оваквих модела када пројектанти процеса процене да постоји потреба за тим.
- Модел производног процеса зависан од производног система садржи у себи детаље попут паметних ресурса који ће извршити кораке производње, активности логистике у производном систему, као и активности конфигурисања машина. Обезбедити да такви модели процеса аутоматизовано воде процес производње тиме што би модели процеса били трансформисани у извршиве инструкције на аутоматизован начин.
- Омогућити откривање и отклањање лоше моделованих процеса и грешака које настану током производње помоћу механизма за праћење извршавања процеса производње.
- Обезбедити да различите групе корисника могу бити фокусиране на аспекте за које су одговорни, креирајући различите погледе на модел производног процеса употребом различитих слојева моделовања. На тај начин, модел производног процеса представља референтно место спецификације потребног знања, а различите групе корисника могу заједно да учествују у специфицирању различитих аспеката производног процеса. Додавањем или уклањањем слојева модела производног процеса, могуће је приказати или сакрити различите аспекте модела, чиме они постају јаснији и читљивији.
- Ослободити пројектанте процеса мануелног одржавања великог броја докумената различитих типова у фабрици, на начин да техничка документација буде аутоматизовано генерисана и ажурирана трансформисањем модела производних процеса у документацију.
- Омогућити да све варијације производа и процеса буду доступне у једном моделу производног процеса. Како је могуће да постоји велик број варијација у моделу процеса, што чини модел тешким за читање и одржавање, потребно је омогућити и механизме за одабир варијације на коју би пројектант процеса био фокусиран.
- Обезбедити аутоматизовану помоћ новим радницима приликом обуке и креирања производа и смањити време које искусни радници морају да уложе у обуку нових радника. То је могуће постићи тиме што би модели производних процеса били употребљени за вођен процес производње, слањем новим радницима једне по једне инструкције са сликама и видео записима, насталим из модела процеса.

**Очекивани доприноси.** Како би биле потврђене или одбачене наведене хипотезе и достигнут дефинисани циљ, спроведено је истраживање које је описано у овој дисертацији. Као резултат истраживања, очекују се доприноси следећих типова:

- **Теоријски доприноси** у области моделовања производних процеса и њиховог аутоматизованог извршавања:
  - Преглед постојећих језика за моделовање производних процеса.
  - Идентификација основних концепата неопходних за креирање наменског језика за моделовање производних процеса у контексту Индустије 4.0, а чији су модели погодни за генерисање инструкција и техничке документације на аутоматизован начин.

- Спецификација новог *MD* приступа за потребе динамичке производње и аутоматизованог генерисања инструкција на основу модела производних процеса насталих употребом наменског језика.
- Спецификација методологије за аутоматизовану трансформацију модела независних од производног система у моделе зависне од производних система.
- Обезбеђена могућност примене *MD* принципа у домену производње, чиме би се допринело једноставнијем моделовању производних процеса и њиховом аутоматизованом извршавању.
- **Доприноси развоју** софтверске подршке за моделовање производних процеса и генерисање инструкција извршивих на ресурсима и техничке документације:
  - Развијен и имплементиран нови софтверски алат који подржава употребу креираног наменског језика за моделовање производних процеса.
  - Развијен и имплементиран генератор инструкција за генерисање инструкција извршивих на ресурсима из модела наменског језика на аутоматизован начин.
  - Развијен и имплементиран генератор техничке документације за генерисање различите документације из модела наменског језика на аутоматизован начин.
- **Доприноси примени** *MD* приступа и наменског језика у домену производње, а у виду студија случаја и анализе и оцене језика и алата за моделовање производних процеса:
  - Показана практична примена *MD* приступа и наменског језика за моделовање производних процеса у монтажној индустрији.
  - Извршена анализа и оцена новог софтверског алата и наменског језика за моделовање производних процеса од стране различитих група корисника.
  - Презентовано ново практично искуство, остварено применом предложеног методолошког приступа, софтверског алата и подржаног наменског језика.
- **Друштвени доприноси** представљају могућност стављања на јавну употребу једног општег модела управљања производним процесом, који је применљив у организацијама ширег спектра, на начин да омогућава значајније унапређење производног процеса и подизање општег кумулираног знања о томе како се на савремен начин може допринети том процесу унапређења.

Остварењем очекиваних резултата истраживања постиже се једноставније и прецизније моделовање производних процеса, чији модели омогућавају примену принципа флексибилне производње и тиме обезбеђују кретање организације у смеру пуне примене филозофије Индустрије 4.0. Тиме организација повећава степен способности да боље одговори на све интензивније захтеве купаца за високо персонализованим производима. Очекивани корисници алата и језика за моделовање производних процеса су пројектанти процеса, које чине инжењери процеса и инжењери квалитета.

**Методологија истраживања.** Истраживање описано у овој дисертацији спроведено је кроз наредних шест активност:

- Идентификација изазова Индустрије 4.0 у постојећој производњи и мотивација за пружање доприноса у решавању идентификованих изазова.
- Дефинисање циљева предложеног *MD* приступа, наменског језика и алата за моделовање производних процеса.
- Пројектовање и развој софтверског решења за спецификацију и извршавање производних процеса, које обухвата наменски језик *MultiProLan*, његов алат за моделовање производних процеса и различите генераторе инструкција и техничке документације.
- Примена софтверског решења у две студије случаја монтажне индустрије.
- Анализа и оцена наменског језика *MultiProLan* и његовог алата за моделовање производних процеса од стране различитих група корисника.
- Представљање резултата истраживања академској заједници на међународним научним конференцијама и у часописима [4,5,8–19], као и релевантним корисницима у индустрији.

У наставку резимеа, сумирани су резултати и будући правци истраживања. Најпре су представљени резултати прегледа актуелног стања у области, опис имплементираних решења, студије случаја, анализа и оцена имплементираних решења и будући правци истраживања.

**Преглед актуелног стања у области.** Истраживања у области моделовања производних процеса, посебно у Индустрији 4.0, све су интензивнија последњих година [20]. На то је утицала чињеница да производни процеси морају бити дигитално подржани у Индустрији 4.0 [21], како би било могуће интегрисати их у оквиру паметних фабрика. Процеси би требало да буду креирани са виртуелном репрезентацијом која захтева апстрактно размишљање и моделовање помоћу специјализованог софтвера [22] – алата за моделовање. Моделовање производних процеса у Индустрији 4.0 је изузетно важна информатичка истраживачка тема, јер је од великог значаја разумети и оптимизовати процесе [23]. Међутим, није довољно само документовати производне процесе и чувати документацију у бази података фабрике. Ова дисертација даје допринос све присутнијем захтеву да производни процеси буду моделовани тако да аутоматизовано воде процес производње, односно да буду спремни за аутоматизовано извршавање или генерисање извршивих инструкција, али да уједно не буду превише комплексни, како би их човек могао прочитати и разумети, и да буду независни од конкретног производног постројења.

У истраживању описаном у овој дисертацији, испитани су различити постојећи језици и приступи за моделовање процеса, а који су потенцијално одговарајући за предложено *MD* решење и тиме могу да допринесу решавању изазова Индустрије 4.0. Приликом иницијалног прегледа литературе, формиран су захтеви које би један језик за моделовање процеса требало да испуни, а чији модели би били коришћени за динамичку оркестрацију и аутоматизовано генерисање извршивих инструкција у ери Индустрије 4.0. Захтеви су детаљно представљени у поглављу 4, док је у наставку представљен њихов кратак опис:

- **Захтев 1:** постојање процесних корака који обухватају улазне производе над којима је потребно извршити одговарајућу активност, излазне производе настале применом одговарајућих активности над улазним производима, способност коју је потребно поседовати за извршавање процесног корака и ресурс који би требало да изврши процесни корак, а поседује тражену способност.
- **Захтев 2:** постојање контроле тока активности, као што су секвенца, одлука, итерација и паралелизам.
- **Захтев 3:** постојање тока материјала, односно знања да ли је одређени производ потребно преузети из складишта или је резултат неког од претходних процесних корака.
- **Захтев 4:** постојање тока порука, односно сарадње између процесних корака или ресурса разменом порука.
- **Захтев 5:** постојање неуређених процесних корака који могу бити извршени било којим редоследом.
- **Захтев 6:** постојање варијација производа и процеса, где варијације производа припадају истој породици производа које се разликују по одређеним карактеристикама, а варијације процеса представљају разлике у начину извршавања процесних корака.
- **Захтев 7:** постојање потпроцеса, чиме је могуће смањити комплексност модела процеса и искористити исте процесне кораке у различитим процесима.
- **Захтев 8:** постојање управљања грешкама које настану током производње, а на које је потребно одговорити одређеним активности за опоравак од негативних ефеката насталих грешака.
- **Захтев 9:** модели производних процеса требало би да буду извршиви или би требало генерисати извршиве инструкције из модела.
- **Захтев 10:** модели производних процеса требало би да буду независни од конкретног производног система.

Наведени захтеви коришћени су за систематичну анализу постојећих језика и приступа за моделовање процеса, у којој је сваки пронађени језик или приступ тестиран да ли испуњава захтеве. Проучавањем литературе идентификовано је више језика и приступа за моделовање производних процеса, који су подељени у следеће четири категорије:

- **Категорија 1: Традиционални приступи спецификацији производних процеса (К1)** [24–29]. Ови приступи најчешће користе неформалан текстуални опис, дијаграме токова или табеларну форму записа како би били представљени разни аспекти производних процеса. Такве форме записа коришћене су годинама.
- **Категорија 2: Језици за моделовање процеса који нису примарно креирани за потребе производних процеса и њихова проширења (К2)** [30–57]. Ови језици представљају другу по величини категорију, у којој *Business Process Model and Notation (BPMN)* са својим проширењима чини језик који је најчешће коришћен и најчешће прошириван за потребе описа производних процеса.
- **Категорија 3: Комбинација различитих језика за потребе моделовања производних процеса (К3)** [58–62]. У овој категорији пронађен је најмањи број приступа и језика. Аутори ових приступа су комбиновали различите језике како би моделовали различите аспекте производних процеса.
- **Категорија 4: Језици креирани да подрже моделовање производних процеса или производних система (К4)** [63–92]. Ова категорија језика је најзаступљенија, у којој су језици посебно креирани за домен производње. Већина ових језика представља наменске језике за моделовање.

Након спроведеног истраживања, примећено је да постоји растући тренд последњих година у броју научних радова објављених на тему моделовања процеса. На основу прикупљене литературе, примећена је доминација радова друге категорије (К2) све до 2019. године. Разлог томе је што је једноставније проширити постојеће језике, него правити нови наменски језик од почетка. Међутим, такви језици нису примарно креирани за домен моделовања производних процеса и било би потребно користити више различитих проширења језика како би се моделовали производни процеси за потребе аутоматизованог извршавања, што би значајно отежало задатак пројектаната процеса. Како је домен моделовања производних процеса комплексан, а самим тим је и комплексност модела производних процеса велика, истраживачи све више креирају нове наменске језике. То је примећено на основу прикупљене литературе у којој број радова друге категорије (К2) опада након 2019. године, док број радова четврте категорије (К4) расте након 2019. године. Овај растући тренд може бити проузрокован услед потребе да се модели производних процеса изврше или да се генеришу инструкције из модела, као и техничка документација. Стога су важни формални језици чије моделе рачунар може да обради на аутоматизован начин. Истраживачи све више времена улажу у креирање нових, формалних језика од почетка, који ће успети да се супротставе комплексностима насталим унутар домена производних процеса. Међутим, креирани формални језици често моделују процесе на високом нивоу, без детаља извршавања. Чак и када су укључени детаљи извршавања процеса, модели су зависни од производног система, због чега не би било могуће употребити их у различитим системима. Други начин одговора на комплексност домена јесте употреба различитих језика како би се моделовали различити аспекти производних процеса. Ипак, овакав комбиновани приступ (К3) је присутан у свега неколико радова од 2015. године. Овај релативно мали број приступа вероватно је проузрокован чињеницом да је потребно додатно знање које пројектанти процеса морају да имају како би познавали различите језике и активно их користили, што чини задатак моделовања процеса изазовним. Прва категорија (К1) језика и приступа настала је знатно раније и представља традиционалне начине описа производних процеса. Ови приступи и језици обухватају најчешће неформалан или табеларан приказ производних корака, а у случајевима дијаграма токова, процесни кораци представљају само графичке симболе, без формалне семантике. Због тога би такве спецификације производних процеса било тешко обрадити на аутоматизовани начин и генерисати инструкције из њих.

Област моделовања производних процеса у контексту Индустије 4.0 још увек је недовољно истражена. Већина језика и приступа не обезбеђује могућност креирања модела процеса погодних за динамичку производњу и аутоматизовано извршавање или генерисање извршивих инструкција. У случајевима када језици подржавају аутоматизовано извршавање модела процеса, такви модели садрже техничке детаље изведбе сервиса, попут адреса сервиса и бројева портова. На тај начин, значајно се отежава рад пројектаната процеса, јер они морају да познају детаље постројења у фабрици у којој ће процес бити извршен. Задатак пројектаната процеса је да специфицирају процесне кораке и да не брину о детаљима у вези производног постројења и извршавања. Такође, полазећи од доминантних ставова у литератури, моделе производних процеса рачунар би требало да прочита на аутоматизован начин, чиме се омогућава аутоматизовано извршење процесних корака описаних у моделу, али би модели процеса требало уједно да буду формиран тако да су независни од конкретног постројења у којем ће бити извршени, како би било могуће исти модел процеса извршити у различитим постројењима [65]. Креирање модела производних процеса које рачунар може да обради и генерише инструкције из њих, а који су такође независни од производног постројења, представља један од главних проблема моделовања производних процеса у ери Индустије 4.0.

**Наменски језик *MultiProLan*.** Услед претходно наведених изазова и немогућности проналаaska одговарајућег језика приликом прегледа литературе, креиран је нови језик за моделовање производних процеса. Модели новог језика садрже довољно детаља потребних за динамичку производњу и аутоматизовано генерисање извршивих инструкција, а језик омогућава пројектантима процеса да моделују процесне кораке без детаља извршавања. Тиме је такође омогућено да модели процеса могу бити употребљени у различитим производним постројењима. У том циљу, креирани су и одговарајући софтверски сервиси који на аутоматизован начин трансформишу моделе процеса који су независни од производног постројења, у моделе који су зависни од њега и спремни за извршавање.

Како би био креиран нови наменски језик, потребно је најпре анализирати домен у којем ће језик бити примењен. Доменско знање за потребе моделована производних процеса прикупљено је током прегледа литературе, кроз техничку документацију, разне студије случаја, као и кроз разговоре са експертима из домена. Након прикупљеног неопходног знања из домена производних процеса, концепти домена су структурирани и представљени помоћу језика *Feature-Oriented Domain Analysis (FODA)* [93], чији су модели захтева описани у поглављу 6.

Креирани наменски језик *MultiProLan*, представљен у овој дисертацији, имплементиран је на основу концепата из домена представљених у моделу *FODA*, а такође испуњава претходно дефинисане захтеве. Мета-модел овог језика, односно његова апстрактна синтакса, имплементирана је помоћу мета-мета-модела *Ecore*, који је део радног оквира *Eclipse Modeling Framework (EMF)* [94,95]. Додатна ограничења која не могу бити исказана помоћу концепата мета-мета-модела, а постоје у домену моделовања производних процеса, имплементирана су уз помоћ језика *Object Constraint Language (OCL)* [96–98]. На основу креиране апстрактне синтаксе, направљена је конкретна графичка синтакса као и прототип алата, користећи радни оквир *Eclipse Sirius* [99,100]. *MultiProLan* првенствено је намењен домену монтажне индустрије, али може бити употребљен и у другим доменима производње. Пројектанти процеса користе овај језик како би моделовали производне процесе чији су модели:

- погодни за аутоматизовано генерисање извршивих инструкција,
- независни од конкретних производних система,
- коришћени за руковање грешкама које настану током производње,
- садрже велик број варијација производа и процеса, и
- погодни за аутоматизовано генерисање и ажурирање техничке документације.

Постоје четири основне карактеристике језика *MultiProLan* које га разликују од осталих језика анализираних током прегледа литературе:



- Основни скуп концепата потребан за спецификацију модела производних процеса који су погодни за динамичку производњу и аутоматизовано генерисање извршивих инструкција.
- Два нивоа детаљности која дозвољавају дистинкцију модела производних процеса који су независни од производног система од модела производних процеса који су зависни од конкретног производног система.
- Основни скуп концепата потребних за управљање грешкама.
- Обједињавање концепата различитих типова техничке документације у један унифицирани модел производног процеса, што омогућава аутоматизовано генерисање и ажурирање техничке документације из модела процеса.

Поред наведених предности и особина језика *MultiProLan*, постоје и неколико претпоставки које су уведене, као и ограничења којих пројектант процеса мора бити свестан приликом моделовања производних процеса:

- Производни процес може бити моделован за извршавање у једном производном постројењу или у једној паметној фабрици. Није могуће моделовати извршавање модела процеса у више различитих фабрика истовремено, као ни њихову колаборацију.
- Постојећи ресурси у фабрици који извршавају моделоване процесне кораке довољно су паметни да разумеју основне инструкције на високом нивоу, као што су *покупи, постави, крећи се и састави*.
- Паметни ресурси представљају се систему приликом њиховог увођења и пружају све потребне информације и семантику на који начин могу да се користе.
- Складишта у производном постројењу попуњена су материјалима и деловима који не могу бити потрошени.

Детаљан опис апстрактне синтаксе, ограничења, конкретне синтаксе, алата за моделовање, као и начин како *MultiProLan* испуњава претходно дефинисане захтеве, представљен је у поглављу 7.

**Студије случаја новог софтверског решења.** Како су представљено *MD* решење и језик *MultiProLan* првенствено креирани за дискретну производњу, посебно за монтажну индустрију, примењени су у студијама случаја монтажне производње. Истраживачи често тестирају своја решења управо у монтажној индустрији јер пружа могућност креирања релативно једноставних производних процеса, али и врло комплексних. Стога је монтажна индустрија добар почетни корак како би ново решење било тестирано. Припремљене су две студије случаја за потребе валидације *MD* решења и језика *MultiProLan*, презентоване у овој дисертацији у поглављу 8.

Прва студија случаја креирана је како би биле демонстриране могућности језика *MultiProLan* и алата за моделовање производних процеса. У овом примеру састављана је дрвена кутија, чији модел производног процеса садржи концепте попут различитих варијација производа, сарадње између ресурса и паралелног извршавања процесних корака. Такође, у овој студији случаја представљени су и примери генерисаних инструкција и техничке документације. Друга студија случаја представља показно окружење креирано како би било тестирано комплетно *MD* решење приликом склапања објеката од *LEGO*<sup>®</sup> коцкица, чиме су избегнути трошкови који би настали услед грешака приликом склапања реалног производа у производном постројењу. Поједини работи коришћени у овој студији случаја представљају индустријске покретне роботe, који се користе у реалној производњи, као и истраживачке паметне роботe коришћене у показне сврхе. Радници такође учествују у овој студији случаја приликом склапања *LEGO*<sup>®</sup> коцкица.

**Анализа и оцена језика *MultiProLan* и алата за моделовање производних процеса.** Поред две студије случаја у којима је показана употреба целокупног *MD* решења, са акцентом на језик *MultiProLan*, извршена је такође анализа и оцена језика *MultiProLan* и алата за моделовање производних процеса, детаљно описане у поглављу 9. Различите групе

корисника учествовале су у процесу анализе језика и алата, укључујући инжењере процеса, инжењере софтвера, истраживаче и студенте. Користећи радни оквир *Framework for Qualitative Assessment of Domain-specific languages (FQAD)* [101], учесници су на систематичан начин извршили анализу и оцену језика *MultiProLan* и његовог алата за моделовање производних процеса и оставили важне повратне информације. Направљен је експеримент у којем су учесници тестирали *MultiProLan* и алат за моделовање процеса и оцењивали следеће карактеристике квалитета: функционалну комплетност, употребљивост, поузданост, изражајност и продуктивност. Поред ових карактеристика, аутор дисертације са коауторима извршио је оцену карактеристика квалитета која захтевају имплементациона знања, а то су: могућност одржавања алата, проширивост, поновна употреба и могућност интеграције са другим језицима. Постављена је следећа хипотеза на коју је било потребно одговорити након извршеног експеримента:

**$H_{null}$**  – *MultiProLan* може бити употребљен у пракси јер има све следеће карактеристике квалитета: функционалну комплетност, употребљивост, поузданост, изражајност и продуктивност.

**$H_{alt}$**  – *MultiProLan* не може бити употребљен у пракси јер нема једну или више од следећих карактеристика квалитета: функционалну комплетност, употребљивост, поузданост, изражајност или продуктивност.

Укупно 25 учесника било је укључено у експеримент и дали су своје оцене на претходно наведене карактеристике квалитета. Учесници су најпре имали задатак да направе један модел производног процеса, након чега су попуњавали анкету. Анкета је садржала питања која су се односила на претходно искуство учесника и на представљене карактеристике квалитета, као и секцију за слободан коментар. Сва питања, осим секције са коментаром, садржала су понуђене одговоре на Ликертовој скали од 1 до 5. Резултати анкете детаљно су описани у поглављу 9, а сумирани резултати који се односе на проценат позитивних одговора је следећи: функционална комплетност (98,00%), употребљивост (88,50%), поузданост (76,00%), изражајност (89,33%) и продуктивност (68,00%). На основу добијених резултата, могуће је потврдити хипотезу  $H_{null}$ , и тиме закључити да *MultiProLan* поседује представљене карактеристике квалитета. Међутим, на основу коментара које су учесници оставили, *MultiProLan* захтева додатна побољшања. Секција у анкети која се односила на слободне коментаре била је врло значајна како би били откривени недостаци језика и алата које би требало уклонити.

**Закључак и будући правци истраживања.** На основу резултата презентованих у овој дисертацији, а који првенствено обухватају ново *MD* решење са новим методолошким приступом и наменским језиком *MultiProLan* за моделовање и извршавање производних процеса, у поглављу 10 дискутовано је да ли су хипотезе дефинисане у уводним поглављима дисертације потврђене или одбачене. Како је креиран наменски језик *MultiProLan*, чији модели процеса учествују у динамичкој производњи и погодни су за аутоматизовано генерисање извршивих инструкција и техничке документације, а такође је омогућено представити моделе процеса независно од производног система и по потреби их допунити са елементима конкретног производног система, могуће је потврдити хипотезе  $H_1$ ,  $H_2$ ,  $H_3$  и  $H_4$ . Тиме је такође могуће потврдити и хипотезу  $H_0$ , а на тај начин испуњен је главни задатак овог истраживања и могуће је потврдити остварење главног циља, односно доприноса увођења фабрика у процес дигиталне трансформације.

У поглављу 10 такође су детаљно описани сви доприноси истраживања представљеног у овој дисертацији, као и будући правци истраживања. Они се деле на три општа правца будућег истраживања:

- Будуће истраживање у области моделовања производних процеса и производних система која обухватају:
  - Истраживање у области моделовања производних система, односно ресурса, њихових способности и ограничења, како би модел производног процеса могао

- на бољи начин да буде извршен, тиме што ће инструкције бити послате одговарајућим ресурсима.
- Истраживање у области моделовања радника у фабрици са истим разлогом споменутим у претходној ставци, међутим, раднике је потребно моделовати са знатно више детаља, попут њихових компетенција, правних и здравствених ограничења и улога у фабрици.
  - Истраживање у областима моделовања ризика по раднике приликом извршавања процесних корака, као и сигурносних аспеката, чиме би инструкције у случајевима који су ризични по човека биле послате другим ресурсима.
  - Истраживање у области моделовања утрошка енергије приликом извршавања производних процеса, како би била обезбеђена одрживост фабрике тиме што би инструкције биле послате ресурсима који би уз мањи утрошак енергије извршили задатак.
  - Истраживање у области моделовања колаборативних производних процеса, у којима је производни процес дељен и извршаван у различитим фабрикама, стога је потребно обезбедити поверење између фабрика и гарантовати да ће сви услови производње бити задовољени.
  - Истраживање у области аутоматизоване трансформације модела производа, попут модела *Computer Aided Design (CAD)*, у модел производног процеса, што би омогућило да се смањи време потребно за моделовање производних процеса.
  - Будући развој језика *MultiProLan* и алата за моделовање производних процеса који обухвата:
    - Развој система препоруке који би на основу модела производног процеса независног од производног система вршио процену времена производње и утрошка енергије у различитим производним системима, и препоручивао у којем систему је најпогодније извршити производни процес.
    - Након извршене стандардизације способности и параметара у будућности, а којима се описују ресурси у производњи и производни процеси, потребно је прилагодити тренутни репозиторијум способности стандардизованим, како би се обезбедила компатибилност са другим системима и решењима.
    - Проширење језика *MultiProLan* концептима који би обезбедили моделовање квалитета производње, чиме би било могуће повећати квалитет производње.
    - Проширење језика *MultiProLan* концептима који би обезбедили моделовање комплексније сарадње између ресурса приликом извршавања разних задатака.
    - Креирање текстуалне конкретне синтаксе језика *MultiProLan* и анализирање да ли је текстуална синтакса боља за пројектанте процеса у односу на графичку синтаксу.
    - Унапређење језика *MultiProLan* и алата за моделовање на основу коментара које су оставили учесници анализе и оцене језика и алата, као и проширење анализе и оцене језика и алата са новим учесницима, посебно инжењерима процеса.
  - Нови домени примене језика *MultiProLan* који обухватају:
    - Примену језика *MultiProLan* у процесној индустрији, односно континуалној производњи, што би захтевало увођење нових концепата моделовања, попут мерача времена и процене времена извршавања корака производње.
    - Интеграцију представљеног *MD* решења са системом за аутоматизовану детекцију извршавања процесних корака, како радници у фабрици не би морали користити уређај са којим би потврдили да ли је процесни корак успешно завршен и тиме губили значајно време. На тај начин додатно се убрзава обука радника кроз систем за вођен процес производње, а такође се спречава да радници забораве да изврше неки процесни корак.
    - Примену језика *MultiProLan* у проширеној или помешаној реалности, у којој би инструкције биле послате на одговарајући уређај коришћен од стране радника у фабрици. Тиме би постојала могућност бржег и једноставнијег извршавања процесних корака, посебно приликом обуке нових радника.

- Примену других научних поља, попут науке о подацима и рударења процеса над подацима прикупљеним током извршавања производних процеса, чиме би потенцијално биле отклоњене аномалије, недостаци, грешке и кораци који успоравају извршавање производних процеса.

Предложено *MD* решење, које је представљено у овој дисертацији, могуће је даље унапређивати и развијати како би додатно подржало флексибилну производњу у индустрији. Тиме би пун потенцијал овог решења био откривен, а такође би настале и примене у новим доменима, подржавајући компаније у ери дигиталне трансформације и Индустрије 4.0.

**Кључне речи:** Моделовање производних процеса; Извршавање процеса; Индустрија 4.0; Наменски језици; Трансформације модела; Инжењерство вођено моделима

# Acknowledgments

The following projects supported the research presented in this thesis:

- "Intelligent Systems for Software Product Development and Business Support Based on Models", III-44010, *National Project*, Ministry of Education, Science and Technological Development of Republic of Serbia.
- "Innovative scientific and artistic research from the FTS (activity) domain", 451-03-68/2020-14/200156, 451-03-68/2021-14/200156, 451-03-68/2022-14/200156, 451-03-47/2023-01/200156, *National Project*, Ministry of Science, Technological Development and Innovations of Republic of Serbia.
- "Digital Factory", *Industrial R&D Project*, KEBA Group AG, Linz, Republic of Austria.



# Contents

List of Figures .....	xxx
List of Graphs .....	xxxiii
List of Tables .....	xxxv
List of Listings .....	xxxvii
List of Acronyms .....	xxxix
1 Introduction.....	1
1.1 Research Challenges, Motivation, Hypothesis and Goal .....	2
1.2 Overview of the Proposed Solution .....	3
1.3 Research Contributions and Results .....	4
1.4 Thesis Structure .....	5
2 Background and Theoretical Foundation.....	7
2.1 Relevant Concepts and Technologies in Industry 4.0.....	8
2.2 Process Planning and Design .....	9
2.2.1 Production Process Planning .....	9
2.2.2 Production Process Design and Skill-Based Engineering .....	10
2.3 Overview of the MD Paradigm and DSMLs.....	12
2.4 Summary .....	15
3 Motivation, Research Hypotheses, Goals and Methodology .....	17
3.1 Motivation.....	17
3.2 Research Hypotheses .....	19
3.3 Research Goals .....	21
3.4 Expected Contributions and Results.....	22
3.5 Research Methodology .....	23
3.6 Summary .....	26
4 State-of-the-Art .....	27
4.1 Application of the MD Paradigm and DSLs .....	27
4.1.1 Information Systems .....	28
4.1.1.1 Information System Design Based on the IIS*Case Approach .....	28
4.1.1.2 Integration of Heterogeneous Technical Spaces .....	29
4.1.1.3 MD Paradigm and DSLs in Document Engineering and Robot Motion Control ..	30
4.1.2 Industry 4.0.....	31

4.1.2.1 Error Handling in Production Processes .....	31
4.1.2.2 Manufacturing Documentation .....	32
4.1.2.3 Guided Production.....	33
4.1.2.4 Process Modeling .....	33
4.2 Production Process Modeling.....	34
4.2.1 Production Process Modeling Language Requirements .....	34
4.2.2 Production Process Modeling Languages and Approaches .....	36
4.2.2.1 Traditional Production Process Specification .....	45
4.2.2.2 Process Modeling Languages and Extensions .....	46
4.2.2.3 Combining Modeling Languages to Model Production Processes.....	58
4.2.2.4 Modeling Languages to Support Production Process or Production System Modeling .....	61
4.2.3 Discussion on Production Process Modeling.....	69
4.3 Summary .....	76
5 MD Solution for Modeling and Automatic Execution of Production Processes.....	77
5.1 Architecture of the MD System.....	77
5.2 Main Steps of the MD Approach.....	79
5.2.1 Main Steps of Modeling and Automatic Execution of Production Processes .....	80
5.2.2 Main Steps of Transforming Production Process Models into Manufacturing Documentation .....	84
5.3 Objectives of the MD Solution.....	85
5.4 Summary .....	87
6 Analysis of the Production Process Modeling Domain .....	89
6.1 Operational and Resource Perspectives .....	90
6.2 Control-Flow Perspective.....	93
6.3 Summary .....	95
7 Multi-Level Production Process Modeling Language .....	97
7.1 Overview and Usage of MultiProLan .....	97
7.2 Abstract Syntax of MultiProLan.....	100
7.2.1 Master-Level Modeling Concepts at Execution Layer.....	100
7.2.2 Detail-Level Modeling Concepts at Execution Layer.....	107
7.2.3 Modeling Concepts at Error Handling Layer.....	109
7.2.4 Modeling Concepts for the Automatic Generation of Manufacturing Documentation and Guided Production .....	112
7.3 Concrete Syntax of MultiProLan.....	114
7.4 Process Modeling Tool .....	122
7.5 Summary .....	126
8 Application of the MD Solution and MultiProLan.....	127
8.1 Example of MultiProLan Models of Customized Wooden Box Assembly .....	128
8.1.1 Master-Level Process Model of Wooden Box Production .....	129
8.1.2 Detail-Level Process Model of Assembling the Frame.....	137
8.1.3 Automatically Generated Instructions and Process Monitoring .....	140



8.1.4 Automatically Generated Manufacturing Documentation .....	144
8.2 Demonstration Environment with LEGO® Bricks .....	153
8.2.1 Test Scenarios Performed in the Demonstration Environment.....	153
8.2.2 Assembling a Flag from LEGO® Bricks .....	156
8.3 Summary .....	159
9 Evaluation of MultiProLan and Process Modeling Tool .....	161
9.1 Experiment Objective and Hypothesis.....	161
9.2 Experiment Participants .....	162
9.3 Experiment Preparation and Execution .....	163
9.4 Experiment Results and Data Analysis.....	164
9.4.1 Questionnaire Results .....	166
9.4.2 Statistical Analysis of the Questionnaire Answers .....	167
9.5 Overview of Other Quality Characteristics .....	168
9.6 Threats to Validity .....	169
9.7 Summary .....	170
10 Conclusions and Future Work.....	171
10.1 Outcome of Hypotheses Testing.....	171
10.2 Research Contributions.....	173
10.2.1 Theoretical Contributions.....	173
10.2.2 Development Contributions.....	174
10.2.3 Application Contributions .....	175
10.2.4 Socio-Economic Contributions.....	175
10.3 Future Work .....	175
10.3.1 Future Research in the Domain of Production Process Modeling .....	176
10.3.2 Further Development of MultiProLan and Process Modeling Tool .....	177
10.3.3 New Application Domains of MultiProLan .....	178
References .....	181
Appendix A. Evaluation Experiment Tasks and Questionnaire.....	197
Appendix A.1. Experiment Tasks .....	197
Appendix A.2. Experiment Solution .....	201
Appendix A.3. Experiment Questionnaire.....	204



## List of Figures

Figure 2.1.	A manufacturing flow of computer aided production systems .....	10
Figure 2.2.	The four-layer infrastructure of model levels .....	13
Figure 3.1.	Critical success factors of the novel MD solution.....	21
Figure 4.1.	The <i>Plank Sawing</i> model example in BPMN .....	48
Figure 4.2.	The <i>Plank Sawing</i> model example in UML AD .....	53
Figure 4.3.	The <i>Plank Sawing</i> model example in PN .....	55
Figure 5.1.	The architecture of the MD solution for production process modeling and execution.....	78
Figure 6.1.	A FODA model of a production process step suitable for an execution.....	91
Figure 6.2.	A FODA model of a production process suitable for an execution.....	94
Figure 7.1.	Context and architecture of MultiProLan .....	98
Figure 7.2.	The first part of the meta-model used for MasL model creation at Execution Layer.....	101
Figure 7.3.	The second part of the meta-model used for DetL model creation at Execution Layer.....	107
Figure 7.4.	The third part of the meta-model used for the error handling modeling at Error Handling Layer .....	109
Figure 7.5.	The fourth part of the meta-model used for the automatic generation of manufacturing documentation and guided production .....	113
Figure 7.6.	The Process Modeling Tool user interface .....	122
Figure 7.7.	Process Modeling Tool Toolbar.....	123
Figure 7.8.	The Process Modeling Tool layer button.....	123
Figure 7.9.	The generate documentation dialog.....	124
Figure 7.10.	Tool palette of Process Modeling Tool without additional layers turned on .	125
Figure 7.11.	Tool palette of Process Modeling Tool with both additional layers turned on .....	125
Figure 8.1.	Parts of the wooden box used for the assembly .....	128
Figure 8.2.	A variation of the assembled wooden box.....	129
Figure 8.3.	The main MasL process model of wooden box production.....	130
Figure 8.4.	The <i>Assemble frame</i> MasL sub-process model.....	131
Figure 8.5.	The <i>Hammer back side</i> MasL sub-process model.....	132
Figure 8.6.	The <i>Glue fabric and dividers and place lid</i> MasL sub-process model.....	134

Figure 8.7.	The wooden box product variations model.....	135
Figure 8.8.	The wooden box variations .....	135
Figure 8.9.	The <i>Glue fabric and dividers and place lid</i> MasL sub-process model with a single variation ( <i>1.3 Box_TD_WL</i> ).....	136
Figure 8.10.	The <i>Assemble frame</i> DetL sub-process model .....	138
Figure 8.11.	A user interface mockup of the human worker application .....	143
Figure 8.12.	An example of monitoring the execution of the <i>Assemble frame</i> DetL sub-process model.....	145
Figure 8.13.	The BOM template used by Instruction Generator.....	147
Figure 8.14.	An automatically generated BOM document from the <i>Glue fabric and dividers and place lid</i> MasL sub-process model.....	147
Figure 8.15.	An automatically generated BOMO document from the <i>Glue fabric and dividers and place lid</i> MasL sub-process model .....	148
Figure 8.16.	An automatically generated ASME FPC document from the <i>Glue fabric and dividers and place lid</i> MasL sub-process model .....	148
Figure 8.17.	The <i>Glue fabric</i> process step key points .....	149
Figure 8.18.	An automatically generated JBS document from the <i>Glue fabric and dividers and place lid</i> MasL sub-process model.....	150
Figure 8.19.	The <i>Insert pins in bottom side</i> process step errors and error handler process steps .....	151
Figure 8.20.	An automatically generated PFMEA document from the <i>Assemble frame</i> DetL sub-process model .....	152
Figure 8.21.	The scheme of assembly demonstration environment .....	153
Figure 8.22.	The digital twin of assembly demonstration environment .....	154
Figure 8.23.	The assembly demonstration environment with research-grade smart robots .....	155
Figure 8.24.	The assembly demonstration environment with an industrial mobile robot ..	155
Figure 8.25.	The MasL process model of assembling the red-blue-white flag out of LEGO® bricks .....	156
Figure 8.26.	The DetL process model of assembling the red-blue-white flag out of LEGO® bricks .....	158
Figure 8.27.	The produced red-blue-white flag out of LEGO® bricks .....	159
Figure A.1.	The wooden box whose production process is to be modeled .....	197
Figure A.2.	The solution of the first experiment task .....	201
Figure A.3.	The solution of the second experiment task .....	202
Figure A.4.	The solution of the third experiment task .....	203

## List of Graphs

Graph 4.1.	Number of peer-reviewed papers per year .....	44
Graph 4.2.	Number of peer-reviewed papers per year for the second, third, and fourth categories .....	44



## List of Tables

Table 4.1.	Search tokens and keywords .....	36
Table 4.2.	Peer-reviewed papers of the reviewed literature .....	39
Table 4.3.	The distribution of the presented languages and related papers by categories.	42
Table 4.4.	A comparison of process modeling languages .....	73
Table 7.1.	The basic modeling concepts of MultiProLan .....	114
Table 7.2.	MultiProLan requirements fulfilment.....	117
Table 9.1.	The questionnaire statistics .....	165
Table 9.2.	Correlation coefficients and p-values for related questions .....	168





## List of Listings

Listing 7.1.	Constraints related to flow-type and collaboration-type relationships .....	102
Listing 7.2.	The specification of the relational operator domain in product and capability constraints .....	103
Listing 7.3.	Constraints related to the start and end process steps.....	103
Listing 7.4.	Constraints related to the <i>startStep</i> relation and the collaboration-type relationship.....	104
Listing 7.5.	Constraints related to the pair of gates .....	105
Listing 7.6.	Constraints related to diverging and converging gates.....	105
Listing 7.7.	A constraint related to selection and iteration patterns.....	106
Listing 7.8.	A constraint related to a process element variation.....	106
Listing 7.9.	Constraints related to an unordered set of steps.....	106
Listing 7.10.	Constraints related to resources and storage .....	108
Listing 7.11.	A constraint related to regular storage.....	108
Listing 7.12.	Constraints related to the start and end process step errors .....	109
Listing 7.13.	Constraints related to local and global error handlers .....	110
Listing 7.14.	A constraint related to a default error handler.....	110
Listing 7.15.	A constraint related to an unordered set of error steps .....	111
Listing 7.16.	Constraints related to error-type and flow-type relationships.....	111
Listing 7.17.	Constraints related to gates from the error group.....	111
Listing 7.18.	Constraints related to the start and end process steps and key points.....	112
Listing 8.1.	An example of the <i>Pick right side</i> high-level instruction .....	140
Listing 8.2.	An example of the <i>Pick right side</i> machine-specific ROS instruction .....	141
Listing 8.3.	An example of the <i>Assemble left-bottom sides</i> high-level instruction.....	142



## List of Acronyms

AAS	Asset Administration Shell.....	68
ADAPT	Asset-Decision-Action-Property-relaTionship .....	60
AGV	Automated Guided Vehicle .....	82
API	Application Programming Interface.....	124
APICS	American Production and Inventory Control Society .....	45
APQP	Advanced Product Quality Planning.....	32
AR	Augmented Reality .....	33
ASME	American Society of Mechanical Engineers .....	45
ASML	Assembly Sequence Modeling Language .....	61
BOM	Bill of Materials.....	32
BOMO	Bill of Materials and Operations.....	45
BOO	Bill of Operations .....	45
BPEL	Business Process Execution Language .....	49
BPM	Business Process Management.....	50
BPMN	Business Process Model and Notation .....	43
BSM	Business System Model .....	14
CAD	Computer Aided Design.....	10
CAM	Computer Aided Manufacturing.....	10
CAPP	Computer Aided Process Planning .....	10
CASE	Computer Aided Software Engineering .....	28
CE- MultiProLan	Collaborative Extension of Multi-Level Production Process Modeling Language .....	25
CIM	Computation-Independent Model .....	14
CPPS	Cyber-Physical Production System.....	8
CPS	Cyber-Physical System .....	8
CT	Composition Tree .....	58
DetL	Detail-Level.....	81
DIN	Deutsches Institut für Normung (English: German Institute for Standardization).....	12
DLT	Distributed Ledger Technology .....	177
DPT	Digital Process Twin.....	67

DSL	Domain-Specific Language .....	13
DSML	Domain-Specific Modeling Language .....	3
DSR	Design Science Research.....	23
DSRM	Design Science Research Methodology .....	23
EMF	Eclipse Modeling Framework.....	13
EPC	Event-driven Process Chain.....	51
ER	Entity-Relationship .....	28
FMEA	Failure Mode and Effect Analysis.....	31
FODA	Feature-Oriented Domain Analysis.....	24
FOSD	Feature-Oriented Software Development .....	89
FPC	Flow Process Chart .....	32
FQAD	Framework for Qualitative Assessment of Domain-specific languages.....	24
GenERTiCA	Generation of Embedded Real-Time Code based on Aspects .....	31
GMA	VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik (English: VDI/VDE Society for Measurement and Automatic Control).....	65
GME	Generic Modeling Environment.....	31
GMPM	Green information-based Manufacturing Process Modeling.....	64
GPML	General-Purpose Modeling Language.....	13
GRAMOSA	GRaphical Modeling and Simulation-based Analysis.....	62
GSMSP	Graph-based Simulation of Multi-Stage Production Processes .....	59
HMI	Human-Machine Interaction .....	7
HMS	Holonic Manufacturing System .....	68
HResModLan	Human Resource Modeling Language .....	25
HSE	Health, Safety, and Environment .....	70
HTTP	HyperText Transfer Protocol.....	51
I4PML	Industry 4.0 Process Modeling Language .....	60
I4PMM	Industry 4.0 Process Modeling Method.....	60
IAPMM	IoT-Aware Process Modeling Method .....	60
IAPMN	IoT-Aware Process Modeling Notation .....	60
IBPM	Industrial Business Process Management.....	59
IDEF	Integration DEFinition .....	57
IDEF3	Integration DEFinition method for Process Description Capture .....	57
IIoT	Industrial Internet of Things .....	9
IIS*Case	Integrated Information Systems*Case.....	28
IIS*Studio	Integrated Information Systems*Studio .....	29
IoT	Internet of Things.....	7
IP	Internet Protocol.....	67
IPPMA	Integrated Product-Process Modeling Approach .....	63
IS	Information System.....	9
IT	Information Technology.....	7
JBS	Job Breakdown Sheet .....	112
JSON	JavaScript Object Notation .....	69

KS A	Korean Standards Association.....	45
LCDP	Low-Code Development Platform.....	69
M2M	Model-to-Model.....	15
M2T	Model-to-Text.....	15
MaRCO	Manufacturing Resource Capability Ontology .....	62
MasL	Master-Level .....	80
MBSE	Model-Based System Engineering.....	67
MD	Model-Driven .....	2
MDA	Model-Driven Architecture .....	12
MDD	Model-Driven Development.....	12
MDE	Model-Driven Engineering.....	12
MDSD	Model-Driven Software Development.....	12
MDSE	Model-Driven Software Engineering.....	12
MDSEA	Model-Driven System Engineering Architecture .....	14
MES	Manufacturing Execution System.....	9
MES-ML	Manufacturing Execution System Modeling Language.....	63
MMPD	Meta-Model for Production Data.....	62
MoDEBiTE	Model-Driven Engineering of Bidirectional Transformations via Epsilon .....	89
MOF	Meta Object Facility .....	13
MPC	Manufacturing Planning and Control.....	9
MPIM	Manufacturing Process Information Model.....	62
MPIMM	Manufacturing Process Information MetaModel.....	62
MR	Mixed Reality .....	179
MService	Manufacturing Service.....	68
MultiProLan	Multi-Level Production Process Modeling Language .....	3
NIST	National Institute of Standards and Technology.....	63
OCL	Object Constraint Language.....	97
OMG	Object Management Group .....	13
OPC UA	Open Platform Communications Unified Architecture.....	54
OWL	Web Ontology Language .....	62
P&ID	Piping and Instrumentation Diagram .....	166
PBM	Process-Based Modeling.....	59
PFMEA	Process Failure Mode and Effect Analysis.....	46
PIM	Platform-Independent Model.....	14
PIPE2	Platform Independent Petri net Editor 2.....	54
PLC	Programmable Logic Controller .....	60
PM-HRC	Process Model-based Human-Robot Collaboration.....	65
PMPM	Part-flow based Manufacturing Process Modeling.....	65
PN	Petri Net .....	46
PPR	Product-Process-Resource.....	38
PPR-SS	Product, Process, Resource, Schedule, and Space .....	59

PQ	Predictive Quality .....	63
PSL	Process Specification Language.....	63
PSM	Platform-Specific Model .....	14
REST	REpresentational State Transfer.....	29
ROS	Robot Operating System.....	79
RPN	Risk Priority Number .....	151
RSD	Rapid Software Development .....	122
SAP	Systemanalyse Programmentwicklung (English: System Analysis Program Development) .....	166
S-BPM	Subject-oriented Business Process Management .....	57
SOA	Service-Oriented Architecture .....	69
SQL	Structured Query Language.....	67
SysML	Systems Modeling Language.....	54
SysML AD	Systems Modeling Language Activity Diagram .....	54
T2M	Text-to-Model.....	15
T2T	Text-to-Text.....	83
TIM	Technology Independent Model .....	14
TNO	Nederlandse Organisatie voor Toegepast-Natuurwetenschappelijk Onderzoek (English: Netherlands Organization for Applied Scientific Research).....	49
TSM	Technology Specific Model.....	14
UML	Unified Modeling Language.....	28
UML AD	Unified Modeling Language Activity Diagram .....	46
UOB	Unit of Behavior .....	57
VDE	Verband der Elektrotechnik Elektronik Informationstechnik (English: Association for Electrical, Electronic & Information Technologies).....	65
VDI	Verein Deutscher Ingenieure (English: Association of German Engineers) ...	12
VSM	Value-Stream Mapping .....	49
WS-BPEL	Web Services Business Process Execution Language.....	51
WWW	World Wide Web .....	62
XMI	XML Metadata Interchange.....	67
XML	Extensible Markup Language .....	60
YAFMT	Yet Another Feature Modeling Tool.....	89

# 1 Introduction

Manufacturing is a term used for the essential activity of making products and services. The word *manufacture* is several centuries old, derived from two Latin words: *manus* – a hand, and *factus* – to make, meaning made by hand. Besides making objects by hands, humans have been using tools and materials since the Stone Age. Various tools and materials are still used nowadays, and, over time, manufacturing became more and more sophisticated. Contemporary manufacturing is mainly done by automated and computer-controlled machinery [1,2].

Manufacturing can be observed from both a technological and an economic viewpoint. From the technological viewpoint, manufacturing represents an application of physical or chemical processes to change the properties of materials, creating parts or products. It is carried out by a set of operations, each transforming materials a step closer to the final product form. From the economic viewpoint, manufacturing is the transformation of materials into parts or products of greater value. Each manufacturing operation changes the properties of materials, adding value to them.

As it is crucial for a country's welfare and a standard of living to sell final products created by manufacturers in the country, the goal of manufacturing is to satisfy customer needs by making sellable products or by providing services. As customer needs change over time, a product or a service needs to be continually improved. Consequently, products and their manufacturing processes are also continually evolving. Due to product changes, different product variations emerge and are often grouped inside product families. Each product variation within a product family differs from others by one or more specific parts or materials. Also, each product variation has a corresponding production process variation. Every time a new process variation emerges, the existing shop floor in a manufacturing company needs to be changed and rearranged. This can be, and often is, challenging as the shop floor and its processes are usually rigid.

Besides manually rearranging shop floors frequently, they may be self-adjustable to meet the demand for customized products. Until the end of the 20<sup>th</sup> century, machines were still not fully independent and self-adjustable to variations in the production processes. Changing a shop floor was a burdensome and time-consuming task that required stopping production. Each time production was stopped, a company lost money. Therefore, the way of manufacturing needs to be changed in order to respond to customer demands and numerous product and process variations.

In the rest of this section, we provide an overview of the research conducted and presented in this Ph.D. thesis. This section is structured as follows. In Section 1.1, we discuss research challenges, motivation, the main hypothesis, and the main goal. A brief overview of the proposed solution is outlined in Section 1.2, while the expected research results and contributions are presented in Section 1.3. The section concludes with the thesis structure presented in Section 1.4.

## 2 Introduction

### 1.1 Research Challenges, Motivation, Hypothesis and Goal

A new way of manufacturing, providing flexible shop floors, is supposed to be implemented during the fourth industrial revolution, named *Industry 4.0*. The aim of Industry 4.0 is to enable the production of numerous individualized products for customers, creating an environment for *lot-size-one* production while preserving the economic characteristics of mass production. However, with such an environment, the number of product and process variations increases even more, as customers raise their needs. Therefore, one of the main challenges of Industry 4.0 is:

*(Ch1) to cope with numerous product and process variations by creating a flexible environment for lot-size-one production.*

In such a dynamic and flexible production, various errors and failures may occur, that can lead to production being stopped. In contemporary production systems, when an error occurs during production, a production line needs to be stopped, causing additional costs to factories. In Industry 4.0 flexible production, the exact procedures on how to handle errors must exist and, in most cases, these error handling procedures must be performed without a need to stop production. Thus, an important Industry 4.0 challenge is:

*(Ch2) to run flexible production smoothly, even if an error occurs, by having well-established procedures that are automatically performed to minimize damage and costs caused by errors.*

Additionally, due to many variations, manufacturing documentation required by different standards and procedures increases as well. Each time a product or its process changes, the documentation of different types must be updated and new versions created, which is a time-consuming task. To keep the documentation up to date, additional personnel need to be engaged, raising costs for a manufacturing company. Accordingly, as creating and modifying product and process variations and keeping the manufacturing documentation up to date becomes a burdensome manual task, another important Industry 4.0 challenge that needs to be addressed is:

*(Ch3) to manage product and process variations easily and keep the manufacturing documentation up to date automatically.*

As various products and their variations are produced in a factory, human workers must adapt and create new products often. Thus, a fast knowledge transfer is needed on how to produce all these products and variations. Otherwise, a lot of time needs to be spent in training workers when creating new products and expert workers need to spend a lot of time training new workers coming to a factory. In addition, as the number of human workers decreases in factories, they often need to change their workplace and create various products. Therefore, another important Industry 4.0 challenge is:

*(Ch4) to provide automated knowledge transfer and guided production to lower the time and costs required for human worker training.*

Recently, both the research community and the industry are trying to create such a flexible environment that manages many product and process variations, handles production errors dynamically, keeps the manufacturing documentation up to date, and provides a fast knowledge transfer to human workers. However, none have managed to cope with the challenges (Ch1–Ch4) fully. The creation of dynamic and flexible production is still in its early stages of development, and researchers apply various existing approaches and create new ones to enable such production.

One way to deal with the challenges (Ch1–Ch4) is by using an intelligent system that is built on the Model-Driven (MD) principles, having models as central artifacts that lead the production process execution. Thus, production processes, alongside different product and process variations, could be specified in the form of formal and machine-readable models with precise semantics. These formal, machine-readable models could be automatically transformed into executable resource instructions that the intelligent system dynamically orchestrates and allocates to the shop floor resources. Whenever these instructions are sent to human workers, the instructions could



contain descriptions, images and videos, guiding workers to create a product and transfer new knowledge to them. If process models are enriched with corrective process steps as a response when an error occurs, the models could be used by the intelligent system to manage error handling. Also, production process models could be used to automatically generate manufacturing documentation and keep the documentation up to date by updating it whenever a change appears in a product or its process.

However, creating such *resource-aware* production process models with all the details related to process execution, error handling and manufacturing documentation, makes the modeling challenging, as it requires factory shop floor details to be included in process models. Creating a resource-aware production process model with all these details requires process designers to have knowledge about the shop floor on which the production process model is to be executed. Thus, process modeling becomes a burdensome and time-consuming task. Additionally, a resource-aware production process model is dependent on a specific production system. In Industry 4.0, production process models should be independent of any production system, allowing *resource-agnostic* production process models to be reused in different production systems. Accordingly, one of the biggest challenges in Industry 4.0 related to production process modeling is:

*(Ch5) to allow the creation of resource-agnostic production process models in a simple manner, easy to understand, and reusable in multiple production systems, but also automatically transformable into executable resource instructions.*

To transform a resource-agnostic production process model into executable resource instructions, the model needs to be manually or automatically transformed into a resource-aware production process model first. The automatic transformation of resource-agnostic into resource-aware production process models is also one of the important Industry 4.0 challenges but is out of scope in this thesis.

Many researchers put a lot of effort into solving the production process modeling challenge (Ch5). Production processes are modeled in different ways, and even if they are suitable for the automatic generation of executable resource instructions, these models are closely coupled with a specific production system to which instructions are to be sent, leaving the modeling challenge (Ch5) still open.

In this thesis, we aim to contribute to solving the research problem discussed through the research challenges (Ch1–Ch5), by creating a novel MD solution for production process modeling and execution. Thus, we formulate the main hypothesis and the main goal of our research:

**Hypothesis 0 ( $H_0$ ).** *It is possible to create an MD solution for resource-agnostic and resource-aware specification of production processes and automatic transformation of such specifications into executable resource instructions and manufacturing documentation.*

*The main goal of our research, derived from the main hypothesis, is to introduce factories into the digital transformation process, by providing dynamic production management and automatic production process execution based on process models.*

## 1.2 Overview of the Proposed Solution

To achieve the research goal, we propose a novel methodological approach and a software solution that utilizes MD principles, and a novel Domain-Specific Modeling Language (DSML), named *Multi-Level Production Process Modeling Language (MultiProLan)*, to specify production processes formally in the Industry 4.0 era. The aim of such an MD solution and MultiProLan is also to help process designers model production processes and keep the manufacturing documentation up to date in a more efficient manner.

## 4 Introduction

To cope with the Industry 4.0 challenges (Ch1–Ch5), our solution uses production process models that automatically lead the execution of production processes by generating and executing resource instructions from the models. Accordingly, such models need to be machine-readable and numerous production and execution details need to be stored in the models. However, including production and execution details in production process models makes manual process modeling difficult for a process designer. Therefore, these models are presented through different *levels of detail* to separate process models that are independent of any production system and modeled by process designers, and process models that are made for specific production systems in which they are to be executed. In addition, to cope with numerous details in production process models, *modeling layers* need to be created to show or hide different aspects of production processes.

As MultiProLan supports different modeling levels and layers, making them one of the main features of our language, we named it Multi-Level Production Process Modeling Language to highlight such an important feature. MultiProLan separates resource-agnostic from resource-aware production process models, allowing process designers to be focused only on production process steps instead of execution details and production systems. An intelligent system, named orchestrator, is used to automatically transform resource-agnostic into resource-aware production process models. This can be achieved as MultiProLan is a *capability-based* process modeling language, meaning that each process step in a model has the capability required to execute the process step. In the orchestrator, there is information about each resource and the capabilities it offers, allowing process steps to be allocated to resources based on capabilities.

As MultiProLan models are machine-readable, executable resource instructions can be automatically generated from resource-aware models and sent to resources to execute instructions and create products. Production process step images and video footage can be automatically generated and sent to human workers one by one with textual and audio descriptions on how to perform each step, enabling guided production and training for them. In addition, manufacturing documentation can be automatically generated and updated whenever new products or variations emerge, or existing products or variations change, eliminating such a burdensome manual task from process designers.

### 1.3 Research Contributions and Results

The overall research presented in this thesis is expected to have the following contributions.

**Theoretical contributions** are expected to cover a novel MD approach and a modeling language in the field of production process modeling, and also a novel methodology for the automatic transformation of production process models into executable resource instructions and manufacturing documentation.

**Development contributions** are expected to cover the development and implementation of a novel software solution for production process modeling and novel instruction and manufacturing documentation generators.

**Application contributions** are expected to cover the application of an MD approach and a modeling language in the production domain in the form of use cases with practical application in the assembly industry and the evaluation of the modeling language and the tool for production process modeling.

In addition to the aforementioned scientific and practical contributions, it is important to mention the socio-economic contributions expected from the conducted research in the light of contemporary trends in the world in the development of the Industry 4.0 field. These **socio-economic contributions** refer to the possibility of putting into public use a general model of production process management, applicable in a wide range of organizations in a way that enables significant production process improvement and raises general accumulated knowledge on how to contribute to such a process improvement contemporarily.

The main **expected result** of our research is easier and simpler formal modeling of production processes in the era of Industry 4.0. Created models are to lead the process execution in flexible production and keep the manufacturing documentation up to date automatically, thus responding to increasing customer demands for individualized products and coping with Industry 4.0 challenges (Ch1–Ch5).

More details about the research motivation, goals, expected contributions and results, and research methodology are presented in Section 3. Also, besides the main hypothesis presented in this section, there are four hypotheses derived from the main one, presented in Section 3. All these details are not presented in the Introduction section to make it clear, concise, and simple. Thus, before going into details in Section 3, first we introduce the background and theoretical foundations related to our research in Section 2. The complete overview of the thesis structure is outlined in the following section.

## 1.4 Thesis Structure

Apart from Introduction, Conclusion, and Appendix, this thesis is structured as follows.

In Section 2, we present the background and theoretical foundation related to this thesis. First, we discuss industrial revolutions, especially Industry 4.0 and its relevant concepts and technologies. Afterward, we present an overview of production process planning and design, and skill-based engineering. The section ends with an overview of the MD paradigm and DSMLs.

Research challenges, motivation, and goals are presented in detail in Section 3. The main and four derived hypotheses are presented and discussed in this section, as well as the expected contributions and results of this research. The research methodology and the activities performed to conduct the research are also presented in this section.

In Section 4, we present the state-of-the-art of different aspects related to the MD paradigm and Industry 4.0, with production process modeling and execution as the main aspects. Based on the preliminary research, industrial use cases we encountered, and the discussion with domain experts, we specified requirements for a production process modeling language whose models would be suitable for dynamic orchestration and automatic model execution or generation of executable instructions. Afterward, the researched languages and approaches are presented in detail, and divided into four categories we identified. These languages and approaches are tested on fulfilling the specified requirements. In the Summary subsection, we discuss the requirements coverage by languages and approaches, the features that are mostly not covered by languages and approaches, and the future research directions in production process modeling.

Our MD solution, comprised of the novel MD approach and the MD system, is discussed in detail in Section 5. First, the architecture of our MD system is presented, containing modeling tools, an orchestrator and a knowledge base, different code generators, and a production system with a digital twin. Afterward, the main steps of our MD approach are outlined, comprising the specification of production system models and production process models, and the automatic transformation of process models into executable resource instructions and manufacturing documentation. The section ends with the objectives that our MD solution aims to fulfill.

To develop a novel DSML for production process modeling, used as the main component of our MD system, the relevant domain needs to be analyzed first. We divided the production process modeling domain into operational, resource, and control-flow perspectives, and presented domain concepts relevant to a modeling language that aims to model production processes in the context of Industry 4.0. The production process modeling domain analysis is outlined in Section 6.

In Section 7, we present MultiProLan, the novel DSML for production process modeling. In this section, we present an overview and the usage of MultiProLan, its detail levels and modeling layers, as well as the users of the language. Then, we present the abstract and concrete graphical syntaxes of MultiProLan and a way in which MultiProLan fulfills the requirements specified in

## 6 Introduction

Section 4. An overview of the process modeling tool that utilizes MultiProLan is presented at the end of Section 7.

In Section 8, the application of our MD solution and MultiProLan is outlined in order to present their usability in different use cases. There are two proof-of-concept use cases presented in this section. In the first use case, a customized wooden box is assembled, having different product variations. This use case is created to demonstrate the possibilities of MultiProLan and its process modeling tool. The second use case represents a demonstration environment created to test the whole solution in assembling objects from LEGO<sup>®</sup> bricks, avoiding failure costs associated with the real production system. In this demonstration environment, different robots are used and some of them are industrial mobile robots, which are used in real production. In this section, various production process models are presented, as well as automatically generated instructions and manufacturing documentation.

The evaluation of MultiProLan and its process modeling tool by different user groups is presented in Section 9. The evaluation participants performed an experiment in which they modeled production processes and evaluated the following quality characteristics of MultiProLan and the modeling tool: functional suitability, usability, reliability, expressiveness, and productivity. In this section, we discuss the experiment objective and hypothesis, user groups that participated in the experiment, the preparation and execution of the experiment, the results of the evaluation, and possible threats to validity.

## 2 Background and Theoretical Foundation

The way of manufacturing has changed over time, especially during industrial revolutions. An industrial revolution involves fundamental changes, altering different aspects of societies, such as production and economics. Each industrial revolution significantly impacted countries over the past three centuries. To this date, three revolutions have happened, triggered by the following breakthroughs: (i) the James Watt's steam engine in 1784; (ii) the power of electricity and oil with internal combustion motors in the 1870s; and (iii) electronics and microcontrollers in the 1960s [102,103]. Although these changes are called industrial revolutions, historians raise concerns over using the term revolution, as these changes took several decades to modify production [103]. Currently, a new revolution is happening. The fourth industrial revolution has been triggered by technological advances, such as highly networked devices and new Information Technology (IT) infrastructure, as well as increasing customer needs for highly customized products. As it began only a few years ago, we expect it to last for years to come, similar to previous revolutions.

The fourth industrial revolution is also known as *Industry 4.0* (German: Industrie 4.0), a term coined in 2011 by Kagermann et al. [104] when they proposed ideas on how to strengthen the competitiveness of the German manufacturing industry. Although this particular name was coined in Germany, other countries also have similar national programs and research projects, albeit named differently, in the domain of smart manufacturing in order to define and implement its concepts [105].

Industry 4.0 emerged at the beginning of the 21<sup>st</sup> century, and companies are moving from *mass production* to *mass customization*. Advanced technologies in the form of smart resources and smart products represent the basis for Industry 4.0, enabling fast changes of products created in factories. Industry 4.0 introduces primarily IT-driven changes in existing production systems to enable the production of individualized products while preserving all beneficial economic characteristics of mass production [106,107].

Ivanov et al. [108] defined the notion of Industry 4.0 as the industry that integrates technologies, organizational concepts, and management principles to adapt to rapid production changes by rearranging and reallocating components and capabilities. The goals of Industry 4.0 are to [105,109]:

- provide IT-enabled mass customization of products;
- make an automatic and flexible adaptation of production;
- track parts and products during production;
- facilitate communication among parts, products, and machines;
- apply Human-Machine Interaction (HMI) and ensure the safety of human workers and machines working together in close proximity;
- achieve Internet of Things (IoT)-enabled production optimization in factories;
- provide new types of services and business models;
- save energy and reduce lead times, costs, and waste; and

## 8 Background and Theoretical Foundation

- improve the quality of products and efficiency of production.

To sum up the goals, Industry 4.0 aims to achieve a higher level of operational efficiency, productivity, and automatization, providing more flexible and *lot-size-one* production.

However, contemporary production environments and machines are not appropriately automated, fully independent, and self-adjustable to support such flexible production. Therefore, the Industry 4.0 goals are yet to be met despite the numerous contributions to smart manufacturing in recent years.

The MD paradigm can be used to create an environment to make production more flexible and in which production process models lead the production process execution. Executable instructions can be automatically generated from process models and sent to Industry 4.0 smart resources for execution. Therefore, in this section, we outline the background and theoretical foundation related to the production process modeling in the context of Industry 4.0 and the MD paradigm. The main concepts and technologies used in the Industry 4.0 context are presented in Section 2.1. The way production processes are planned and designed is described in Section 2.2. A brief overview of the MD paradigm and DSMLs is outlined in Section 2.3. The summary of the background and theoretical foundation is presented in Section 2.4.

### 2.1 Relevant Concepts and Technologies in Industry 4.0

Numerous technologies are used to bring the Industry 4.0 concepts to life in companies [109,110]. Therefore, to discuss further about the Industry 4.0 context related to our research, in this section we describe the main concepts and technologies commonly used in Industry 4.0:

- **Smart Factory** is an intelligent production system based on digital and automated production, using IT to improve the management and quality of production [111]. A smart factory realizes flexible manufacturing, dynamic reconfiguration, and production optimization, adapting to rapid changes in production due to customer needs. It consists of intelligent and autonomous shop floor entities aiming to decentralize production [112]. These entities represent smart materials or smart products and smart resources, able to store information of different types, such as their state, position, and history. *Smart materials* and *smart products* are service consumers, processed by *smart resources* that are service providers. As these smart entities should be autonomous and independent, a smart factory should be able to produce customized and small-lot products efficiently and profitably [113]. Such customized production and a smart environment are enabled by advanced technologies, such as Cyber-Physical System (CPS) and IoT [114], introduced further in this section.
- **Cyber-Physical System (CPS)** is an automated system that integrates both physical and virtual worlds, networking devices of different types [115,116]. It consists of a control unit, controlling sensors, and actuators necessary to interact with the real world, with the purpose of synchronizing physical and virtual worlds [115,117]. There are various fields of application for CPS, such as medical equipment, driving safety and assistance, autonomous cars, industrial process control and automation systems, and the smart electric grid. CPS applied in manufacturing is usually called Cyber-Physical Production System (CPPS). It relies on computer science, information and communication technologies, and manufacturing science and technologies, enabling the equipment of a smart factory to become intelligent and thus leading to smart production [21,118].
- **Digital Twin** is a digital part of a CPS, representing a virtual model of a physical object. It is a digital representation of an active product or a product-service system, and it comprises their characteristics, conditions, and behavior [119]. A digital twin can be used to simulate the object's behavior, while the object can respond to changes made in the simulation [120]. It should be possible to simulate all production steps and depict their influence on production, enabling product-specific costs to be calculated in advance [121]. Simulations

need to be highly utilized in Industry 4.0 to simulate products, robots, and humans in order to reduce failures and optimize resource consumption [122]. Accordingly, by testing production using digital twins and simulations, production failures may be mitigated or missing production steps detected, lowering costs and wastes due to production failures.

- **Internet of Things (IoT)** may be seen as a CPS connected to the Internet [115]. It is considered Internet of the future, consisting of numerous heterogeneously and wirelessly connected devices that interact without human intervention [123]. IoT applied in manufacturing is usually referred to as Industrial Internet of Things (IIoT). It covers the domain of machine-to-machine and industrial communication technologies, enabling efficient and sustainable production [124]. As the pillar of digital manufacturing, IIoT aims to connect all industrial assets with Information Systems (ISs) and business processes.
- **Production orchestration** represents diverse activities of scheduling and allocating operations to resources. These activities should be performed automatically and dynamically in Industry 4.0, enabling fast adaptation of the production floor to customer needs. A component named orchestrator can be used in production systems to achieve such goals. An *orchestrator* is software run on top of a cluster of industrial computers that can orchestrate smart resources and assign them process steps for execution [3–5].
- **Manufacturing Execution System (MES)** provides a common user interface and a data management system, integrating multiple execution management components into a single solution [125]. MES assists production by managing activities on the shop floor and serves as a bridge between planning and control systems [126]. Smart factories utilize MES to assist human workers and machines in executing their tasks during production.

The presented concepts and technologies are referenced throughout this thesis, but mostly as a context in which we conduct or apply the research related to the production process modeling. The following section introduces an overview of production process planning and design as they are currently performed by manufacturing companies to the best of our knowledge.

## 2.2 Process Planning and Design

In this section, we describe activities of designing products and processes, planning processes, and manufacturing products. This section is divided into the following two subsections. In Section 2.2.1, process planning activities are discussed, while process design and its integration with skill-based engineering are described in Section 2.2.2.

### 2.2.1 Production Process Planning

Manufacturing Planning and Control (MPC) comprises a wide range of activities, from managing materials and scheduling resources to coordinating suppliers and planning customer shipments [127]. Part of MPC activities is related to designing products and planning production processes, which are relevant to the research presented in this thesis. Other MPC activities are not considered in this thesis.

*Process planning* represents a set of strategies, methods, and activities needed to systematically determine manufacturing operations required to create a product economically and competitively [128–132]. Each manufacturing operation needs to be defined in the best possible way so that created products fulfill the intended quality at a competitive price. The input to process planning is a product design, such as a product's blueprint, while the output of process planning are various manufacturing documents, including a production process specification and a plan for how to produce a product, and the manufacturing of a product.

There are different types of planning, such as: (i) strategic or long-term; (ii) tactical or medium-term; and (iii) production scheduling or short-term planning [133]. Long-term planning determines a manufacturing company's supply chain structure, which represents a network of production

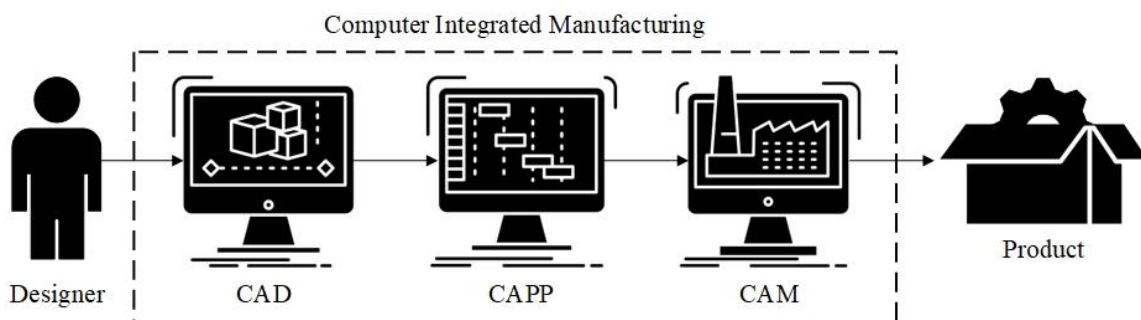
facilities and distribution options. Medium-term planning includes decisions such as assigning production targets to production facilities and transportation logistics from facilities to warehouses. Short-term planning is performed daily or weekly, and at the production level, it is referred to as scheduling. *Production scheduling* represents an allocation of tasks to available production resources over time, to best satisfy a set of various optimization criteria [134]. In the research presented in this thesis, we are focused on short-term planning, especially in process design and execution. Thus, we do not consider the management of production facilities or material handling at the level of warehouses and facilities in our research.

Since computers are known for their speed and consistency, Niebel [135] first presented the idea of using them to assist in the determination of process plans in 1965, and the year after, Schenk [136] discussed the feasibility of automated process planning. Computer Aided Process Planning (CAPP) has been researched since then, and its purpose is to automate process planning tasks in order to generate process plans consistently [129]. CAPP is an essential component of a computer integrated manufacturing environment, and it acts as a bridge between Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) [129,137]. CAD uses computer systems to assist in the creation, modification, analysis, and optimization of a design, increasing the productivity of designers [138]. Products or parts can be designed using CAD software, and the output of such software is two-dimensional drawings or three-dimensional models. CAM uses computer systems to plan, manage, and control operations of production resources in a manufacturing plant, automating a manufacturing process. Computer integrated manufacturing utilizes computer systems to integrate concepts such as CAD, CAPP, and CAM into one extensive system, controlling the entire production. A manufacturing flow from computer aided product design to manufacturing is presented in Figure 2.1.

A novel methodology that is considered nowadays and is derived from computer aided planning is the MD process planning. The MD process planning is a methodology in which digital models are applied to create, represent, and use the information of products, processes, and resources, supporting process planners in creating process plans [131,132]. The resulting process plan is digital, and, unlike traditional CAPP plans, it is a computer interpretable model with exact and precise semantics, defining what and how it will be produced. Furthermore, the MD process planning enables a comparison between design objectives – *as planned*, and production and inspection information – *as realized*, by gathering feedback from the process execution. Therefore, structured knowledge is developed, making it possible to improve product and process design.

## 2.2.2 Production Process Design and Skill-Based Engineering

A *manufacturing process design* is a part of the process planning phase consisting of process steps needed to create a product. A *manufacturing process* is a designed procedure that physically, mechanically, thermally, electrically, or chemically changes an input material intending to increase its value [1,139]. Besides materials, the input of a manufacturing process includes machines, tooling, energy, labor, and knowledge, and after a set of operations is executed, a processed part or



**Figure 2.1.** A manufacturing flow of computer aided production systems.



product with scrap, waste, and emissions are the output of such a process [1,140]. Therefore, a manufacturing process is usually defined as a transformation activity consisting of a collection of operations in which human workers and machines use energy and information to transform raw materials into finished products, fulfilling certain requirements [2,32,139]. There is also a term *production process*, which is usually referred to as a superset of all kinds of processes in a manufacturing company. Therefore, it includes all business processes as well as manufacturing processes in a company [32]. A *process operation* or a *process step* is an activity performed to produce the desired result or effect. Manufacturing processes are divided into discrete and continuous processes [141]. *Discrete processes* have an output that can be identified and is measurable by distinct units, while *continuous processes* have an output that can be identified and is measurable by mass or volume.

Manufacturing processes are executed in a *manufacturing system* or a *factory* that represents a complex set of physical elements that includes machines, people, materials-handling equipment, and tooling [2]. A *production system* is usually referred to as an enterprise system or an entire company, including all aspects of commerce, such as sales, advertising, profit, distribution, and a manufacturing system [2]. For the rest of this thesis, we will use the terms production process and production system, as we aim to cover different aspects of production in our research, but we mainly think of manufacturing processes and systems when we use these terms. Also, production process and production system terms are widely used in the literature and usually stand for manufacturing processes and systems.

*Production process specifications* are created by process designers. By the notion of *process designer*, we denote a person in charge of transforming a valuable idea or an experiment into an industrial process in a way to fulfill not only originality, efficiency, quality, and sustainability criteria but to consider many, often contradictory constraints. It includes all the users participating in process specification, such as process engineers, quality engineers, and resource managers. We should note that we use the syntagm "*process engineer*" to denote someone who specifies production processes, not only in the process industry but covering a wider range of processes.

As production processes change rapidly and numerous process variations emerge in the era of Industry 4.0, production systems need to adapt to these changes. Traditionally, to execute production processes, resources are arranged in a factory to form a production system [2]. However, to increase the adaptability of production systems, production processes need to be decoupled from them [65]. Therefore, production processes should be modeled independently from any production system, and production systems should be flexible enough to support various processes.

*Skill-based engineering* may be applied to decouple production processes from production systems. Machines and robots made by different vendors may participate in a single production system. Some may use low-level instructions at the level of sensors and actuators, while others may use high-level instructions, such as *pick*, *place*, and *move*. Even at the same abstraction level, instruction syntax may vary by a resource vendor. There are also human workers working in a production system that can use instructions in the form of textual descriptions. Managing production in such a heterogeneous system can be particularly difficult, and it requires specific knowledge from different domains and can be prone to errors [6]. Accordingly, there is a need to unify the way how instructions are sent to resources, which may be achieved through skill-based engineering [6,7]. Vathoopan et al. [7] defined skill-based engineering as "a very new method that describes the individual components, constituting an automation system as objects with their respective services, named skills".

A *skill* is usually referred to as a synonym for a *capability*, and a capability is defined as the "implementation-independent potential of an Industry 4.0 component to achieve an effect within a domain" [142]. There is also another view on skills and capabilities, indicating that a capability is a type of skill, and a skill is a parametrized and executable instance of a capability. We will use the term capability for the rest of this thesis as a synonym for a skill.

Production system resources may be seen as objects with their respective capabilities and standardized interfaces. Therefore, resources are offering a set of capabilities they have. Production

## 12 Background and Theoretical Foundation

process steps may require the capabilities needed for their execution. Accordingly, offered and required capabilities can be matched, thus matching process steps with resources that may perform them. After the matching process is finished, a system is needed to automatically transform matched resources and process steps into instructions to execute operations. These instructions can be sent to resources for execution. By using capabilities to describe resources and process steps, production process models can be specified independently from any specific production system. Also, process steps and resources are described in a uniform way, independent of the instructions' abstraction level and syntax.

To match process steps with resources, capabilities and their parameters should be specified in a uniform and standardized way [143]. Otherwise, the matching process would be compromised if synonyms are used for capabilities, the capability name is misspelled, or some unknown capability parameters are used. A standardization of capabilities and their parameters in a manufacturer-independent and cross-vendor way is still an open challenge [7,144–146], and such standardization is necessary to establish production steps for manufacturing [65]. As the standardization is still a work in progress, researchers have created capability taxonomies based on the mixture of different existing standards [144], such as the Association of German Engineers' VDI 2860 [147] (German: Verein Deutscher Ingenieure, VDI) for assembly and handling operations, and the German Institute for Standardization's DIN 8580 [148] (German: Deutsches Institut für Normung, DIN) for manufacturing processes in general. An example of such a capability taxonomy is presented by Hammerstingl and Reinhart [145]. The authors pointed out the necessity of the solution-neutral standardization of capabilities and their parameters for cross-vendor interoperability and proposed the capability taxonomy for assembly and handling operations. Another example is the SkillPro project [149], which aimed to unify and abstract capabilities provided by resources and required by production steps.

The Association of German Engineers (VDI) works on a standardized classification of capabilities in the domain of the assembly industry [7]. The standardization of assembly capabilities is expected soon under the standard VDI 2860. However, until the standardization of capabilities becomes realized, a capability repository containing a capability taxonomy with available parameters for each capability can help process designers specify production processes. Without the repository, capability names would be specified as arbitrary text, allowing process designers to misspell a capability or use a synonym for its name. Thus, an orchestrator or another intelligent system cannot match process steps with resources.

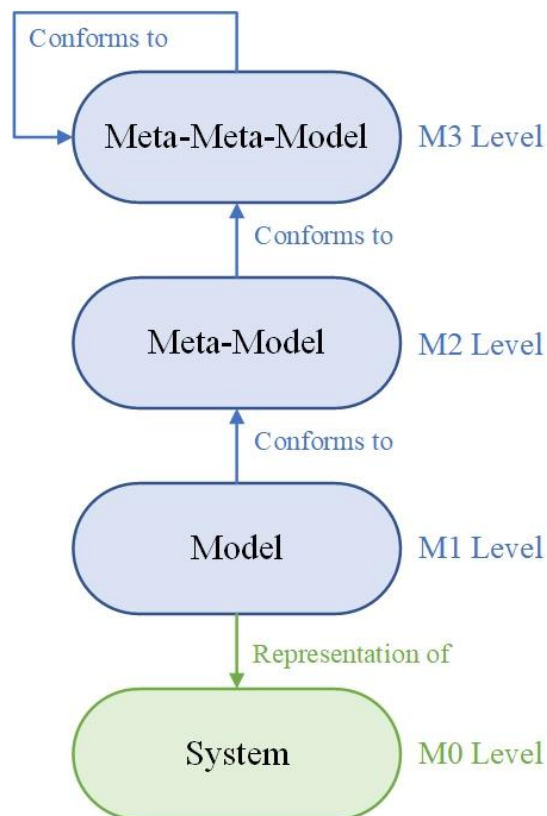
### 2.3 Overview of the MD Paradigm and DSMLs

*Model-Driven\** (MD\*) as a prefix is an umbrella term to indicate, among others: MD Engineering (MDE), MD Software Engineering (MDSE), MD Development (MDD), MD Software Development (MDS), and MD Architecture (MDA) [150], all discussed in this section. According to the MD paradigm, *models* represent a central artifact at all stages of system development. Although models have been used extensively in the software development process for decades, they were often used just for documentation purposes. Problems of growing complexity, number of different platforms, and interoperability in system engineering in general, as well as in software engineering, have motivated a paradigm shift. This shift caused a different usage of models, not just for documentation purposes but now to lead the system development.

By using MDD principles, software engineers usually want to cope with system complexity by raising abstraction levels. Therefore, by using the traditional four-layer infrastructure [151,152], MDD aims to automate many complex tasks. The *four-layer infrastructure*, depicted in Figure 2.2, represents a hierarchy of model levels, starting from M0 to M3. The bottom level (M0) represents a system under study in which observed entities exist. The level above (M1) represents a model of the system under study. A *model* is a representation of the observed entities, including only relevant information about each particular entity and particular relationships between such modeled entities. Therefore, it can be said that a model represents a system under study. A *modeling language* is

needed to create such models. To create a modeling language, a system under study needs to be observed, as well as the different types of entities that reside in the world, their relevant properties, and their relationships. A specification of entity types, relationship types, and their properties is called a *meta-model*, and it belongs at the next level of abstraction (M2). Every model created with a modeling language conforms to its meta-model. Similarly, a meta-model needs to be specified by using a language called *meta-modeling language*. Each meta-model is created using the same set of concepts defined in a *meta-meta-model* at the next level (M3). Therefore, a meta-model conforms to a meta-meta-model. Concepts defined in a meta-meta-model are independent of a particular domain and are defined by the environment in which meta-models are specified. As there is no need to introduce a level of abstraction above M3, a meta-meta-model is usually defined reflectively using its own concepts. One frequently used meta-modeling language is the Object Management Group's (OMG) Meta Object Facility (MOF) meta-meta-model. MOF is a self-defined platform-independent meta-data management framework that enables the development of model and meta-data driven systems, such as modeling and development tools [153]. One concrete implementation of MOF is the Eclipse Modeling Framework's (EMF) Ecore meta-meta-model [94,95].

At the M2 level, modeling languages, commonly referred to as *Domain-Specific Languages* (DSLs) or *Domain-Specific Modeling Languages* (DSMLs), are utilized. A DSL is a language tailored for a specific domain, in which concrete syntax is usually implemented as textual, such as the syntax programming languages have. A DSML is a language that can be seen as a specialization of a wider notion of DSLs [154,155], often providing users with a graphical notation instead of a textual syntax. The purpose of DSMLs is to bring modeling concepts closer to users familiar with an application domain, so that they can specify their solution in less time and with fewer errors in comparison to *General-Purpose Modeling Languages* (GPMLs). Solutions are specified faster and with fewer errors due to the usage of domain-specific modeling concepts that are more related to users, eliminating accidental complexity caused by a modeling tool or a GPML. The complexity that needs to be solved using DSMLs is caused by the domain itself and such complexity exists independently of whether DSMLs or GPMLs are used.



**Figure 2.2.** The four-layer infrastructure of model levels.

## 14 Background and Theoretical Foundation

MDS is a software development paradigm usually built around a DSML. The main goals of MDS include [156]:

- the increase of development speed through automation and single point of system definition;
- the increase in software quality through formalization;
- the increase in component reuse and improved manageability of complexity through abstraction;
- greater domain expert inclusion in the development process; and
- better communication between different stakeholders in the software development process.

MDA is currently the most mature formulation of the MDS paradigm and, according to Brambilla et al. [157], "the particular vision of MDD" proposed by OMG. MDA is a modeling framework that proposes the creation of models at different abstraction levels and applies the transformation methods between these models. There are three levels of abstraction defined in MDA [157]:

- *Computation-Independent Model (CIM)* – represents the context, requirements, and purpose of the solution at the highest level of abstraction. CIMs have no bindings to computational implications and also hide IT-related specifications. Therefore, such models aim to stay independent of how a system will be implemented. They are usually referred to as business or domain models, using concepts and terminology from the application domain.
- *Platform-Independent Model (PIM)* – represents the part of the CIM that will be solved using a software-based solution. It describes the behavior and structure of the solution, independent of the implementation platform. Thus, PIM can be mapped to one or more concrete implementation platforms. PIM includes information and algorithms independent of the implementation technology.
- *Platform-Specific Model (PSM)* – contains all required information about the behavior and structure of a system developed on a specific platform, which developers can use to implement executable code. PSM represents a technology-aware detailed specification of a system.

A Model-Driven Service Engineering Architecture framework is inspired by MDA and discussed by Vallespir and Ducq [158]. This framework applies MDA in smart manufacturing and has attracted increased interest in recent years. Since its original publishing, the framework has been generalized and renamed into Model-Driven System Engineering Architecture (MDSEA) [159] and is applied to manufacturing systems. Similar to MDA abstraction levels, MDSEA has three abstraction levels as well:

- *Business System Model (BSM)* – describes service systems and their communication at a high level of abstraction. BSMs are independent of the future technologies and skills that resources will use.
- *Technology Independent Model (TIM)* – the second abstraction level that provides a detailed specification of the structure, functionality, and operational details of the modeled service system. TIMs are also independent of the technological details that will be used for their implementation.
- *Technology Specific Model (TSM)* – enhances a TIM model with the implementation details, such as a machine technology or a specific person. TSMs are ready for the implementation of service systems. Based on TSMs, it is possible to implement a designed system in terms such as applications, services, machines, material handling, or human resources and organization, ensuring human-related tasks and operations.

Each BSM can be mapped to different TIMs, and each TIM can be mapped to different TSMs. The same can be stated for CIM, PIM, and PSM mappings.

MDSE is a methodology that applies the advantages of modeling to software engineering activities. Models created with DSMLs are changed and transformed over time, and both models

and transformations compose the following equation in the MDSE context: "Models + Transformations = Software" [157]. *Model transformations* represent operations on models, and they are specified once at the level of meta-models as a set of transformation rules and can be executed multiple times at the level of models. Each transformation has a source meta-model, a target meta-model, and mappings between their modeling concepts. Model transformations are specified using a transformation language that can be considered a DSL for model transformations. Depending on a transformation target, there can be Model-to-Model (M2M) or Model-to-Text (M2T) transformation types. There are also Text-to-Model (T2M) transformations, which are usually applied in reverse engineering.

## 2.4 Summary

In this section, we presented background and theoretical foundations related to the research presented in this Ph.D. thesis. The research comprises an application of the MD paradigm and DSMLs in the production process modeling domain in the Industry 4.0 era. Therefore, we presented an overview of Industry 4.0 and its concepts and technologies, process planning and design, and the MD paradigm and DSMLs.

The MD paradigm is already applied in the manufacturing domain, such as the MDSEA framework that provides different abstraction levels in manufacturing systems. Also, the MD paradigm is applied in process planning, allowing the creation of digital process plans that are computer interpretable, having exact and precise semantics on what and how will be produced. These applications, alongside the usage of skill-based engineering in production process design, provide us with the foundation to cope with the challenges (Ch1–Ch5) discussed in Section 1 and reach the main research goal. Therefore, by applying the MD paradigm and skill-based engineering, dynamic production management and automatic production process execution based on process models may be achieved, introducing factories into the digital transformation process.



## 3 Motivation, Research Hypotheses, Goals and Methodology

In this section, we present the motivation and define the goals of our research related to production processes modeling suitable for dynamic production orchestration, automatic instruction generation, and execution of instructions. We also lay out research hypotheses as well as expected research results and the expected value of the research contributions.

The research motivation in the domain of production process modeling is presented in Section 3.1. Research hypotheses are outlined in Section 3.2 and the research goals are presented in Section 3.3. Expected contributions and results are discussed in Section 3.4, while the methodology used to conduct this research is discussed in Section 3.5. This section concludes with a summary presented in Section 3.6.

### 3.1 Motivation

Industry 4.0 introduces many challenges traditional production systems must overcome to produce highly customized products. Producing such customized products in traditional production systems often requires multiple production lines, or in a case of a single production line, stopping production to allow the reconfiguration of machines. All of this incurs additional costs. The production needs to be carried out without stopping a production line for the machine reconfiguration in order to enable flexible, individualized, lot-size-one production that is economically viable [160]. Therefore, it is necessary to solve the problem of costly and time-consuming machine adaptation to frequent production changes, which is common in Industry 4.0. To enable such flexibility, the entire reconfiguration of the production line must be done "on the fly" by the automated mechanisms of a smart factory and based on the production process models. To be used by the automated mechanisms of a smart factory, production process models need to be formally described with attached machine-readable semantics.

The increasing demand for customized products required to be produced in small batches and in the least amount of time makes manual process planning difficult [132]. Therefore, a high degree of automation in process planning and manufacturing is necessary to stay profitable and competitive. However, completely automated process planning has not been fully realized yet due to the complexity and level of uncertainty in process planning and the difficulty of rationally capturing a process planner's expertise.

Besides formally specifying and planning production process execution, error handling needs to be considered in order to run flexible production smoothly. Error handling is not crucial to support automatic production management and execution, but it is of utmost importance in creating flexible production. By implementing error handlers within production process models, any known issue or failure that may occur is formally specified as well as corrective steps on how to solve such

issues and failures. Therefore, if an error occurs, a well-established procedure will be performed to minimize damage caused by the error and continue with production as soon as possible to minimize costs caused by production being stopped. Some errors will still cause production to be stopped, but the reaction to errors will be nearly instant and automatically performed. In contemporary production systems, known errors and issues are mostly well documented but written in the form of textual descriptions and thus cannot be used for the automatic elimination of error effects and the recovery from an error occurrence.

Additionally, as contemporary customers want their products custom-tailored to fit their needs, the number of product variations and corresponding production processes increases significantly in comparison to mass production. Consequently, this increase is followed by an increase in the number of product and process specifications and required documentation. Therefore, it is essential that process models are easy to change and adapt to new process and product variations, enabling automatic generation and update of manufacturing documentation. Due to many products and product variations in customized, lot-size-one production, keeping manufacturing documentation up to date manually becomes particularly difficult. Different documents need to be updated if a product is changed in any way. Having a single point of knowledge stored in production process models can facilitate the automatic update of documentation or the automatic creation of a new version of the documentation, decreasing the amount of manual work that needs to be performed by process designers. Issues associated with keeping the manufacturing documentation up to date manually include redundant specifications, human-related errors which occur while writing or modifying documentation, and the long time needed to create such specifications. These issues could be mitigated with the automatic generation of manufacturing documentation from process models.

There is also a problem of frequent position changes and relocations of human workers in a factory [161]. Due to a decreasing number of human workers and an increasing level of automation in factories, workers are required to perform different tasks. Frequently changing workers' tasks leads to increased production dynamics and requires fine coordination of workers in a factory, so their work can be optimized, and production downtime avoided. As workers often switch between tasks, especially when allocated to a new workplace, fast knowledge transfer is required, so they do not lose time when changing workplaces. Also, fast knowledge transfer is needed when novice workers start producing new products, as expert workers need to invest their time in helping novices. Guided production can be based on production process models, thus sending production steps one by one to workers, including production descriptions, audio, images, and video footage. Accordingly, time spent by expert workers when helping novices can be reduced and novice workers may perform tasks more efficiently.

However, including all the details needed for flexible production, error handling, management of process and product variations, creation and modification of manufacturing documentation, and human workers' training and guided production makes production process modeling difficult. Therefore, different *levels of detail* and *modeling layers* need to exist in order to ease the modeling task done by process designers. With different levels of detail, production process models can be specified independently of any production system. The separation of production process models from any specific production system is one of the main challenges of Industry 4.0 production process specification. A production process model should be machine-readable, allowing automatic execution of process steps, but also, a production process model should be independent of any production system in which it is to be executed, enabling production process models to be used in multiple production systems [65]. To achieve such a goal, process designers should model a production process without details specific to a production system, and an intelligent system, such as an orchestrator, should automatically enrich the model with details specific to the production system which is chosen to execute the production process. Thus, process designers do not need to think of production resources or production logistics, creating *resource-agnostic* process models. Such process models can then be used in various production systems, or different parts of process models can be used in different production systems. To execute such models, they need to be enriched with data from specific resources automatically, creating *resource-aware* process models.



The execution performance of such process models can be estimated and compared for different production systems before they are executed, thus providing an option to choose a production system that would bring the best overall performance.

The research challenges (Ch1–Ch5) presented in Section 1 and the research problem discussed in this section, motivated us to create a novel solution for production process modeling and execution. We believe that a novel MD solution with a DSML for production process modeling can be a basis for improving production flexibility in a smart factory. Such a DSML needs to be used to create machine-readable models, both resource-agnostic and resource-aware production process models, allowing the automatic transformation of models into executable resource instructions and manufacturing documentation. Therefore, production process models are not used just for documentation purposes but to guide production process execution.

In the following section, we formulate hypotheses related to this research, based on the motivation discussed in this section.

## 3.2 Research Hypotheses

Considering the research problem and motivation outlined in the previous section, we formulate the main hypothesis of our research.

**Hypothesis 0 ( $H_0$ ).** *It is possible to create an MD solution for resource-agnostic and resource-aware specification of production processes and automatic transformation of such specifications into executable resource instructions and manufacturing documentation.*

The main task of our research, derived from the main hypothesis, is to define a methodological approach and a software solution that utilizes MD principles and a DSML to specify production processes formally, contributing to flexible production and increasing customer demands, and helping process designers in modeling production processes and keeping the manufacturing documentation up to date in a more efficient manner. Furthermore, such formally specified production process models need to be machine-readable, thus enabling the automatic generation of executable resource instructions and manufacturing documentation.

The following four hypotheses are derived from the hypothesis  $H_0$  to better address different aspects of the main hypothesis. By confirming or rejecting the derived hypotheses, we will confirm or reject the main hypothesis  $H_0$ .

**Hypothesis 1 ( $H_1$ ).** *It is possible to create a DSML that can be used to model production processes with all the details required for the automatic generation of executable resource instructions.*

Production processes need to be modeled by using a DSML in a coherent and concise way, thus creating machine-readable process models. By enriching process models with all the necessary details needed to execute production processes, such models can be automatically transformed into executable resource instructions which are then sent to the factory's shop floor.

Production process models that contain enough details to be used for automated production, often need to be modeled by different users, such as process and quality engineers. Therefore, models must include different aspects and viewpoints of production process modeling, such as process execution, error handling, and quality and safety aspects. All these different aspects and viewpoints need to be unified within a single process model, thus creating a single point of knowledge about production processes. Therefore, the second derived hypothesis is defined as follows.

**Hypothesis 2 ( $H_2$ ).** *It is possible to represent different aspects of production processes in a uniform, resource-agnostic way, thus allowing for model reuse in different production systems while maintaining model readability and understandability.*

## 20 Motivation, Research Hypotheses, Goals and Methodology

Instruction-generation-ready production process models become overloaded with details about resources that execute production process steps, production logistic activities, and machine configuration activities. It would be hard to specify such resource-aware process models manually. Process designers need to specify production process models independently of any specific production system, thus not considering resources and transportation and configuration process steps. Such resource-agnostic process models could be used in any production system and would be easy to read and understand.

However, resource-agnostic models do not contain enough information to be transformed into resource instructions to execute process operations. Therefore, resource-aware production process models need to be created based on resource-agnostic process models and production system details in which models are to be executed. Resource-aware process models could be created manually by process designers, which would be a burdensome task, or they can be created automatically by an intelligent system, such as an orchestrator. As modeling concepts related to production systems need to be included in the DSML for production process modeling, process designers will use such concepts to manually change or optimize resource-aware process models created by an orchestrator whenever necessary. The need for the DSML to contain modeling concepts related to production systems to enable manual or automatic enrichments of resource-agnostic process models is presented in the following hypothesis.

**Hypothesis 3 (H<sub>3</sub>).** *It is possible to create a DSML comprising modeling concepts that allow enriching the resource-agnostic production process models with details about a specific production system, thus creating resource-aware production process models.*

To be independent of any production system, the modeling of resource-agnostic production process models can rest on the main concepts of skill-based engineering. Thus, production process steps can be described with the capabilities required for their execution. Resources of production systems can offer a set of capabilities they have. Accordingly, these required and offered capabilities can be matched automatically, thus matching process steps and resources. An orchestrator is able to perform such an automatic matching, which can be seen as an M2M transformation, automatically creating resource-aware process models from resource-agnostic process models.

Besides matching process steps with resources to execute them, an orchestrator needs to automatically add transportation and configuration process steps as well, based on the production system topology and logistics. An orchestrator with its matching algorithms is not a part of this research and is used as a black box, while more details about the orchestrator can be found in [4,5]. With this automatic enrichment of resource-agnostic process models, process designers can create production process models independent of a production system yet be able to automatically prepare them for execution, making the process modeling task easier. To execute production processes, their models need to be transformed into executable resource instructions, as described in the following hypothesis.

**Hypothesis 4 (H<sub>4</sub>).** *It is possible to automatically generate executable resource instructions and manufacturing documentation of different types from production process models.*

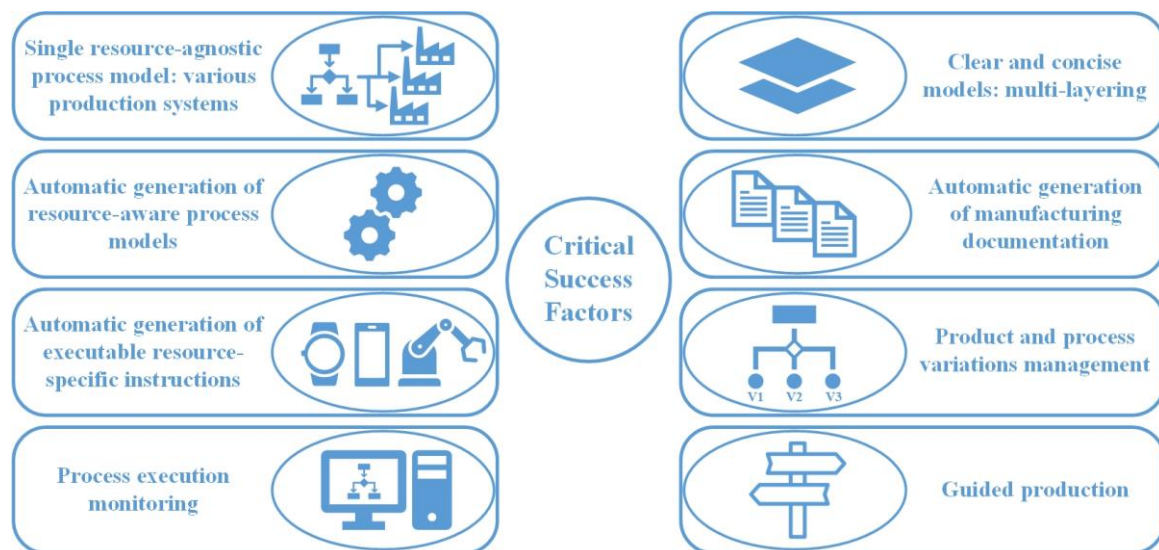
By using an instruction generator with M2T transformation rules, resource-aware production process models can be automatically transformed into executable resource instructions. The generated instructions can be sent to resources for execution, thus completing the main purpose of creating production process models – to lead production process execution on the shop floor. Also, as production process models are machine-readable, they can be transformed into manufacturing documentation. Such a transformation would help process designers avoid keeping the manufacturing documentation up to date manually. Therefore, process designers' time can be saved due to the automatic generation of manufacturing documentation, and errors caused by manual documentation editing can be mitigated.

In the following section, we discuss research goals and critical success factors derived from the research hypotheses presented in this section.

### 3.3 Research Goals

The main goal of our research is to introduce factories into the digital transformation process, by providing dynamic production management and automatic production process execution based on process models. To accomplish the main research goal, the following critical success factors should be achieved, as summarized in Figure 3.1:

- The usage of a single production process model in various production systems should be provided, by creating a resource-agnostic production process model, independently from any production system. Thus, process designers can model a production process more easily, without the need to have knowledge about a production system in which the process is to be executed.
- Process designers should be released from the burdensome manual task of creating a resource-aware production process model, specific to a production system. Therefore, the automatic transformation of resource-agnostic into resource-aware production process models should be provided. However, manual resource-aware model changes should be allowed whenever seem necessary by process designers.
- A resource-aware production process model is enriched with production system details, such as smart resources, production logistic activities, and machine configuration activities. Such a production process model should lead production process execution automatically. Thus, the automatic transformation of a resource-aware production process model into executable resource-specific instructions should be provided.
- Tracking and fixing badly modeled processes and production errors should be enabled with the mechanisms for production process execution monitoring.
- Different user groups should be focused only on production process aspects they are responsible for, by creating different views on a process model through different modeling layers. Therefore, a production process model represents a single point of knowledge and different users can participate together when modeling different aspects of a production process. By adding or removing modeling layers from a production process model, it should be possible to show or hide different aspects of a production process, making its model clearer and more concise.



**Figure 3.1.** Critical success factors of the novel MD solution.

- Process designers should be released from keeping the manufacturing documentation up to date manually. Numerous manufacturing documents of different types should be generated and updated automatically, by transforming a production process model into the documentation.
- All product and process variations should be stored in a single production process model. However, as too many variations may exist in a process model, making it hard to read and maintain, mechanisms to choose a variation to be focused on should be provided.
- Novice workers should be provided with automatic help during the training and creation of products, and the time spent by expert workers when helping novice ones should be reduced. This can be achieved by using production process models for guided production, sending instructions one by one to human workers, alongside images and video footage on how to perform the instructions, generated from a process model.

In the research presented in this thesis, we utilize MD principles, creating a novel approach to the production process modeling and execution. Core elements of the approach are a novel DSML and a tool for production process modeling and code generators for the automatic transformation of production process models into executable resource instructions and manufacturing documentation. In the following section, we discuss the expected contributions and results of our research.

### 3.4 Expected Contributions and Results

We expect the following theoretical, development, application, and socio-economic contributions from our research:

- **Theoretical contributions** in the fields of production process modeling and automatic process execution are identified as follows:
  - survey on existing languages and approaches for production process modeling;
  - identification of main concepts needed to implement a DSML for production process modeling in the context of Industry 4.0, which models are suitable for the dynamic orchestration and the automatic generation of executable resource instructions and manufacturing documentation;
  - specification of an MD solution for dynamic orchestration and automatic instruction and documentation generation based on DSML models;
  - specification of a methodology to automatically transform resource-agnostic process models into resource-aware process models; and
  - application of MD principles in the production domain, contributing to easier production process modeling and automatic execution of processes.
- **Development contributions** by implementing a tool for production process modeling and code generators for the automatic generation of executable resource instructions and manufacturing documentation are identified as follows:
  - developed and implemented a novel modeling tool that utilizes the DSML for production process modeling;
  - developed and implemented an instruction generator for the automatic generation of resource instructions to execute process operations based on DSML models; and
  - developed and implemented documentation generators for the automatic generation of different manufacturing documentation based on DSML models.
- **Application contributions** are reflected through several use cases in which the MD solution is applied and the evaluation process of the DSML and its modeling tool. These contributions include:
  - demonstration of a practical application of the MD solution and the DSML for production process modeling in the assembly industry;
  - evaluation of the DSML and the modeling tool for production process modeling by users from different user categories; and

- presentation of a new practical experience from applying a novel methodological approach, a modeling tool, and a DSML.
- **Socio-economic contributions** by putting into public use a general model of production process management, applicable in a wide range of organizations in a way that enables significant production process improvement and raises general accumulated knowledge on how to contribute to such a process improvement contemporarily.

The main **expected result** of our research is easier and simpler formal modeling of production processes in the era of Industry 4.0. Created models are to lead the process execution in flexible production and keep the manufacturing documentation up to date automatically, thus responding to increasing customer demands for individualized products and coping with Industry 4.0 challenges (Ch1–Ch5). The expected end-users of the production process modeling tool and language are process designers, consisting of process and quality engineers, who need to specify production process models.

In the following section, we present a research methodology we follow in order to investigate the problem domain, design, develop and evaluate the solution, publish the research results, and confirm or reject the research hypotheses.

### 3.5 Research Methodology

The real world is being modified by humans as they design and develop tools to change its environment, making it more artificial [162]. The design of tools is concerned with how developed artifacts will change the real world to attain specific goals. Nowadays, these tools and artifacts are usually based on IT, making the world even more artificial. The design of an IT system is required to make progress in solving real-life problems and understanding how and why the system works. Researchers in IT need to conceptualize and represent real problems, construct appropriate techniques for their solution, and implement and evaluate the solution using appropriate criteria [163]. These steps lead us to the *design science* paradigm, which aims to develop ways to achieve human goals. The design science paradigm's goal is to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts [164].

Hevner et al. [164] described the characteristics of the Design Science Research (DSR) paradigm in ISs. DSR is a problem-solving paradigm aiming to enhance technology and science knowledge by creating innovative artifacts that solve real-world problems and improving an environment in which the artifacts are instantiated [164,165]. The DSR results consist of newly designed artifacts and design knowledge, providing a better understanding of why the artifacts contribute or not to the application environment.

There are several process models on how to perform DSR [165], but the most referenced DSR process model is proposed by Peffers et al. [166]. The authors presented Design Science Research Methodology (DSRM), incorporating principles, practices, and procedures required to produce and present research in the IS field. A commonly accepted methodology is provided by the authors for successfully carrying out design science research. The methodology helps with recognizing and legitimizing the research, its objectives, processes, and outputs, as researchers follow the commonly understood methodology instead of doing ad-hoc research. The methodology introduces six activities when performing the DSR process [165,166], which we utilize in our research to define the research plan. In the following list, we describe the activities we performed when we conducted the research presented in this thesis:

- **Activity 1.** First, we identified research problems that are present in contemporary production systems aiming to become flexible and responsive to customer needs in the era of Industry 4.0. The identification of research problems and the motivation corresponds to the first DSRM activity. Challenges that appear in the era of Industry 4.0 related to our research and the motivation to cope with these challenges are discussed in Section 1 and Section 3. The decision to create a novel DSML – MultiProLan is based on the research

conducted and presented in Section 4. In the research presented in Section 4, we formulated requirements for a production process modeling language in the era of Industry 4.0 needs to fulfill and analyzed various existing languages whether they fulfill these requirements. According to the reviewed literature, technical documentation, interviewed domain experts, use cases, and the Industry 4.0 environment, we have analyzed a domain of production process modeling suitable for automatic instruction generation and execution. The domain analysis has been performed by means of the Feature-Oriented Domain Analysis (FODA) method [93] and the results are given in Section 6.

- **Activity 2.** After the research problems were identified, i.e., challenges that Industry 4.0 introduced, we defined the objectives that our proposed solution aims to achieve and the boundaries of our solution. Defining the objectives of the proposed solution represents the second DSRM activity. The goals of our research are presented in Section 1 and Section 3, while the objectives of our MD solution, MultiProLan, and the process modeling tool, are outlined in Section 5.
- **Activity 3.** The specification and development of our MD solution, comprising of a novel DSML – MultiProLan, the production process modeling tool, and various code generators, are done after their objectives were defined. The design and development of the solution is part of the third DSRM activity. The design of our MD solution and MultiProLan and the technologies we use to implement the solution and the language are presented in Section 5. Implementation details, abstract and concrete syntaxes of MultiProLan, and its process modeling tool are presented in Section 7.
- **Activity 4.** Our MD solution and MultiProLan are applied in two proof-of-concept use cases. In the first use case, a customized wooden box is assembled, demonstrating the possibilities of MultiProLan. The second use case represents a demonstration environment created to test the whole solution in assembling objects from LEGO® bricks. These use cases are presented in Section 8. The demonstration of the solution is part of the fourth DSRM activity.
- **Activity 5.** MultiProLan and its modeling tool are the main components of our MD solution for production process modeling and execution. Therefore, we evaluated their quality characteristics, by applying the Framework for Qualitative Assessment of Domain-specific languages (FQAD) [101]. The evaluation participants performed an experiment in which they modeled production processes and evaluated the following quality characteristics of MultiProLan and the process modeling tool: functional suitability, usability, reliability, expressiveness, and productivity. The evaluation results are presented in Section 9. The evaluation of the solution corresponds to the fifth DSRM activity.
- **Activity 6.** Research and development results of production process modeling and execution are presented both within the academic community and in the industry. We presented our research results in international conferences and journals and gathered feedback from reviewers and conference participants about our MD solution and MultiProLan. We also present these results in this thesis. Our research and development results are discussed with relevant stakeholders in the industry, and we applied the whole MD solution in an industrial demonstration environment. The presentation and discussion of contributions and results of the developed solution and the whole research are part of the sixth DSRM activity.

DSRM activities 1 through 5 related to our research are presented in various sections of this thesis, while the sixth DSRM activity related to our research results is presented in the rest of this section. The initial research proposal in the context of Industry 4.0 manufacturing and the first draft of our MD solution was presented at *International Scientific Conference on Informatics (Informatics)* [8]. This research proposal was extended with more details, and an overview of existing production process modeling languages with a pilot comparative analysis of a few languages was published in the *Open Computer Science* journal [9]. The DSML for production process modeling named MultiProLan was first introduced at *Federated Conference on Computer Science and Information Systems (FedCSIS)* with its abstract and concrete syntaxes and the wooden box production example [10]. The improved abstract and concrete syntaxes were then presented in

*Journal of Computer Languages*, alongside the production process modeling domain analysis and the evaluation of MultiProLan and its process modeling tool [11]. The automatic generation of manufacturing documentation from MultiProLan models was introduced at *International Conference on Information Society and Technology (ICIST)* together with the needed abstract syntax extensions and a new production process modeling example [12]. An application of MultiProLan in an assembly use case, creating a customizable flag from LEGO® bricks, was presented at the *Advances in Production Management Systems (APMS)* conference [13]. Finally, an overview of the whole MD solution for production process modeling, with a new assembly use case and more complex process model examples are presented in *International Journal of Production Research*, alongside the introduction of language requirements and a comparison of process modeling languages based on these requirements [14].

In addition to the papers published during our research, other authors have published research in which we also participated, and which was inspired by the outcomes and artifacts of our research. The first overview of the orchestrator's architecture, which we are using in our research, was presented by Pisarić et al. at the *Advances in Production Management Systems (APMS)* conference [4]. An extended and more detailed version of the orchestrator's architecture was presented in the *Applied Sciences* journal [5]. This paper also includes an assembly use case consisting of MultiProLan process models and production system models, introduced for the first time, and created by the resource modeling tool we are using in our MD solution.

In addition to the production process and production system modeling, Antanasijević et al. [15] proposed research on human resource modeling in the context of Industry 4.0 and presented it at *International Scientific Conference on Informatics (Informatics)*. Based on the extensions of MultiProLan and the production system modeling language, the authors plan to create a new language for the formal specification of human workers' roles, skills, competencies, capabilities, and limitations to achieve better integration of human workers and machines in a smart factory. The DSML named Human Resource Modeling Language (HResModLan) is in the development phase, aiming at human resource modeling from two different perspectives: production and organizational. The first prototype of the HResModLan's production perspective, aimed at the formal specification of a human worker as a production resource, is presented in the *Acta Electrotechnica et Informatica* journal [16] and will be further developed in the future. The organizational perspective of the language named HResModLan:Org is already created and presented in the *IPSI Bgd Transactions on Internet Research* journal [17] with the aim to provide the easier and more effective requiring, selection, hiring and development of human workers within an organization.

A research proposal and an MDS approach to enable the automatic generation of smart contracts used by collaborative parties to supervise the state of production was presented by Todorović et al. at *Federated Conference on Computer Science and Information Systems (FedCSIS)* [18]. This research proposal aims to connect collaborative parties when production is shared between them and ensure that production is conducted according to the agreed conditions. To model collaborative production processes in the proposed MDS approach, a new MultiProLan-based DSML, named Collaborative Extension of MultiProLan (CE-MultiProLan), was created by Todorović et al., and its model examples were presented at *International Conference on Innovative Intelligent Industrial Production and Logistics (IN4PL)* [19].

The research results presented in this Ph.D. thesis are also part of the sixth DSRM activity. The discussed research plan, consisting of six DSRM activities, creates guidelines to achieve the research goal: to introduce factories into the digital transformation process. Before creating a novel software solution that aims to help us achieve the research goal, the research problem, motivation, and objectives of the solution must be precisely defined. After the design and development of the novel software solution, its performance must be evaluated and demonstrated in various use cases. By presenting the solution to the academic community and in the industry, useful feedback and future research directions can be gathered. Therefore, we applied the DSRM methodology in our research to systematically develop the novel MD solution for modeling and automatic execution of production processes, discussed in detail in the following sections.

### 3.6 Summary

In this section, we presented the research motivation, hypotheses, goals, expected contributions and results, and the methodology used to conduct the research. To cope with the Industry 4.0 challenges (Ch1–Ch5) and introduce factories into the digital transformation process, we proposed an application of the MD paradigm and a DSML to provide a formal specification and execution of production processes. Accordingly, we formulated the main hypothesis of the research, in which we stated that it is possible to create a novel MD solution that would provide the specification of resource-agnostic and resource-aware production process models, used for the automatic generation of executable resource instructions and manufacturing documentation.

Based on the background and theoretical foundation presented in the previous section, and the discussion provided in this section, we believe that it is possible to create such an MD solution for production process modeling and execution. We also investigate the application of the MD paradigm in ISs and Industry 4.0 in the following section, as the proposed MD solution could be seen as an IS applied in the Industry 4.0 context. The central part of the proposed MD solution would be a DSML for production process modeling, whose models would be suitable for dynamic production orchestration and automatic execution. Therefore, in the following section, we also investigate existing languages and approaches for production process modeling, in order to explore whether there is an existing language or approach that fits our needs.



## 4 State-of-the-Art

To achieve the research goals presented in the previous section, we examined the literature, gathering knowledge related to the MD paradigm, DSLs, Industry 4.0, and production process modeling. Accordingly, we investigated the state-of-the-art presented in this Ph.D. thesis based on the following aspects related to our research, especially on production process modeling languages and approaches.

In Section 4.1, we discuss the application of the MD paradigm and DSLs in the ISs and Industry 4.0 domains. In the domain of ISs, we examined the application of MD principles in the design and development of ISs, the integration of heterogeneous technical spaces, document engineering, and measurement and control systems. These aspects are relevant to our research as we plan to develop a new MD solution that can be seen as an IS used for production process modeling and execution. In the domain of Industry 4.0, we investigated the application of MD principles in general, particularly in the aspects related to the challenges we discussed in Section 1. The aspects related to Industry 4.0 we investigated are error handling in production processes, manufacturing documentation, guided production, and process modeling.

In Section 4.2, we discuss production process modeling, the main aspect of the state-of-the-art related to this thesis. As the main part of the proposed MD solution would be a DSML for production process modeling, we examined whether there is an existing process modeling language that fits our needs. Therefore, we defined the requirements a production process modeling language needs to fulfill in order to be used for dynamic production orchestration and automatic execution of production processes in the Industry 4.0 era. Then we analyzed various existing modeling languages and approaches that could be used for such a purpose, based on the defined requirements.

The summary of the state-of-the-art related to this Ph.D. thesis is outlined in Section 4.3.

### 4.1 Application of the MD Paradigm and DSLs

The MD solution for production process specification and execution we propose in this Ph.D. thesis can be seen as an IS applied in the manufacturing domain of Industry 4.0. Thus, in this section, we present an overview of the related research and the application of the MD paradigm and DSLs in ISs and Industry 4.0.

In Section 4.1.1, we discuss the research related to the MD paradigm and DSLs in the field of ISs, conducted by the Data Science and Information Systems group. In Section 4.1.2, we outline the related research in the domain of Industry 4.0 and the application of the MD paradigm and DSLs in such a domain.

### 4.1.1 Information Systems

Research related to the MD paradigm and DSLs in the field of ISs has a long history in the Data Science and Information Systems group of the Faculty of Technical Sciences, University of Novi Sad. The group has been applying MD principles and DSLs in the domains of the design and development of ISs, integration of heterogeneous technical spaces, document engineering, and measurement and control systems, among others. The research presented in this Ph.D. thesis was partially conducted in the Data Science and Information Systems group, having foundations on the knowledge and experience that the group has gathered throughout the years.

In this section, we discuss the research results of the Data Science and Information Systems group, related to the application of the MD paradigm and DSLs in various domains, presented in the following subsections. In Section 4.1.1.1, we discuss the IIS\*Case approach created for the design and development of ISs. The integration of heterogeneous technical spaces, based on the MD paradigm, is outlined in Section 4.1.1.2. The MD paradigm and DSLs in document engineering and robot motion control are discussed in Section 4.1.1.3.

#### 4.1.1.1 Information System Design Based on the IIS\*Case Approach

The Integrated Information Systems\*Case (IIS\*Case) approach and tool have been developed for decades by researchers from the Data Science and Information Systems group. The tool has been used for the design and development of various aspects of ISs, and some of these aspects are discussed in the rest of this section.

To design database schemas, various approaches and techniques can be used, such as Entity-Relationship (ER) modeling or Unified Modeling Language (UML), with an appropriate Computer Aided Software Engineering (CASE) tool. However, these approaches and techniques are often incomprehensible to end-users, which may lead to misunderstandings between designers and end-users when designing a database schema. To overcome such an issue, Luković et al. [167] proposed a new approach and a CASE tool for automated database schema design based on the concept end-users are familiar with – the form type concept. It is the central concept of the presented IIS\*Case tool for the design and development of complex database schemas, representing screen forms that end-users are familiar with when communicating with an IS. The IIS\*Case tool is used to design a database schema based on form types and generate an implementation of the database schema automatically. As such an approach and a tool do not require advanced knowledge related to database schemas, they can simplify the design and development of database schemas, avoid or alleviate misunderstandings between designers and end-users, save work time, and lower the designers' required effort.

Besides designing database schemas, IIS\*Case is also used for the conceptual modeling of business applications and for generating application prototypes. Luković et al. [168] presented a DSL and a tool embedded into IIS\*Case for the specification of check constraints and complex functionalities of business applications. The DSL is developed to provide the specification of check constraints on the PIM level, allowing designers to specify check constraints using problem domain concepts. Business application functionalities are also specified at the PIM level in IIS\*Case in a visually oriented way. Therefore, designers are not burdened with the implementation details and can be focused on the problem domain. To automatically transform PIMs of check constraints and application functionalities into PSMs and program code, M2M and M2T transformations are implemented. Therefore, fully executable application prototypes can be generated automatically.

The IIS\*Case PIM concepts are specified in a formal way through the meta-model by using the Ecore meta-meta-model [94,95]. Ćeliković et al. [169] presented a part of the IIS\*Case meta-model, its concrete textual syntax, and an application of the language by creating an IS model example. The meta-model can be used for the verification of generated database schemas, enabling the detection and resolution of formal conflicts at the level of the database model, such as various constraint collisions.

As the database reverse engineering process can be used to transform a physical database schema into a conceptual or a logical database schema, or even an application prototype, Ristić et al. [170] outlined the importance of using meta-models and transformations in such a process. The authors presented part of meta-models and the M2M transformation between physical database schema and generic relational database schema. By integrating meta-models and a chain of M2M and M2T transformations into IIS\*Case, the reverse engineering process can be performed, allowing IIS\*Case to transform legacy relational database schemas into executable application prototypes.

IIS\*Case, which was renamed into Integrated Information Systems\*Studio (IIS\*Studio) after many languages and extensions were integrated into it, has been further improved and developed nowadays. New technologies and platforms are used to create new versions of IIS\*Studio and introduce novel features. For example, Luković et al. [171] presented a part of the IIS\*Case PIM meta-model, its novel graphical concrete syntax, and a code generator developed on the ADOxx platform [172–174]. The concrete graphical syntax was created as some designers may use it more efficiently than the textual one. However, the evaluation of the user experience with IIS\*Case textual and graphical syntaxes is left for future research.

Future research and development related to IIS\*Case also comprise the support and application in contemporary fields of the IS development domain. The research conducted by Terzić et al. [175] resulted in the development of a new approach for the specification of microservice software architecture. The presented research was derived from the IIS\*Case ambient, and the IIS\*Case approach is planned to support the specification of such architectures. Terzić et al. presented MicroBuilder which comprises MicroDSL and MicroGenerator modules. MicroDSL is a DSL for the specification of REpresentational State Transfer (REST) microservice software architectures, while MicroGenerator is a code generator used to generate executable program code from MicroDSL models. Therefore, the process of microservice software architecture specification and configuration can be automated and more easily performed.

As the main task of the research presented in this Ph.D. thesis is the development of a novel MD solution for modeling and automatic execution of production processes, the accumulated knowledge and experience in developing IIS\*Case can help us create such a novel MD solution. The solution represents an IS that utilizes MD principles and has various DSMLs, modeling tools, and code generators.

#### 4.1.1.2 Integration of Heterogeneous Technical Spaces

The exchange of models between different modeling tools is necessary as usually various modeling tools are used to complete a certain task in an organization. Also, the exchange of models is needed when replacing an old modeling tool with a new one that better fits the users' needs. Therefore, Kern et al. [176] presented a mapping-based approach to realize the exchange of models between tools by connecting their meta-models. The central part of the approach is a declarative mapping language and a tool named AnyMap that allows the specification of mappings between source and target meta-models. The approach was created to allow efficient and user-oriented import and export of models in various modeling tools.

In the IoT environment, various devices are connected and they need to exchange data continuously. Accordingly, similar to the exchange of models between tools, different schemas of transmitted and received data between IoT devices need to be connected. IoT devices often communicate using different protocols and send data belonging to different technical spaces. Thus, Dimitrieski et al. [156,177] used the AnyMap language and tool to provide a uniform way of creating transformations between technical spaces. The authors proposed an algorithm that reuses previously created mappings to automatize the process of adapting new mappings to the schema variations, while keeping an option to manually refine the new mappings when needed. The algorithm is evaluated in a case study related to the integration of sensor devices and MESs.

To enable continuous information flow in Industry 4.0 production, the integration of ISs and machines on the shop floor is required, which is a challenging, time-consuming, and expensive task.

Such integration is required as machine interfaces may differ and are often adapted for a certain domain, manufacturer, or machine itself. Kern et al. [178,179] presented a novel approach and a framework for a structured, automated, and reusable integration of ISs and machines on the shop floor. The authors used the AnyMap language and tool, and an intelligent solution for connecting different systems that rely on the integrated, machine-independent learning mechanism, allowing a systematic reuse of integration knowledge from previous projects. By using the proposed framework, a specification of new mapping can be automated when a new machine arrives on the shop floor, and existing machines can be connected with an IS with minimum impact on the system, thus preparing them for the Industry 4.0 environment.

The integration of heterogeneous technical spaces is also needed in the research presented in this thesis. In the proposed MD solution, we plan to match capability-based production process models with the models of smart resources working on the shop floor. As production process steps require capabilities for their execution, and smart resources offer various capabilities they have, the integration of their models can be achieved. Therefore, it would be possible to transform resource-agnostic into resource-aware production process models automatically.

#### 4.1.1.3 MD Paradigm and DSLs in Document Engineering and Robot Motion Control

Document engineering refers to the activities of specification, implementation, and usage of documents in an organization. As documents can be frequently refined and updated, new solutions are needed to keep the documentation up to date easily. One way to formally specify and render documents is to use DSLs, thus allowing the automatic update of the documentation. Djukić et al. [180] proposed a framework and DSLs for the formal and incremental specification and rendering of documents in directory publishing. The authors presented DSLs used in document engineering, for the modeling of small advertisements and business activities related to documents and their content units. The framework that contains these languages enables the automation of document engineering, by providing the generation of documents in an organization.

Djukić et al. [181] presented an approach in which DSLs and their tools are used as client applications in the areas of document engineering and measurement and control systems. Action reports are used in the presented approach as special transformations that, in addition to the description of M2T transformations, contain commands and rules for invoking commands during model execution. The usage of the approach with action reports allows the specification of processes for documenting model validation and the synchronization of actions on a model to the state of the real system. Such synchronization can be applied in production systems as there is a need to document each action of business procedures to models or to execute each action on models by relying on the previously generated documents.

The same authors also presented an approach to handling frequent variations of modeling languages and models for robot motion control [182]. As there is a need for improvements in software development in automation and robot control, particularly the development of tools for formal specification and execution of control processes and the creation and application of robot-motion control languages, the authors proposed a new MD approach. The approach may contribute to the development of intelligent robot controllers and the development of measurement and control systems.

The experience gathered from document engineering, especially in measurement and control systems, can help us in creating a solution for keeping the manufacturing documentation up to date effortlessly. The development of an approach for robot motion control also helps us understand the way robots are utilized, which are particularly used in the Industry 4.0 era.

## 4.1.2 Industry 4.0

In recent years, we are witnessing that the MD paradigm, meta-modeling, and DSLs are increasingly applied in the context of Industry 4.0 [183]. They are particularly used in:

- development and management of CPSs and smart manufacturing [184,185];
- integration of heterogeneous technical spaces [156];
- fields of IoT, manufacturing systems, and multi-agent systems [186];
- fields of robotics, the development of software for robots [187], and mobile robotic systems [188]; and
- formal description of processes, process information exchange, decision-making support, simulation of manufacturing systems, and material flow systems [189].

The MD paradigm, DSLs, meta-modeling techniques, and model transformations are helpful when needed to make a higher abstraction level of a domain problem and contribute to solving problems systematically. Therefore, we believe that DSLs may contribute to automatic production by formally defining production process models and automatically generating instructions to execute process operations using model transformations and following MD principles.

Various technologies and environments for meta-modeling are used in Industry 4.0 in general and particularly in CPS, IoT, robotics, and process modeling domains. Most commonly used environments and languages to develop DSLs, meta-models, and tools are UML [43], Ecore within the modeling platform EMF [94,95], Generic Modeling Environment (GME) [190,191], MontiCore [192,193], MetaEdit+ [194,195], MOF [153], and Xtext [196]. Languages such as Generation of Embedded Real-Time Code based on Aspects (GenERTiCA) [197] and Xtend [198] are used to implement different types of transformations.

As there are plenty of DSLs and meta-modeling techniques used to address various Industry 4.0 challenges, it may imply that such challenges cannot be addressed properly by using established modeling techniques. As for Industry 4.0 process modeling, DSLs are the most popular, followed by the application of knowledge representation techniques, formal methods, and UML [183]. Meta-modeling is mostly applied in manufacturing to achieve uniformity, standardization, and coherent and formal description of processes [189]. We may also notice a strong focus on model transformations, code generations, and the usage of models at runtime.

Accordingly, we believe that applying an appropriate MD approach with DSLs can support flexible, orchestrated, and highly automated production in the domain of Industry 4.0. In the rest of this section, we discuss the following Industry 4.0 aspects in more detail: error handling in production processes (see Section 4.1.2.1), manufacturing documentation (see Section 4.1.2.2), guided production (see Section 4.1.2.3), and process modeling in general (see Section 4.1.2.4). We discuss these aspects as they are present in contemporary production systems and the MD paradigm and DSLs are applied in these aspects. They are also related to the research challenges (Ch1–Ch5) presented in Section 1.

### 4.1.2.1 Error Handling in Production Processes

As industrial systems become increasingly complex and expensive, there is less tolerance for any faults that could cause performance degradation, productivity decrease, or safety hazards [199]. During the process execution, there is a need to detect, classify, and mitigate errors as soon as possible and implement error handling mechanisms to minimize costs caused by errors. An error handler represents a set of actions needed to mitigate the effects of an error and avoid failure scenarios. There are different error handling methods to aid manufacturing systems; however, they are rarely integrated within a production process, as there is a lack of defined workflow, tools, and processes [200].

In contemporary manufacturing companies, known errors and failures that could occur during production are usually specified in textual documents or spreadsheets, such as Failure Mode and Effect Analysis (FMEA). FMEA is a technique used to define, identify, and eliminate known and

potential failures from a system, design, process, or service before they reach a customer [27]. Textual, informal documents cannot be used to automatically derive corrective steps from them and choose specific resources to perform such steps. In flexible production, negative effects caused by an occurred error need to be removed by using formally defined corrective steps, automatically assigned resources to perform the steps, and without stopping production, whenever it is possible to do so. Thus, detecting any disturbance during production requires error handling, which is essential as errors can occur at any step of a production process. Accordingly, they need to be carefully managed and modeled in order to define errors clearly and identify their boundaries [201].

One way to formally specify potential errors and error handlers is to integrate them into production process models created by means of a DSL. By utilizing appropriate code generators, such production process models could be used to automatically initiate corrective steps whenever a known error occurs, leading to better response to production errors and minimizing costs caused by errors.

#### 4.1.2.2 Manufacturing Documentation

Manufacturing documentation is required by various methodologies and standards that a manufacturing company utilizes. Creating and modifying the documentation is usually a time-consuming task that additionally burdens process designers. Especially when there are numerous product and process variations, keeping manufacturing documentation up to date is challenging. Manufacturing documentation needs to be updated whenever a product is changed, certain technical modifications are done in a production system, production volume is adjusted, or production is optimized. These changes are usually stored as separate documents or, in some cases, can be even completely undocumented [202]. The documentation must be kept up to date as any additional change to a product or its maintenance requires accurate documentation. Accordingly, to keep the documentation up to date, additional personnel need to be engaged, raising the costs for a manufacturing company and even for a final product. Also, the creation and update of documentation is usually done in addition to the process planning activities. Therefore, process designers waste time writing manufacturing documentation using information already created in the process planning phase [131].

For instance, Advanced Product Quality Planning (APQP) is a structured method for defining and executing actions required to ensure that a product satisfies a customer – all required steps are completed on time, with high quality and acceptable cost [29]. Documents such as FMEA spreadsheets are required to manage APQP in production, resulting in a high workload of teams in charge of manufacturing documentation writing. FMEA spreadsheets can be large, leading to heavy manual work due to the lack of a proper tool and the complexity of the documentation writing task [131]. Such heavy work is usually not effective enough; thus, FMEA is not updated continuously as changes in production appear. The same statements can also be applied to other document types, such as Bills of Materials (BOMs), Flow Process Charts (FPCs), and user manuals. This issue is additionally emphasized in the context of Industry 4.0, having numerous product and process variations.

The MD paradigm can be applied to keep the documentation up to date and release process designers from such a burdensome manual task. For such a purpose, production process models need to be formally specified and store the information required by the documentation. Storing all the information in a single place can be useful, as different document types share some information, such as names of a production process and its steps or the materials used in production process steps. Therefore, the same information does not need to be written multiple times in documents of different types but stored once in a production process model. As such a model can be formally specified by using a DSL, the model can be automatically transformed into the required documentation, saving the time process designers need to invest and lowering the costs of a manufacturing company.

#### 4.1.2.3 Guided Production

Considering the human workforce in Industry 4.0 manufacturing companies, there are certain challenges that need to be addressed. As different products and their variations are produced in a factory, human workers must adapt and often create new products. Therefore, human workers are reallocated frequently, changing their position, workplace, and products they produce [161]. Additionally, it is possible to lose the know-how in manufacturing companies as increasing automation in factories may lead to the reduction of the human workforce [203], leaving human workers to perform a large variety of tasks. Such a need could be seen during the pandemic of COVID-19 as well, when there were frequent worker replacements and changes of workers' tasks due to their unplanned absences. Thus, these task changes lead to increased production dynamics and require fine coordination of workers in a factory, so their work can be optimized, and production downtime avoided. Accordingly, fast knowledge transfer is required so human workers do not lose time when changing workplaces, as well as to spend less time training the workers when creating new products and for expert workers to spend less time training novice workers coming to a factory.

Human workers need to be employed in such a way that their skills and talents are fully realized in Industry 4.0 [161], thus manufacturing companies need to take special care in reskilling and upskilling the human workforce [204]. Careful management of training and production tasks is necessary for Industry 4.0 manufacturing also due to demographic change [205], as elderly workers are becoming the majority. Although with great experience in manufacturing, they require more adapted tasks and improved skills.

To achieve a fast knowledge transfer on how to produce different products and their variation and provide efficient and low-cost human worker training, production process models can be used to generate resource instructions from them automatically, thus enabling guided production, and also preserve production knowledge permanently. Accordingly, production process models need to be specified formally and MD principles need to be applied, creating an environment for the automatic transformation of production process models into human-readable instructions that include textual descriptions, images, audio, and videos.

The idea of guiding workers step by step through complex activities based on production process models has been already discussed by Gorecky et al. [161]. By using a detection system to recognize the execution of production steps automatically, real-time assistance can be provided to human workers, sending instructions to them one by one in the form of textual descriptions. Whenever necessary, human workers can initiate guided production that can include a user manual, images, audio, or videos to guide them in creating products. Besides receiving instructions through a mobile device, smart watch, or monitor, human workers can also wear Augmented Reality (AR) glasses through which they can get instructions by changing their field of vision, receive remote assistance from expert workers, or receive repair or error handling instructions [206]. Therefore, various technologies can be used in combination with MD principles and formally specified production processes to guide and train human workers whenever necessary.

#### 4.1.2.4 Process Modeling

Research in the context of Industry 4.0 has been gaining much attention in recent years. However, many software specification and development aspects are still not sufficiently covered, such as standardization and modeling languages. In the Industry 4.0 environment, modeling languages aim to contribute to solving the challenges of digital representation and integration. Usually, modeling languages are developed and applied to solve basic challenges in Industry 4.0 but are rarely evaluated through experiments to investigate their benefits in practice [183].

Research related to process modeling, especially in the Industry 4.0 context, has also grown over the years [20]. Production processes need to be digitally supported in Industry 4.0 [21], so they can be integrated within a smart factory. Processes need to be engineered with virtual representations that require abstract thinking and modeling with the support of specialized software [22] – a modeling tool.

Modeling production processes in Industry 4.0 is an essential industrial informatics research topic, as it is crucial to understand and optimize the processes [23]. However, it is not enough to document processes and store them in a factory database, as most contemporary manufacturing companies are doing it nowadays. Production processes need to be modeled to lead production process execution, and, at the same time, not too complex for a human to comprehend. Therefore, production processes need to be formally specified, whose models are machine-readable and thus can be used for dynamic production orchestration and automatic execution. As production process modeling is the main aspect of the state-of-the-art of this thesis, we discuss it in more detail in the following section.

## 4.2 Production Process Modeling

We have investigated the state-of-the-art in the domain of production process modeling and presented the investigation results in [9–11,13,14]. As an outcome of the initial investigation, we have identified requirements for a language that would be used for production process modeling suitable for dynamic orchestration and automatic model execution or generation of executable instructions. Based on the identified requirements, we have created a framework for comparing and analyzing languages and approaches we have found. The framework was presented for the first time in [14].

This section is structured as follows. In Section 4.2.1, we present the identified production process modeling language requirements, while the analysis of languages and approaches we have found are outlined in Section 4.2.2. Discussion and conclusions of the production process modeling state-of-the-art are given in Section 4.2.3.

### 4.2.1 Production Process Modeling Language Requirements

During our preliminary research [8,9], we investigated the production process modeling domain and identified basic concepts for modeling production processes within discrete product manufacturing. The investigation included information gathering from the literature, an industrial use case in which we participated, and domain experts we encountered. Based on the investigation, we have formulated the requirements for a production process modeling language whose models would be suitable for dynamic production orchestration and automatic model execution or generation of executable resource instructions. These requirements, first time presented in [14], do not constitute a complete list of all requirements for production process modeling. They are a core set of characteristics a production process modeling language should have so that process models can be dynamically orchestrated, executed or resource instructions generated from them, and applied in various use cases in the Industry 4.0 context.

The core set of requirements, which need to be fulfilled by a process modeling language whose models are suitable for dynamic orchestration and automatic execution, is defined as follows:

- **Requirement 1 (R1):** the existence of the *process step* modeling concept, representing a single production step needed to execute a production process. It comprises input products, capability, output products, and resource.
  - **Requirement 1.1 (R1.1):** the existence of the *product* modeling concept, which can be an input or an output of process steps. Input products of a process step represent ingredients, such as materials, parts, and intermediate products needed to create output products of a process step, representing intermediate or finished products. The specification of different *constraints* or *properties* of a product needs to be supported, such as dimensions and mass.
  - **Requirement 1.2 (R1.2):** the existence of the *capability* modeling concept, representing a skill needed for process step execution. *Constraints* need to be specified



for a capability, such as a gripper's range required to pick a product, and different *parameters*, such as storage location from which products need to be picked.

- **Requirement 1.3 (R1.3):** the existence of the *resource* modeling concept, representing a human worker, a machine, or a robot able to execute a process step.
- **Requirement 2 (R2):** the existence of the *control flow* modeling concept into which process steps can be organized for execution. Different flow patterns, such as sequence, decision, iteration, and parallelism, need to be supported.
  - **Requirement 2.1 (R2.1):** the existence of the *sequence* modeling concept, representing a flow in which process steps need to be executed one after another.
  - **Requirement 2.2 (R2.2):** the existence of the *decision* modeling concept, representing a flow in which different process steps need to be executed based on a condition.
  - **Requirement 2.3 (R2.3):** the existence of the *iteration* modeling concept, representing a flow in which process steps are iteratively executed until a termination condition is fulfilled.
  - **Requirement 2.4 (R2.4):** the existence of the *parallelism* modeling concept, representing a flow in which process steps are executed in parallel.
- **Requirement 3 (R3):** the existence of the *material flow* modeling concept needed to track materials, parts, and products through a production process. For each input product, it must be known whether it needs to be taken from storage or represents a result of previous process steps. Also, for each output product, it must be known whether it needs to be stored in storage or used in the succeeding steps.
- **Requirement 4 (R4):** the existence of the *message flow* modeling concept that facilitates collaboration between different process steps or resources.

Additional requirements exist that need to be fulfilled by a process modeling language to support a broader array of use cases, especially in Industry 4.0 (e.g., requirements R6 and R8). They are also needed to semantically enrich process models that participate in production orchestration and execution. These language requirements are defined as follows:

- **Requirement 5 (R5):** the existence of the *unordered steps* modeling concept, representing a set of process steps that can be executed in an arbitrary order.
- **Requirement 6 (R6):** the existence of *product and process variations* modeling concepts, representing different sets of process steps that result in multiple products of the same product family or result in the same intermediate or finished product.
- **Requirement 7 (R7):** the existence of the *sub-process* modeling concept, representing a reference to another process, utilized to reduce redundancy and model complexity, and increase reusability.
- **Requirement 8 (R8):** the existence of the *error handler* modeling concept needed to specify errors that could occur during production and process steps needed to handle them.
  - **Requirement 8.1 (R8.1):** the existence of the *local error handler* modeling concept, representing a collection of error-specific steps needed to recover from the error specified at the level of a single process step.
  - **Requirement 8.2 (R8.2):** the existence of the *global error handler* modeling concept representing a collection of process steps that can be used to recover from various errors specified in multiple process steps.

The following two requirements represent overall process modeling language requirements, fundamental in Industry 4.0 to achieve flexible production. These requirements are defined as follows:

- **Requirement 9 (R9):** the models written in the language should be *executable or suitable for automatic instruction generation*, i.e., models should be transformable into a set of executable resource instructions via formally defined transformation rules.
- **Requirement 10 (R10):** the models written in the language should be *independent of production systems*. A process model should be independent of the production system or technology details so that they can be easily specified and used in different production

systems. Also, such *resource-agnostic* process models need to be easily and automatically enriched and transformed into *resource-aware* process models, adding resources to execute process steps, production logistics, and machine configuration activities.

Based on the presented requirements, we have checked whether the languages identified in the literature survey support them or not, as discussed in the following section. We also used these requirements to evaluate whether MultiProLan fulfills them or not, as presented in Section 7.3.

## 4.2.2 Production Process Modeling Languages and Approaches

Before searching for the production process modeling literature, we conducted a pilot study to familiarize ourselves with the domain. We have searched for production process modeling languages using the Google Scholar search engine and gathered several mostly cited and searched papers. By reading the papers, we have formulated search tokens, i.e., the main terms for searching the literature, that we have used to gather literature in a more systematic manner. In Table 4.1, we present the search tokens and the related keywords we used in search engines.

**Table 4.1.** Search tokens and keywords.

Tokens	Keywords
Production Process	Production Process
	Manufacturing Process
Modeling	Modeling
	Modelling
	Model
	Describe
	Description
	Specify
Language	Specification
	Language
	Meta-Model
	Domain-Specific Language
	Domain-Specific Modeling Language
Industry 4.0	Extension
	Industry 4.0
	Fourth Industrial Revolution
	Smart Factory
	Smart Production
	Smart Manufacturing
	Digital Factory
Factory of the Future	
Execution	Execution
	Execute
	Executable
	Instruction
	Interpretable
	Machine-Readable
Model-Driven	Model-Driven

The first two tokens, "Production Process" and "Modeling", have always been used when we searched for literature, as these are the main words of our research related to the production process modeling domain. Other tokens have been optionally combined with the first two tokens to find more specific papers. The token "Language" has been optionally used to narrow the search area and

papers we got from search engines. The token "Industry 4.0" was also optional as we wanted to find production process modeling languages before the Industry 4.0 term began to be used. The token "Execution" has been optionally used as we aimed to find production process modeling languages whose models are execution-ready or instruction-generation-ready. Finally, the token "Model-Driven" has also been optionally used as we wanted to find languages used in MD approaches that usually support model transformations and have formal languages. We did not search abbreviations of the tokens as we assumed that at least once an entire phrase was used in the text. Similarly, we did not search for plural words, assuming that singular words were used at least once in the text.

We searched for the related literature using the search engines of the following digital libraries: Institute of Electrical and Electronics Engineers (IEEE) Xplore, Association for Computing Machinery (ACM) Digital Library, Scopus, Science Direct, Web of Science (WoS), and Google Scholar. To use the advanced search in these libraries, we had to create complex search strings. The search strings may vary a bit depending on the syntax used by the search engines, but they all have keywords written between logical operators. Some search engines have limitations, such as Science Direct, in which only eight logical operators can be used. An example of the search string is presented in the following text:

**("Production Process" OR "Manufacturing Process") AND ("Modeling" OR "Modelling" OR "Model" OR "Describe" OR "Description" OR "Specify" OR "Specification") AND ("Language" OR "Meta-Model" OR "Domain-Specific Language" OR "Domain-Specific Modeling Language" OR "Extension")**

Google Scholar has also been used to search for literature, even though it provides non-peer-reviewed publications as well, such as presentations and websites. However, we have used it to find different standards, technical reports, book sections, and Ph.D. theses in which detailed descriptions of some modeling languages are provided. When we searched for the peer-reviewed literature, we put 2010 as the lower bound of a publication year, while the implicit upper bound was December 2021, when we conducted the research.

We first applied inclusion and exclusion criteria to the studies when we gathered the literature from the aforementioned digital libraries. Inclusion criteria consist of:

- peer-reviewed studies published in journals, conferences, and workshops;
- standards, technical reports, book sections, and Ph.D. theses, that may be non-peer-reviewed studies;
- studies that are accessible electronically; and
- studies that are written in English.

Exclusion criteria consist of:

- non-peer-reviewed studies, such as presentations, informal reports, reviews, editorials, abstracts, keynotes, posters, informal surveys, and websites;
- studies that are not available in English;
- studies that we did not have access to;
- studies that are not related to production process modeling; and
- duplicates.

After applying inclusion and exclusion criteria, we have read the title, keywords, and abstract of the remaining studies and excluded ones that do not fit in the production process modeling domain. Whenever we have not been sure whether to exclude a study based on the title, keywords, and abstract, we read the introduction and conclusion sections of the study to make a final call to include or exclude the study. Due to the possibility that some of the studies have been omitted, we have applied the backward and forward snowballing guidelines [207] to the remained studies. The backward snowballing means that we have used reference lists of the studies to identify the studies they have referenced. The forward snowballing means that we have identified studies that have cited studies we have initially gathered. When we have been looking for referenced and citing studies of the initially gathered ones, we also applied inclusion and exclusion criteria, read the title,

keywords, and abstract, and optionally read the introduction and conclusion sections to conclude whether to keep new studies or not.

The remaining literature includes 51 peer-reviewed studies and 18 non-peer-reviewed studies. Although not fitting the publication year search criteria, an additional peer-review study [24] from the year 2000. has been included since the Bill of Materials (BOM) specification is frequently used in companies. We have used these studies to find answers to the research questions we formulated in this thesis. The research questions are not the hypotheses presented in Section 3.2 but are related to the reviewed literature. These questions are:

- **Research Question 1 (RQ1):** *In which areas process modeling languages have been applied, and which concerns do they address?* This question aims to reveal research trends in process modeling, especially in the context of Industry 4.0.
- **Research Question 2 (RQ2):** *What are the related languages and approaches used for production process modeling?* This question aims to point out relevant languages and approaches used in the production domain.
- **Research Question 3 (RQ3):** *Which languages and approaches are most frequently used and extended for the production domain?* By answering this question, we aim to identify languages and approaches applied by researchers, pointing out current trends in production process modeling.
- **Research Question 4 (RQ4):** *To what degree are identified requirements (c.f. Section 4.2.1) fulfilled by languages and approaches?* This question aims to point out what the concerns and challenges in production process modeling are and what are future research directions.
  - **Research Question 4.1 (RQ4.1):** *How production process models are executed?* This question aims to uncover how production process models are executed or executable instructions are generated from models, which is one of the biggest challenges in production process modeling, especially in the Industry 4.0 context.
  - **Research Question 4.2 (RQ4.2):** *In which way is the production system independence achieved in production process models?* This question aims to reveal one of the biggest challenges in production process modeling – how production system independence can be achieved.
  - **Research Question 4.3 (RQ4.3):** *Is there a language that fulfills all the identified requirements?* This question aims to uncover whether there is a language that fulfills all the requirements and whose models can be used for dynamic production orchestration and automatic execution.

In Table 4.2, we present only the peer-reviewed conference and journal papers alongside a language name, a publishing year, tags that describe the content of a paper, and a name of a conference or a journal in which the paper is published (sorted by the language name and publishing year). To distinguish between different extensions of the same language, the name of the first author of the extension is presented in parentheses. Different tags are added to each paper, having the following meaning:

- General – a language used to model production processes in general;
- MES – a language used to model production processes in the MES domain;
- IoT – a language used to model production processes in the IoT domain;
- CPS – a language used to model production processes in the CPS domain;
- CPPS – a language used to model production processes in the CPPS domain;
- Skills – a language uses skill or capability modeling concepts;
- Services – a language stores service information in models for execution purposes;
- Engines – language models are used as input to engines for execution purposes;
- PPR – a language based on the Product-Process-Resource (PPR) concept;
- VDI/VDE 3682 – a language based on the VDI/VDE 3682 standard;
- MD – a language used in the MD paradigm; and
- Combination – a combination of languages is used to model production processes.

**Table 4.2.** Peer-reviewed papers of the reviewed literature.

Language	Year	Tags	Conference	Journal
ADAPT [62]	2018	General, MD, Skills, Combination	2018 IEEE 16th International Conference on Industrial Informatics (INDIN)	-
ASML [64]	2014	General	-	International Journal of Production Research
BOM/BOO/BOMO [24]	2000	General	-	Concurrent Engineering
BPMN ext. (Zor) [31]	2010	General, Engines	43rd CIRP International Conference on Manufacturing Systems (ICMS 2010)	-
BPMN ext. (Zor) [32]	2011	General	44th CIRP International Conference on Manufacturing Systems (ICMS 2011)	-
BPMN ext. (Michalik) [37]	2013	MES	-	Quality Innovation Prosperity
BPMN ext. (Meyer) [40]	2013	IoT	25th International Conference on Advanced Information Systems Engineering (CAiSE 2013)	-
BPMN ext. (Meyer) [41]	2015	IoT	27th International Conference on Advanced Information Systems Engineering (CAiSE 2015)	-
BPMN4CPS [38]	2016	CPS	2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)	-
BPMN ext. (Bocciarelli) [39]	2017	CPS, MD	2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)	-
BPMN ext. (Polderdijk) [33]	2017	General	International Conference on Business Process Management (BPM 2017)	-
BPMN ext. (Ahn) [34]	2018	General, Engines	Advances in Production Management Systems (APMS 2018)	-
BPMN ext. (Ahn) [35]	2019	General, Engines	-	Sustainability
BPMN ext. (Abouzid) [36]	2019	General	2019 5th International Conference on Optimization and Applications (ICOA)	-
BPMN ext. (Schönig) [42]	2020	IoT, Services	-	Software and Systems Modeling
CT [57]	2012	General	-	Advanced Materials Research
DSL for production workflows [65]	2015	General	2015 IEEE International Conference on Industrial Technology (ICIT)	-
DSML for CPS processes [86]	2015	CPS, Services	-	Journal of Computational Science
GMPM [78]	2014	MES	-	International Journal of Precision Engineering and Manufacturing

GRAMOSA [66]	2015	General	-	The International Journal of Advanced Manufacturing Technology
GSMSP [58]	2018	General, Combination	-	Enterprise Modelling and Information Systems Architectures
Hierarchical DSL for CPPS [90]	2021	CPPS, Services	26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)	-
IAPMM [60]	2015	IoT, Combination	2nd Management and Innovation Technology International Conference (MITiCON2015)	-
I4PMM [61]	2016	IoT, Combination	2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)	-
Information model of DPT [85]	2020	General, VDI/VDE 3682	25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)	-
IPPMA [71]	2020	General	-	International Journal of Production Research
LCDP [92]	2021	General, MD, Services	Advances in Production Management Systems (APMS 2021)	-
MaRCO [69]	2019	General, Skills	-	Journal of Intelligent Manufacturing
MES-ML [74]	2012	MES	-	IEEE Transactions on Industrial Informatics
MES-ML ext. (Weißberger) [75]	2015	MES, MD	2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)	-
MES-ML ext. (Chen) [76]	2018	MES, MD	-	Computers in Industry
MES-ML ext. (Chen) [77]	2021	MES, MD	-	The International Journal of Advanced Manufacturing Technology
MMPD [70]	2020	General	14th CIRP Conference on Intelligent Computation in Manufacturing Engineering (ICME'20)	-
MPIMM/MPIM [67]	2016	General, MD	9th International Conference on Digital Enterprise Technology (DET 2016)	-
MPIMM/MPIM [68]	2018	General, MD	-	The International Journal of Advanced Manufacturing Technology
MService HMS [91]	2016	General, Services	-	Engineering Applications of Artificial Intelligence
Object PN (Latorre-Biel) [50]	2018	General, Services	9th Vienna International Conference on Mathematical Modelling (MATHMOD 2018)	-
PBM for ship block assembly planning [59]	2020	General, PPR, Combination	-	Processes

PMPM [79]	2018	General, PPR	Advances in Production Management Systems (APMS 2018)	-
PMPM [80]	2020	General, PPR, MD	-	International Journal of Precision Engineering and Manufacturing
PN and PN-like models [48]	2019	General, Skills	52nd CIRP Conference on Manufacturing Systems (CMS)	-
PPR DSL [83]	2020	General, PPR, VDI/VDE 3682, CPPS	25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)	-
PPR DSL [84]	2021	General, PPR, VDI/VDE 3682, CPPS	26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)	-
PSL ext. (Qiao) [73]	2011	General	-	The International Journal of Advanced Manufacturing Technology
S-BPM (Fleischmann) [54]	2010	General	International Conference on Subject-Oriented Business Process Management (S-BPM ONE)	-
S-BPM (Fleischmann) [55]	2012	General	-	Universal Access in the Information Society
S-BPM (Neubauer) [56]	2017	General, Services	-	International Journal of Production Research
Skill-based meta-model for assembly processes [87]	2019	CPS, Skills	39th Central America and Panama Convention (CONCAPAN XXXIX)	-
Skill-based meta-model for assembly processes [88]	2020	CPS, Skills	2nd Eurasia Conference on IOT, Communication and Engineering (ECICE)	-
Skill-based meta-model for assembly processes [89]	2021	CPS, Skills	3rd Eurasia Conference on IOT, Communication and Engineering (ECICE)	-
SysML (Fallah) [45]	2016	MES, Services	2016 1st International Workshop on Cyber-Physical Production Systems (CPPS)	-

We analyzed 29 conference papers and 22 journal papers as state-of-the-art in production process modeling. Based on the reviewed literature, conference papers are primarily published in IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) (5 papers), Advances in Production Management Systems (APMS) (3 papers), and the International Academy for Production Engineering, CIRP (French: College International pour la Recherche en Productique) Conference on Manufacturing Systems (CMS) (3 papers). Other conferences have one or two papers presented in this thesis. As for the journals, papers are primarily published in The International Journal of Advanced Manufacturing Technology (4 papers), International Journal of Production Research (3 papers), and International Journal of Precision Engineering and Manufacturing (2 papers). Other journals have one paper each.

As presented in Table 4.2, modeling languages were mostly created to support process modeling in general, CPS/CPSS, IoT, and MES. Various languages utilize the PPR concept, as products, process operations, and resources are the main concepts for production process modeling. Skill-based engineering has also been applied recently, as it allows specifying production processes in a production-system-independent manner. The MD principles have been applied nowadays in order to automatically transform process models into code or models of different types, and a few languages are created for the execution of their models.

There are different languages and approaches to modeling production processes and their various aspects. We divided the languages and approaches into four categories based on which this section is also divided into four subsections:

- **Category 1 (Cat1):** traditional ways to specify production processes (Section 4.2.2.1);
- **Category 2 (Cat2):** process modeling languages that are not primarily created for production process modeling but can be used to accomplish that task, and their extensions made to fulfill specific production process modeling requirements (Section 4.2.2.2);
- **Category 3 (Cat3):** a combination of different modeling languages used to model various aspects of production processes (Section 4.2.2.3); and
- **Category 4 (Cat4):** modeling languages made with the exact purpose to support production process or production system modeling (Section 4.2.2.4).

The distribution of the presented languages and related papers by four categories presented in this section is summarized in Table 4.3. The table includes all the literature we have found and analyzed, including journal and conference papers, technical reports, book sections, and standards.

**Table 4.3.** The distribution of the presented languages and related papers by categories.

Category	Modeling Language	Papers
(Cat1) Traditional ways to specify production processes	BOM	[24]
	BOO	[24]
	BOMO	[24]
	ASME FPC	[25]
	KS A 3002	[26]
	FMEA/PFMEA	[27–29]
(Cat2) Process modeling languages that are not primarily created for the production process modeling and their extensions	BPMN	[30]
	BPMN extensions	[31–42]
	UML AD	[43]
	SysML AD	[44,45]
	PN	[46,47]
	PN and PN-like languages	[48]
	Object PN	[49,50]
	IDEF3	[51,52]
	EPC	[53]
	S-BPM	[54–56]
CT	[57]	

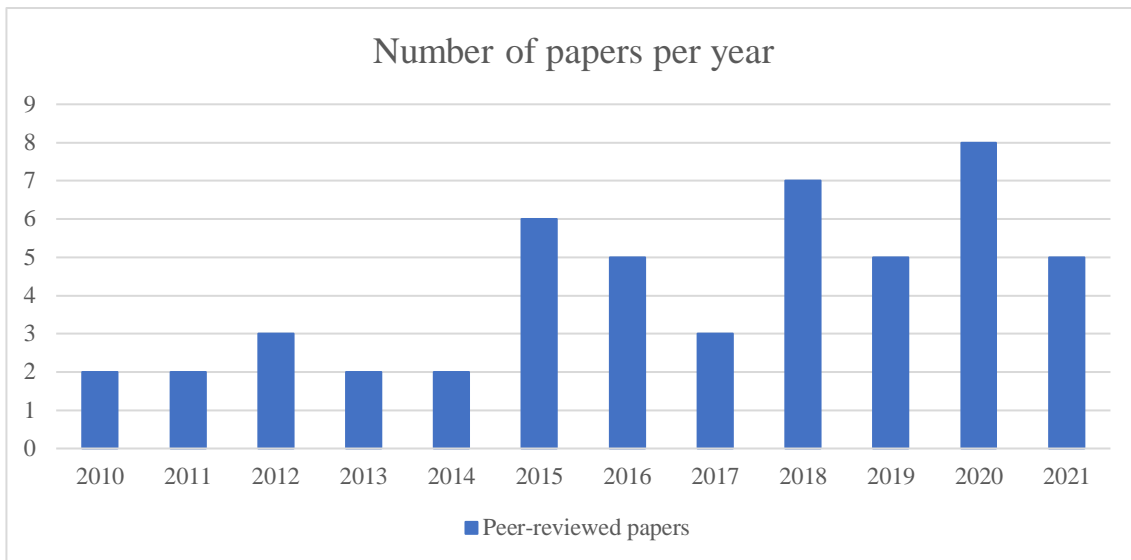


(Cat3) A combination of different modeling languages used to model production processes	GSMSPP	[58]
	PBM for ship block assembly planning	[59]
	IAPMM	[60]
	I4PMM	[61]
	ADAPT	[62]
(Cat4) Modeling languages created to support production process or production system modeling	VSM	[63]
	ASML	[64]
	DSL for production workflows	[65]
	GRAMOSA	[66]
	MPIMM/MPIM	[67,68]
	MaRCO	[69]
	MMPD	[70]
	IPPMA	[71]
	PSL	[72]
	PSL extension	[73]
	MES-ML	[74]
	MES-ML extensions	[75–77]
	GMPM	[78]
	PMPM	[79,80]
	VDI/VDE 3682	[81,82]
	PPR DSL	[83,84]
	Information model of DPT	[85]
DSML for CPS processes	[86]	
Skill-based meta-model for assembly processes	[87–89]	
Hierarchical DSL for CPPS	[90]	
MService HMS	[91]	
LCDP	[92]	

Traditional ways to describe production processes (Cat1), that have been used for years, mostly use spreadsheets to represent various aspects of processes. The category (Cat2) of languages that are not primarily made for production process modeling is the second largest, and Business Process Model and Notation (BPMN), with its extensions, represents the most used language. The third category (Cat3) represents approaches in which a combination of languages is used to describe production processes, with only a few such approaches found. The largest category (Cat4) comprises languages specifically created for the production domain, most of which are DSMLs. All these languages are discussed in detail in Section 4.2.2.1 throughout Section 4.2.2.4.

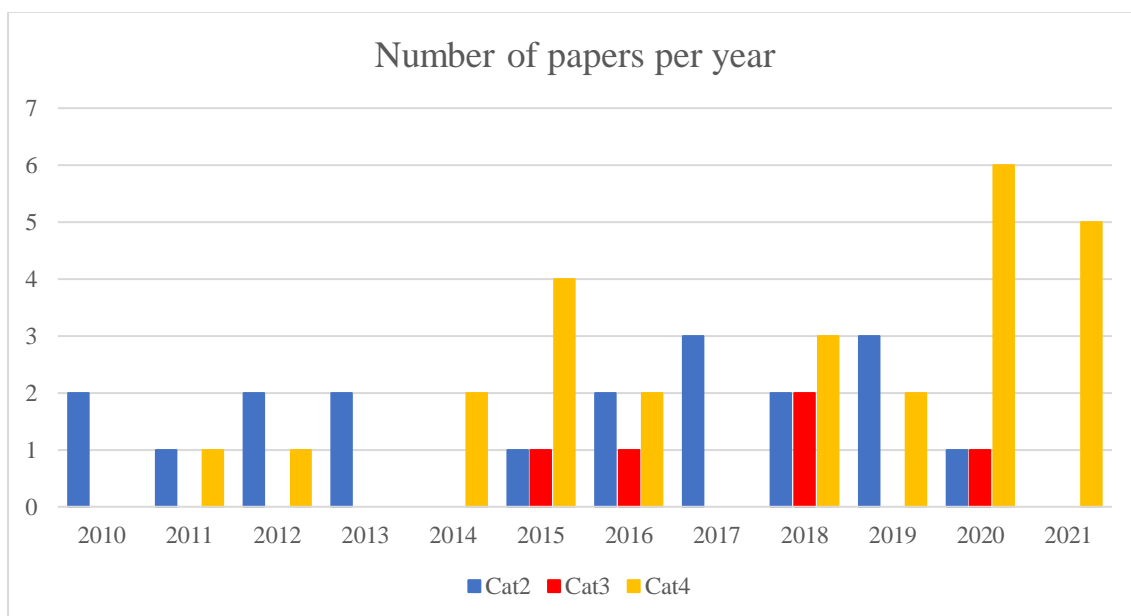
In Graph 4.1, the number of peer-reviewed papers per year, distributed from 2010 to 2021, is presented. Only a single peer-reviewed paper [24] is not presented in Graph 4.1, as it is from 2000 and represents the only outlier that we omitted so as to keep the graph more compact. Based on the papers found, we can see a rising trend of published papers in production process modeling during that period, especially from 2015. Furthermore, as our state-of-the-art investigation was finished in December 2021, there may be additional papers in 2021 that would raise the trend even more. Therefore, it may be stated that production process modeling has been gaining much attention in recent years. Such an increase in research intention may be caused by Industry 4.0 and a need to create flexible production. Also, the rising complexity of production processes and multiple process variations have probably caused increased production process modeling usage.

More information may be obtained if the number of peer-reviewed papers per year of the second (Cat2 – 19 papers), the third (Cat3 – 5 papers), and the fourth (Cat4 – 26 papers) categories are separately presented, as it is in Graph 4.2. The first category (Cat1) mostly includes papers that are not peer-reviewed, such as book sections and standards; thus, it is not presented in Graph 4.2.



**Graph 4.1.** Number of peer-reviewed papers per year.

Languages from the second category (Cat2) have a rising trend until 2019 and most of them are based on BPMN and its extensions. BPMN is extensively used not only in the manufacturing domain but in many other domains as well, as the implementation of a novel DSML from scratch could be time-consuming [208]. However, as the production process modeling domain is complex and the complexity of production process models is rising, researchers are creating novel DSMLs for such a domain. Switching from language extensions to novel DSMLs may be seen in Graph 4.2, as the second category (Cat2) has a falling trend from 2019, and the fourth category (Cat4) has a rising trend from 2019. This rising trend may be caused by a need to execute process models and generate instructions and documentation automatically. Therefore, formal languages with machine-readable models are required. Thus, researchers invest more time in creating novel, formal languages from scratch that will cope with a complex domain of production process modeling. Another way to cope with the complex domain of production process modeling is to combine various languages to model different aspects of production processes. However, such a combined approach (Cat3) is present only in a few papers from 2015. The shortage of combined approaches is probably due to the need for additional knowledge that process designers must have, as using



**Graph 4.2.** Number of peer-reviewed papers per year for the second, third, and fourth categories.

various languages requires process designers to know how to use all of them, making process modeling a more burdensome task.

In the rest of this section, each category, with the representative process languages and approaches, is discussed in its own subsection in detail.

#### 4.2.2.1 Traditional Production Process Specification

In traditional manufacturing, companies use documentation and models of various types, such as schemas, drawings, or textual descriptions, to describe production processes. Production process documents are usually textual, containing a description of how to produce a product. However, such documents do not have a precise and concise syntax that would lower the amount of text needed for the description. Such arbitrarily textual documents are not applicable for automatic instruction generation and execution of the instructions due to their informal description, thus not being machine-readable. As process designers often create production process specifications in spreadsheets, this could be time-consuming, and different errors may occur during the writing. Companies also use manufacturing process charts and Bill of Materials (BOM) to specify production processes. However, even if they are more structured and formal than arbitrary text, none of these specifications provide enough data to facilitate an automatic execution.

A BOM specification represents a structured list of parts [24] used in a product's composition hierarchy. The specification is composed of hierarchy levels with parent items and component items, and for each component item, a quantity is specified. In American Production and Inventory Control Society (APICS) Dictionary [209], BOM is defined as "a listing of all the subassemblies, intermediates, parts, and raw materials that go into a parent assembly showing the quantity of each required to make an assembly." However, BOM specifications are insufficient to understand a production flow [34,35], as they only present a product's composition.

Bill of Operations (BOO) or routing as it is also called, represents a production structure of a product [24]. It provides information on operations and their sequence to produce an item, involved work centers, and standards for setup and execution. BOO may also provide information on tooling, workers' skill levels, inspection operations, and testing requirements [209]. In addition to BOM, the production flow is described through BOO, but products, materials, and their flow during production are missing.

Although BOM and BOO are separate kinds of documents, they are often used in conjunction to describe manufacturing. Thus, Bill of Materials and Operations (BOMO) specifications have been created by integrating product structure and operation information into one document [24]. BOMO covers a sequence of production operations, materials, parts, a final product used in a process, and information on resources or work centers. However, BOMO specifications are still insufficient to specify all the necessary production process details, such as communication between resources.

The American Society of Mechanical Engineers (ASME) Flow Process Charts (FPCs) [25] are used to specify a production flow. Basic activities are defined in FPCs in the form of operation, inspection, transportation, and delay. Such classification of activities is done for analytical purposes and to detect and eliminate inefficiencies in a production process. Storage is also defined, indicating that the material flow may be specified. However, smart resources and communication between them cannot be modeled. Additionally, if FPCs are represented in the form of a table, only sequential processes can be specified. Due to these insufficiencies, FPCs are not suitable for automatic execution of production processes.

There is also the Korean Standards Association's (KS A) 3002 [26] manufacturing process chart standard, which provides a set of graphical notations for manufacturing process modeling. The standard was discussed by Ahn and Chang [34,35], stating that tooling support for modeling and the possibility to execute models automatically are missing from the KS A 3002 standard.

Failure Mode and Effect Analysis (FMEA) is a method for analysis of known and potential failures that could occur in a system, service, or process. Performing FMEA in a company may

minimize damages caused by failures [27]. The FMEA method aims to identify and evaluate potential failures of products or processes and identify actions that may eliminate or reduce the occurrence of potential failures [28]. However, as FMEA focuses only on one component at a time, it is not suitable to model a combination of component failures, and thus one way is to deal with the problem via process modeling and simulation [210].

There are different types of FMEA methods [27], and one of them is Process-related FMEA (PFMEA), which is driven by process functions and product characteristics and is used to analyze manufacturing and assembly processes. The PFMEA method is used to ensure that process-related failures and their associated causes are considered and resolved [29]. A PFMEA document is created after the completion of a production process flow diagram, and the document consists of identified failure modes, causes and effects, and the risks prioritization and their mitigation plans in the process [29]. It specifies all the details about the error handling, but its purpose is not to specify a production process, especially one that will be used for automatic code generation.

Traditional methods are commonly used by process designers who are familiar with these kinds of process specifications. Therefore, due to familiarity with such methods, process designers are usually not eager to replace them with new approaches, even if these kinds of informal specifications could slow them down. As these methods are not formal, e.g., specifications in the form of textual description can be arbitrarily specified, or a spreadsheet's structure can vary, they can be time-consuming and prone to errors during specification writing. In addition, due to the informal specification of production processes and the lack of certain modeling concepts needed for the automatic generation of instructions, these specifications would be hard to transform into executable resource instructions.

The presented traditional methods provided us with knowledge of how production processes are usually specified by process designers. Thus, we plan to use similar concepts when creating a novel DSML for production process modeling, such as ASME FPC basic activity types, to make the language more suitable and appealing for process designers. As DSML is a formal language, its models are machine-readable and thus may be used for the automatic generation of executable resource instructions.

#### 4.2.2.2 Process Modeling Languages and Extensions

As per research presented by Sott et al. [211], Business Process Model and Notation (BPMN), Unified Modeling Language (UML), and Petri Net (PN) are identified as some of the most relevant languages in smart manufacturing. However, by using process modeling languages like BPMN, UML Activity Diagram (AD), and PN, it is difficult to model production processes primarily as these languages are not created for that purpose. These difficulties are even more noticeable whenever the languages need to be used to model all production process concepts required for automatic execution.

Different annotations in the form of arbitrary text are often used in models to add all information required for the production domain. However, a custom parser needs to be implemented to execute such models, as the arbitrary text is not part of a modeling language and needs to be parsed by the language parser. Even if the language parser is created for such a purpose, it is still difficult to read and comprehend the text by humans. To solve this problem, researchers extend existing languages to add missing semantics. Unlike previously mentioned model-level extensions, these are meta-model-level extensions. However, even these extensions are not sufficient due to the wide application domain of a language. Thus, instead of extending existing general-purpose languages, researchers often try to create new domain-specific languages [183], as presented in Section 4.2.2.4.

In this section, we present an overview of BPMN, UML AD, PN, and their extensions, and other languages related to process modeling. In addition, we created a production process model example by using BPMN, UML AD, and PN to present the possibilities and limitations of these three common languages in the context of production process modeling. We created a model

example of wooden plank sawing, presented for the first time in [9], as modeling such a process requires most of the core production process modeling concepts to be used.

The example named *Plank Sawing* is constructed in a way not to include too many modeling concepts but only basic ones from the requirements described in Section 4.2.1, making it possible to test whether these languages can fulfill the requirements. The *Plank Sawing* example represents a process in which two planks need to be sawed and packed. This process is a sub-process of assembling the wooden box, which needs to be done before the assembling activities. In this example, we only consider sawing and packaging of these two planks. We assume that the process will be executed within a smart factory, having smart resources that utilize high-level instructions.

In the example, two planks need to be sawed in parallel. Due to their dimensions, we assume that the same smart resource cannot simultaneously saw both planks. Moreover, both planks need to be inspected after they are sawed. If there is any plank defect, a procedure to discard planks needs to be performed, which should be presented as a sub-process as it is often reused in other processes as well. Otherwise, if there are no plank defects, planks need to be placed in a box. One smart resource needs to hold the planks, while another smart resource gets a box from storage and places it beneath the planks. After that, planks need to be placed in the box, which should be closed afterward, finishing the *Plank Sawing* process. To model production processes suitable for automatic instruction generation, smart resources with their properties need to be specified within the model as well.

This section is divided into BPMN-based, UML-based, and PN-based languages and extensions, other languages without any extension presented, and the summary related to presented languages and extensions. For BPMN, UML AD, and PN languages, the *Plank Sawing* example is presented as well.

**BPMN-based languages.** Business Process Model and Notation (BPMN) [30] is designed by Object Management Group (OMG) with the goal to provide a notation understandable by various business and software users, creating a bridge between business process design and implementation. The language was created to standardize business process modeling, being one of the most frequently used languages for such a purpose. In the context of production process modeling, BPMN covers some modeling concepts, such as error handling and a message flow [9,212], i.e., communication between process steps or resources. However, it is still insufficient to model concepts such as a material flow [9,66,212]. Furthermore, smart resources cannot be fully modeled using BPMN, as the standard BPMN resource definition is abstract without specifying details [39]. They can only be represented by pools and lanes [9,212], but different attributes and constraints cannot be modeled [9]. Also, BPMN does not support the modeling of process variations [86]. Due to the large variety of modeling concepts and BPMN's complexity, non-experts struggle with using such a language [86]. In general, BPMN does not cover all aspects of the manufacturing domain as it is not tailored for such a domain [35]. Accordingly, it does not have domain-specific modeling concepts to express different types of flows [58], and it cannot be used to control production automatically [65]. Still, it is one of the most usable and extendable languages in the production domain.

The *Plank Sawing* production process is modeled using BPMN and presented in Figure 4.1. The BPMN model is created by using Yaoqiang BPMN Editor [213]. There are three lanes representing three smart resources – two robots and a human worker, without the possibility of specifying resource properties in detail. Parallel process steps to pick and saw planks are assigned to robots.

Annotations with arbitrary text can be used to specify input and output products and capabilities for each process step. Also, storage can be specified within annotations whenever a product is retrieved from storage or placed in storage. Annotations that represent products can be connected whenever an input product of one process step is an output product of some previous process step. However, the material flow modeling concept is hard to model when using BPMN, because:

- any annotation can be connected, even ones that represent capabilities;
- connections are not directed; and
- arbitrary text is used to specify products and storage.

In Figure 4.1, annotations are assigned only to parallel process steps to take less space and to keep the model as simple as possible. To automatically generate instructions, there is a need to recognize and parse products and capabilities, especially if constraints and parameters are specified

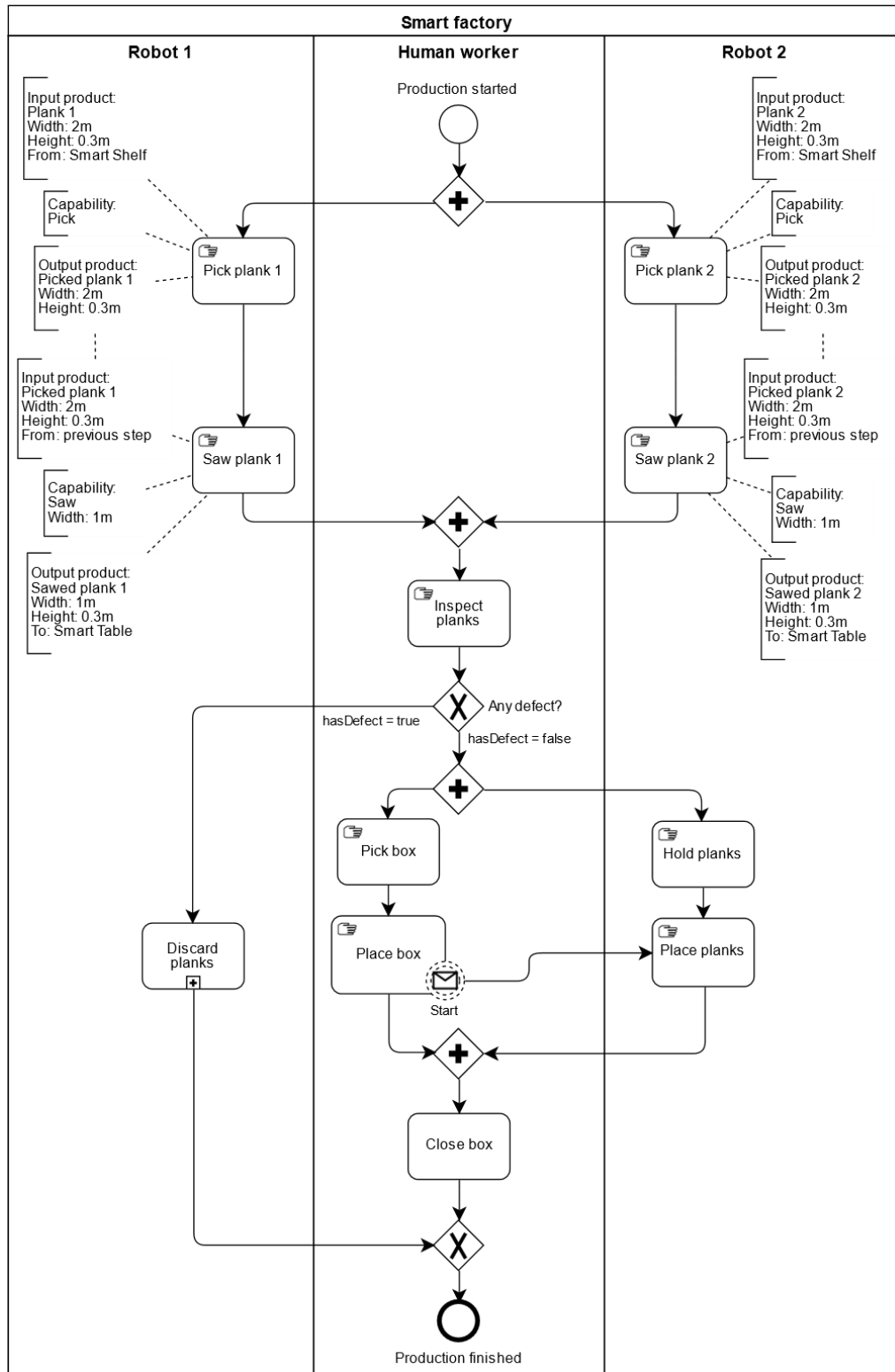


Figure 4.1. The Plank Sawing model example in BPMN.

as arbitrary text in annotations. Using arbitrary text in annotations requires the creation of a parser, as the text is not formally specified as a part of the modeling language.

In the example presented in Figure 4.1, the *Pick plank 1* process step has the input product *Plank 1* that needs to be retrieved from a smart shelf. The same step has the output product *Picked plank 1*, which is the same as the input product as it is only picked from the smart shelf and is not processed in any way. The *Saw plank 1* process step has the input product *Picked plank 1*, which is the same plank from the previous process step, and has the output product *Sawed plank 1*, which is sawed at a smart table.

After the inspection of planks, a decision pattern is modeled, indicating whether planks should be discarded or placed in a box, depending on whether defects are present. Plank discarding is modeled as a sub-process, while plank packaging is modeled with concepts to support collaborative actions. A message flow and a collaboration of smart resources can be modeled using parallelism gates and message events. The message event indicates that planks should not be placed until the box is placed under the planks. Afterward, the box should be closed.

There are different types of process steps presented on the process model diagram, such as operation, transportation, and inspection. However, different process step notations cannot be modeled and distinguished using BPMN.

Based on the presented example, BPMN can be used to specify some of the basic concepts of production process modeling. However, it lacks the semantics of production processes, especially ones whose models are used for dynamic production orchestration and are suitable for the automatic generation of executable instructions. It is hard to model concepts, such as process steps with all the details, the material flow, and process variations. Furthermore, to describe products, storage, constraints, and parameters in BPMN, textual annotations with arbitrary text are used, making it hard to generate instructions as free-form text is susceptible to frequent errors. The error handling modeling concept is covered well, but a BPMN modeling tool should support an additional modeling layer to lower the number of elements on a diagram. BPMN extensions are created to support different aspects of production process modeling. However, BPMN is still business-oriented and is not tailored for a production domain.

For the rest of this subsection, BPMN extensions related to the production domain are discussed.

Zor et al. [31] identified elements that BPMN lacks to represent manufacturing processes, and they suggested adding a set of concepts to BPMN. The authors were inspired by the Value-Stream Mapping (VSM) language, whose detailed description is outlined in Section 4.2.2.4, and mapped VSM material and material flow concepts into BPMN data object and data flow concepts. By mapping VSM modeling concepts into BPMN, the authors aimed to map BPMN modeling concepts into Business Process Execution Language (BPEL) and execute processes with BPEL engines. However, the smart resource modeling concept is not specified, and only the number of workers needed for process execution is presented within tasks.

Zor et al. [32] then presented BPMN extensions to model production processes. They extended the language with manufacturing tasks, machines, tools, parts, and material gateways. However, it is still difficult to model the material flow modeling concept [66]. Additionally, it would be difficult to model human workers with all the required details as they are represented with pools and lanes. As production processes are coupled with machines, process models depend on a specific production system [65]. The tool support for such an extended BPMN is not implemented and is considered by authors as an ongoing work.

As occupational risk factors are often not supported by process modeling languages, Polderdijk et al. [33] extended BPMN with human tasks and physical risk properties. These extensions were created to model and visualize human physical risks, such as lifting and carrying heavy objects or using vibration tools for too long. Within BPMN, the authors integrated Netherlands Organization for Applied Scientific Research' TNO (Dutch: Nederlandse Organisatie voor Toegepast-Natuurwetenschappelijk Onderzoek, TNO) Checklist Physical Load [214], which is focused on

occupational risk factors. With these BPMN extensions, it is possible to fill in a questionnaire dialog within a modeling tool, stating which risk types are present in each human task. These extensions aim to allow users to analyze occupational risk factors that should be mitigated by redesigning production processes. However, the automatic execution of production process models has not been considered in this paper, and thus most of the BPMN insufficiencies in modeling production processes remain. Including occupational risk factors in production process models can be useful, as an intelligent system, such as an orchestrator, can better decide whether to choose a robot or a human worker to perform a particular process step based on the specified risks.

BPMN extensions are also proposed by Ahn and Chang [34] to calculate a similarity measure of manufacturing processes, based on production-related operation similarity. Assuming that a company adopts a Business Process Management (BPM) approach and has many manufacturing process models stored in a repository, these similarity measures extensions enable a search and reuse of the models or their parts when designing a new manufacturing process model. The authors defined a minimal set of BPMN notations as sufficient to model the process examples they presented. Therefore, not all BPMN modeling concepts are used, e.g., the authors limited notations to be used only for sequential and parallel control flow patterns. BPMN is extended with operations, i.e., manufacturing tasks, and components, i.e., materials, parts, and products, but there is no mention of whether smart resources can be added.

The same authors then extended their work and introduced a similarity-based hierarchical clustering method for manufacturing process models [35]. The authors described an agglomerative clustering algorithm for manufacturing process models. If some deficiencies are identified in a process model, the algorithm can help to detect a cluster of similar process models and eventually redesign the whole cluster of process models that may also have such deficiencies. Also, the algorithm can help design new process models by searching a process model cluster of similar products and thus reusing or referring information from the cluster. The authors presented the conversion of process models into a textual form that BPM execution systems may execute, but the model execution is not described in the paper.

Abouzid and Saidi [36] proposed BPMN extensions for modeling manufacturing processes and presented examples of manufacturing process models in carpentry and textile domains. BPMN is extended with concepts for modeling the process step scheduling and inventory. The authors aimed to model the minimum and maximum time needed to complete each process step, making it possible to calculate the time for the whole process. Furthermore, by adding the inventory modeling concept, it is possible to model the inventory of goods and materials, providing insight into semi-finished goods waiting to be used in the following process steps. However, by using these extensions, process models become complex [36], and all the previously described insufficiencies of BPMN in the production domain are also present in this version.

Michalik et al. [37] presented an application of BPMN in modeling a flexible production line at an MES level of information and control systems. The authors presented an example of a BPMN model for the management maintenance function at MES and technological levels. However, the production process within the BPMN model is represented as a single task only. Therefore, it is not clear whether the production process could be executed automatically. Also, all the BPMN insufficiencies in the context of production process modeling are still present. The authors concluded that it is not possible to capture the complex technological details of an MES system and that it is necessary to find new approaches to model these systems, such as a combination of different existing technologies. However, that may increase the models' complexity and require knowledge of different technologies from process designers.

Graja et al. [38] implemented the BPMN4CPS language by extending BPMN to model CPS-aware processes. These BPMN extensions aim to enable process designers to precisely model different CPS concepts. The authors introduced cyber and physical tasks and different attributes for tasks, such as the technology that will be used to send or receive messages. Lanes are extended to represent CPS devices and their constraints and parameters. However, as multiple CPS devices can be modeled, a diagram's size will rise and become hard to read. Therefore, the authors proposed



using colored physical tasks where each color refers to a specific device role written as a comment in the diagram. Physical entities also need to be modeled, which are relevant to the physical activities of CPS devices. As the real-world environment is composed of a number of physical entities, it can be modeled as a pool. However, there are no details on how to execute BPMN4CPS models [42], how to hide implementation details from the process model, and how to model concepts such as the material flow.

Bocciarelli et al. [39] also presented BPMN extensions to address the modeling of resources in the CPS-aware Industry 4.0 business processes. As BPMN lacks modeling concepts to specify IoT devices and CPSs, the BPMN extensions enable the modeling of CPSs as resources associated with BPMN tasks. Also, the extensions cover the performance and reliability-oriented characteristics of resources, making it possible to specify them at different abstraction levels. These extensions are based on the MDA approach, enabling the simulation-based analysis of CPS. However, production process modeling concepts that BPMN lacks to specify are still missing, and thus the automatic orchestration and execution of production processes would be hard to achieve. Also, there is a dependency on a production system by specifying resources and their components, i.e., control units, communication interfaces, sensors, and actuators, within production process models.

Meyer et al. [40] analyzed and identified IoT domain concepts and proposed BPMN extensions to model IoT devices and their native software components, creating IoT-aware process models. The authors chose to extend BPMN as it is more appropriate for IoT extensions in comparison to Event-driven Process Chain (EPC) (discussed further in this section), UML AD, and Web Services Business Process Execution Language (WS-BPEL) [215] process notations [216]. Besides humans participating in business process executions, IoT-aware processes also include IoT devices that can perform some tasks in a smart factory. BPMN is extended with a modeling concept of IoT devices with parameters represented by a swim lane. The IoT devices are added to the process flow as resources for documentation and automation purposes. Also, BPMN is extended with an IoT device's native service modeling concept represented with an activity.

Meyer et al. [41] then proposed further BPMN extensions to model physical entities in the context of IoT as custom participants visualized as collapsed pools in BPMN. Thus, the authors aimed to make extensions close to the standard BPMN notation. A physical entity or a "thing" of IoT represents a passive object that is a part of a physical environment and is of interest to a process as its state can be measured or changed by IoT devices. However, other concepts BPMN lacks are still missing, such as the product and the material flow, and there are no details on how to execute the models [42].

Schönig et al. [42] proposed an approach for integrating IoT objects with business process models ready for execution as, per their claim, many languages and approaches provide no details on how to execute models. They extended BPMN to enable the integration of IoT objects with process models and to preserve the possibility of executing the models in existing BPM execution systems. The models are executed as business processes, i.e., HyperText Transfer Protocol (HTTP) requests are specified in Script and Human tasks, and instructions are sent to machines or human workers. Thus, technological details need to be implemented within the models, making a dependency on a production system. In addition, it would be difficult to model concepts such as material flow, smart resources, storage, products, capabilities, and constraints. Therefore, an orchestrator would not be able to dynamically and automatically orchestrate the production based on the models.

BPMN is a process modeling language, commonly used in various domains, especially for business process modeling. It has good-looking graphical elements and constructs that can be relatively easily learned. Some of these graphical elements, such as gateways, ad-hoc sub-processes, and error events, can be used when creating a novel DSML for production process modeling. Also, BPMN and many of its extensions can be executed in existing BPM execution systems. However, BPMN is a business process modeling language and is not suitable for production process modeling.

**UML-based languages.** Unified Modeling Language (UML) [43] is designed by OMG and represents a family of languages used by system architects, software engineers, and software developers to analyze, design, and implement software-based systems and also model business and similar processes. There are several UML diagram types, but UML Activity Diagram (AD) is used to model a workflow of activities and thus may be used to describe production processes. Vathoopan et al. [7] mentioned they are using UML AD to create a production workflow as a sequence of skills in their prototype tool named SystemPlanner, which is used for skill-based engineering. The action modeling concept of UML AD is used to represent the skills of the production workflow. However, UML AD models are not suitable for automatic execution as they do not consider automation and technical requirements [74]. As UML represents a family of languages, it could be possible to use different UML languages to model production processes and cover some of the insufficiencies of the sole usage of UML AD. However, this would additionally load process designers as they would need to be familiar with multiple languages to model production processes. Thus, the complexity of production process models would grow. According to Lee et al. [78], UML cannot present process flows, detailed activities and resources, and sustainability information at the same time, as there are differences between business and production processes.

The UML AD model of the *Plank Sawing* production process is presented in Figure 4.2. The UML AD model is created by using the diagrams.net [217] modeling tool. Three lanes are used to represent smart resources, but using UML AD, it is not possible to model resource properties.

Parallel process steps are specified to pick and saw two planks. A capability is presented within an activity – a process step, and capability parameters are specified within the activity as input parameters. Capability constraints need to be added as arbitrary text within annotations.

Input and output products are modeled as objects, while their constraints and storage are modeled as text annotations attached to objects. Storage properties are hard to specify within annotations. Like BPMN models, it would be hard to recognize and parse the text from UML AD annotations to generate instructions automatically and execute them. Objects and activities can be connected to specify if a product is a result of a previous process step.

In the example presented in Figure 4.2, the *Plank 1* object represents the input product of the *Pick plank 1* activity, while the output product of the same activity is the *Picked plank 1* object, which is the same as the input product. However, the name of these two objects is not the same since the state of the plank changed – from a state in which the plank is stored on a smart shelf to another state in which the plank is already picked. The *Picked plank 1* object is then used in the *Saw plank 1* activity as an input product, and after it is sawed, the output product of the same activity is the *Sawed plank 1* object with a shorter width than it was before the activity.

Material flow modeling requires the usage of different modeling concepts, such as objects, activities, and annotations. This is more complex compared to BPMN models, in which only annotations are used. Similar to the BPMN *Plank Sawing* example, input and output products are only presented for parallel process steps, while others are hidden from the diagram to make the model as simple as possible.

The decision pattern is modeled after the inspection of planks. A sub-process of discarding planks can be modeled as a call behavior action. Compared to BPMN, it is harder to represent a message flow and a collaboration of smart resources, which may be done by means of synchronization. For example, there is the *Send message* activity in the *Human worker* lane, and in the *Robot 2* lane, it must be checked whether the message arrived using the decision pattern. This solution may be difficult for automatic instruction generation due to a need to continuously check if the message has arrived. The message flow may also be modeled using signals, but their purpose is to communicate with an external participant. Similar to modeling with BPMN, process step notations cannot be modeled.

Like BPMN, UML AD can be used to specify some of the basic concepts, but it lacks the semantics of production processes. Due to object and parameter modeling concepts, UML AD provides more concepts to model process steps compared to BPMN. However, by using BPMN, it

is possible to model unordered process steps and the message flow can be modeled in more detail as the message attributes can be specified, and the message flow can be presented more clearly using relationships. If UML AD is used to model production processes, it would be hard to dynamically orchestrate production and automatically generate executable instructions based on models alone.

UML profiles could be created to extend the language with production process modeling concepts, but too many extensions would need to be created. Using the initial set of modeling concepts and many language extensions would make the production process modeling difficult.

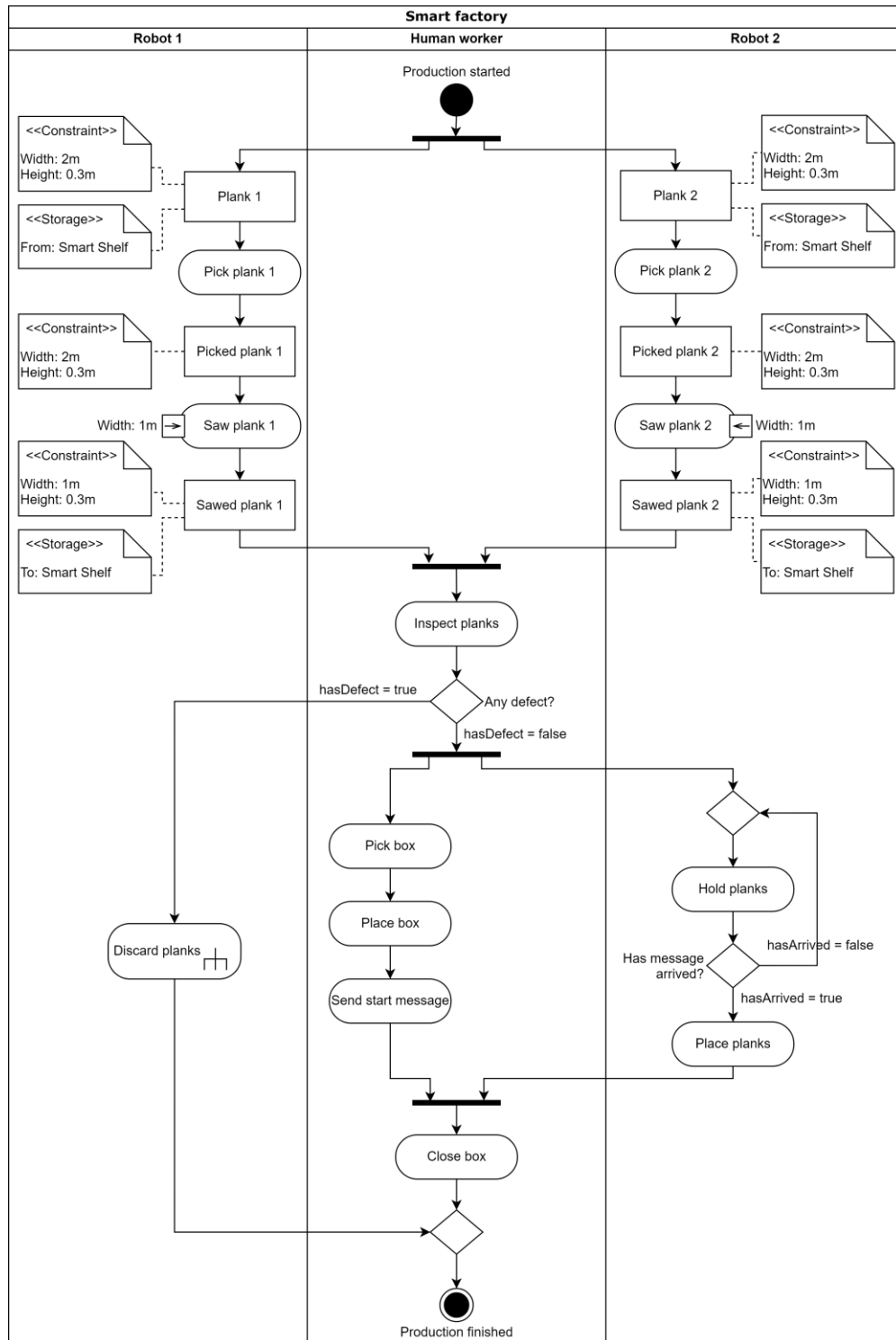


Figure 4.2. The Plank Sawing model example in UML AD.

For the rest of this subsection, we present UML-based languages and their application in the production domain.

Systems Modeling Language (SysML) [44] is designed by OMG and represents a GPML for systems engineering. SysML reuses a subset of UML, making UML a base for SysML modeling and extends it with modeling concepts for system engineering. The language is supposed to be customized to model different domain-specific applications, such as automotive, aerospace, and ISs. It aims to provide simple constructs for modeling a wide range of system engineering problems. Like UML, SysML contains a collection of different types of diagrams, but SysML does not use all UML diagram types. A diagram type that may be considered for production process modeling is the SysML Activity Diagram (AD), which is an extension of UML AD. The scope that SysML AD covers in production process modeling is similar to UML AD. Therefore, all the insufficiencies of UML AD in the context of production process modeling are also present in SysML AD.

Fallah et al. [45] proposed a conceptual framework to model a modular MES using SysML. As there is a need for both flexible and low-cost production, the authors wanted to create a conceptual approach for the model-integrated service-oriented MES to ease reconfiguration efforts. Models would be used not only to design MES but also for the execution of operations. The solution would be built on MDE principles, allowing comprehensive system modeling at different abstraction levels. SysML MES models would represent platform-independent models, which would be connected through the platform-specific Open Platform Communications Unified Architecture (OPC UA) protocol with cell controllers. OPC UA [218,219] is a service-oriented architecture that provides data exchange in a platform-independent manner. It was developed by the OPC Foundation, aiming to create a standard way of communication between different systems and devices through various networks. Fallah et al. aim to model production processes on a higher abstraction level by developing a modular MES through SysML. However, this framework is not implemented. Also, neither a code generator for model transformation into executable code nor an interpreter for direct model execution are implemented. Nevertheless, the idea of creating an MDE solution and transforming PIMs to PSMs in the MES domain may be applied in the production process modeling domain as well, transforming resource-agnostic to resource-aware production process models automatically based on the MD paradigm.

UML represents a family of languages that is utilized in various domains. Different languages from UML may be used to specify various aspects of production processes but would require additional knowledge from process designers when modeling production processes. This may lead to difficulties and errors during modeling. UML AD alone does not provide enough modeling concepts to cover the production process modeling domain fully.

**PN-based languages.** Petri Net (PN) [47] has been designed by Carl Adam Petri [46] to visualize chemical processes. PN is a language with only a few modeling concepts, such as places, transitions, arcs, and tokens. Places represent passive components such as states; transitions represent active components such as activities; arcs are used to connect places and transitions; and tokens represent the current state of a PN system. Due to the limited number of modeling concepts, PN is easy to use and can be applied in many domains. However, PN is too generic to be used for modeling production processes. Fleischmann [54] stated that subjects and objects could only be modeled as comments using PN, which also applies to resources and products. Therefore, it would be hard to model these concepts like that. In addition, the complexity of PN models is high, even for simple production processes containing just a few process steps.

The model of the *Plank Sawing* production process expressed by the concepts of PN is presented in Figure 4.3. The PN model is created using the Platform Independent Petri net Editor 2 (PIPE2) [220–222]. Capabilities of process steps can be specified by adding transitions in a PN model. However, products, smart resources, storage, constraints, and parameters cannot be added to the model. They can only be specified as arbitrary text attached to each transition. It would be hard to parse such a text, and it would also be hard to read for humans. Thus, process designers would have difficulty modeling production processes using the PN language.

Transitions with multiple relationships are used to specify parallelism. In the example presented in Figure 4.3, parallel process steps to pick and saw different planks are modeled between transitions with multiple relationships. After the inspection of planks, the decision pattern is modeled using places with multiple relationships.

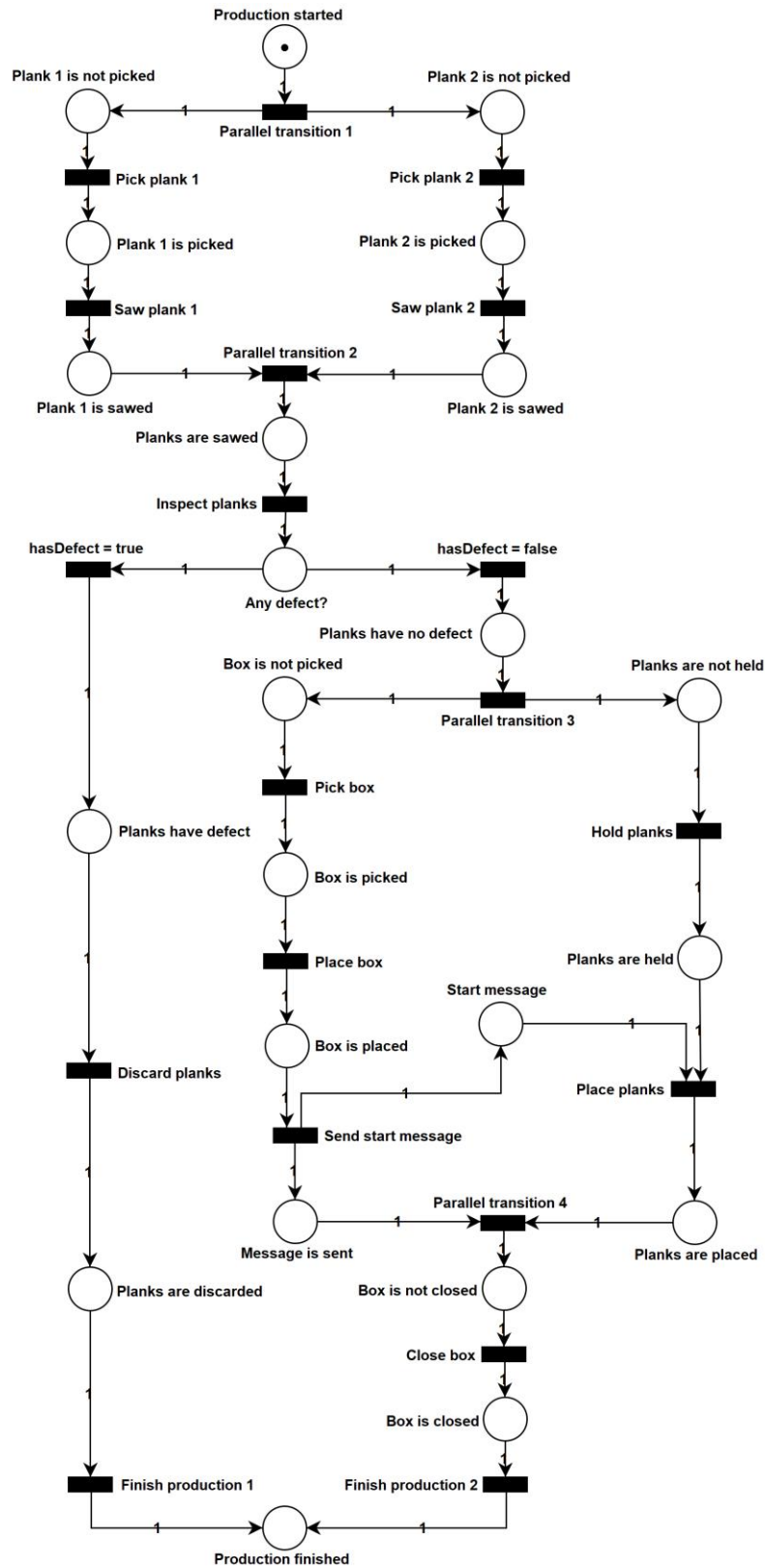


Figure 4.3. The Plank Sawing model example in PN.

As opposed to BPMN and UML AD, sub-processes cannot be modeled using PN, and consequently, they are modeled as transitions, such as the *Discard planks* transition. A message flow can be expressed like other activities in a PN model. In Figure 4.3, the *Send start message* transition is used to represent a process step of sending a message after the box is placed and the *Start message* place is used to represent the message, which is a precondition for the *Place planks* transition. However, no details can be specified for a message, and it cannot be distinguished from other places in a PN model.

Similar to the conclusions of the BPMN and UML AD examples, it can also be concluded that it would be hard to orchestrate production and generate executable instructions based on PN production process models. Likewise, process step notations cannot be modeled by means of PN. The language can be used to model only a few basic concepts of production processes, as it is too generic to model any specific concept of the production process domain. This is especially true if models are specified in detail, as required for dynamic orchestration and automatic instruction generation. Unlike BPMN, an advantage of PN is that it has only a few basic concepts that process designers use to model processes, making the modeling easier. However, the consequence of having only a few modeling concepts is that production process models become too complex, even with a few process steps.

For the rest of this subsection, we present PN-based languages and their application in the production domain.

Müller et al. [48] presented an approach for planning and programming an assembly system based on the PN language. The approach aims to reduce the implementation effort of flexible assembly systems. An assembly process is composed of skills, while resources offer a set of skills. Based on such generic process step descriptions, resources with the required skills can be chosen via services to perform such steps. Specifications of assembly systems and processes are used as work instructions for a control system. An assembly process is divided into two types of flows: (i) a program flow – a logic to perform an assembly process; and (ii) a data flow – a flow of information during the process. A program flow is expressed as a PN model, in which places represent states while performing a process, and transitions represent skills needed to perform process steps. A data flow is expressed with a PN-like model, specifying a connection between different software functionalities or production equipment with other elements. By introducing different levels of workflow, the production system's independence can be achieved. Models at the process planning level are abstract, without any resources. A program flow level contains additional operations and details about resources needed for process execution, while a data flow level specifies the workflow fully. Transitions are described with arbitrary text, and such text needs to be interpreted to be executed. Such a process of interpreting PN model elements may be prone to errors. Also, other insufficiencies of PN to represent production processes still exist. However, utilizing skill-based engineering in production process modeling may provide independence of production process models from a specific production system, which also motivates us to consider applying skill-based engineering when creating a novel DSML for production process modeling.

Latorre-Biel et al. [50] used Object PN to model an Industry 4.0 manufacturing facility so that models can be considered for analysis, performance evaluation, and decision making support. Object PN [49] supports the nets-within-nets paradigm, providing tokens with the structure of PN. These tokens are called token nets or object nets, and they belong to a surrounding net called a system net, forming an object net system. Between different token nets and between token nets and system nets, interaction mechanisms may exist. The authors used a token net to describe a single product, which can request a list of services, and a system net to describe a manufacturing facility with its resources, such as robots and conveyor belts. Each product is modeled with a different token net. Communication between a system net and token nets makes requesting services possible. This approach focuses on a production facility and a product that changes state as it moves through the facility. Therefore, there is a possibility to model the material flow, as storage may be modeled as places called buffers and warehouses. However, it would be difficult to specify different constraints, parameters, and details about products, resources, and storage.

As a general-purpose language, PN has a wide range of application domains. Due to only a few modeling concepts PN supports, it can be easily learned and applied in various use cases. However, it would be hard to model production processes using PN, as it does not have semantics to cover all the required modeling concepts and constructs, and with only a relatively complex production process, PN model diagrams become large and hard to read. The concept of a token, used in a PN model to track the state of a system, can be useful in production process modeling, allowing the tracking of the production process steps execution and monitoring of the whole production.

**Other modeling languages.** The Integration DEFinition (IDEF) method for Process Description Capture (IDEF3) [51] is designed to document and analyze the processes of the existing or proposed systems. IDEF3 was created to describe sequences of activities, and its goal is to provide a method used by domain experts to specify the activities of a system or an organization. IDEF3 was recently used for business process modeling, analysis, reengineering, and management [52] but was also applied in the manufacturing domain [212]. However, IDEF3 lacks modeling concepts to describe materials and resources related to activities [73]. Objects could be used to represent products – from raw materials to packaged goods, and Units of Behavior (UOBs) could be used to represent process steps [212]. Smart resources could be modeled using the object elements [212], but there would be a conflict with the representation of products and materials. IDEF3 process models can be easily understood because of the language's simplicity and the small number of modeling concepts, but it struggles to represent complex processes [78].

Event-driven Process Chain (EPC) [53] is used to model processes with four types of objects: events, functions, rules, and resources. These objects form a chain of functions and events. As EPC was created for business process modeling, it does not provide all the modeling concepts for execution-ready production processes. Like with previously described languages, it would be hard to model resources such as humans, robots, and machines with all the details needed for execution. The material flow concept is neglected [66], and so are the message flow and the error handling. Thus, the automatic execution of the process models would be hard to achieve.

Subject-oriented Business Process Management (S-BPM) is an approach that combines different properties of BPM 2.0, including technical perspectives as well as human interaction [54]. This approach recognizes a subject, i.e., an actor, as the starting point for modeling business processes and focuses on the interaction between the subjects [55]. Thus, communication modeling is the prime aspect of this language. When modeling with S-BPM, abstract subjects are used instead of concrete ones, and concrete subjects are assigned when a process is embedded into an organization, indicating that the independence of the production system may be achieved. However, products and the material flow are still difficult to model, due to the lack of modeling concepts. Therefore, the automatic execution of production process models would be hard to achieve. The assignment of concrete subjects based on abstract ones may be seen as similar to the creation of resource-aware production process models based on resource-agnostic production process models we plan to achieve by utilizing skill-based engineering. However, in the approach we plan to create, the enrichment of resource-agnostic production process models with storage, transportation, and configuration activities should also be provided.

Neubauer et al. [56] specified requirements for the vertical integration of business and technical manufacturing processes so that companies could flexibly react to production changes. The authors investigated the applicability of S-BPM to achieve vertical process integration, and they modified the language by adding different types of messages. They proposed a generic modeling pattern for subject-oriented design and execution of production processes. This pattern comprises three types of subjects: a service, a human, and a coordinator. The service subjects can represent devices, machines, and web services, while the human subjects can represent employees, workers, and other human users in the production environment. The coordinator subjects coordinate the interaction of subjects in the production environment. However, all the insufficiencies of S-BPM in production process modeling still exist. Their vision of subject-oriented modeling, especially the usage of a coordinator in the proposed pattern, is similar to what an orchestrator is supposed to do – to automatically coordinate production.

Many intermediate products and materials exist only for a short time during a manufacturing process and thus do not have any formal name. Wen and Tuffley [57] applied the Composition Tree (CT) language to solve the naming problem that may appear when:

- a product is transformed into a new one;
- different products are combined to form a new one; and
- a product is split into new products, or a part is removed from a product that becomes a product on its own.

CT is a formal language, a part of behavior engineering developed for large software systems. It may also be applied in the domain of manufacturing processes. The root node of CT is used to represent a final product, while other nodes under it are used to represent materials needed for the final product. An ordinal number of a process step to which material belongs is also presented within a node. Therefore, a sequential process may be concluded from a CT model. There are also different attributes for each product and material and relationships to other materials and products, e.g., whether the material needs to be added or mixed with another one. Accordingly, a material flow can be modeled, but the storage modeling concept is not available. Also, only sequential processes may be represented. Thus, behavior trees may be needed to model dynamic aspects of manufacturing processes [57]. The naming issue, discussed by Wen and Tuffley, could be addressed when creating a novel DSML through the material flow modeling concept that connects all materials, intermediate parts, and final products of a production process, thus formally specifying all of them through the process model. A newly created DSML, whose models would be used for production process execution, needs to allow the specification of all materials and intermediate parts in order to track the material flow and enable the automatic execution of production processes.

**Summary.** A common conclusion can be drawn for the presented languages. These languages are not created to support production process modeling, especially the process models suitable for dynamic orchestration and automatic execution. Thus, they cannot be used to specify all the production details. Various authors extended such languages to cover certain aspects of production process modeling but usually not all the details required for process execution. Due to the complexity of the production process modeling domain, many different extensions would need to be used to specify models suitable for execution or automatic instruction generation. Such a language would be hard to use for process designers as it has its base modeling concepts that are not tailored for the manufacturing domain. Even if process designers are trained to model production processes using these languages, none of these languages are conceptually familiar to them. Therefore, they would need to invest a lot of time and effort in learning to use any of these languages and their modeling tools. Difficulties in learning to use a modeling tool that supports one of these languages are not caused by the complexity of the domain but by the complexity of the modeling language and the tool. Accordingly, if the languages are extended to support production process modeling concepts they are missing, these languages are still not tailored for such a domain.

As too many extensions need to be created, it is much more efficient to create a novel language from scratch, adapting it for the production process modeling domain. Therefore, process designers would use modeling concepts they are familiar with, and such a domain-specific language would have formal semantics, with exact rules on how to model production processes, minimizing faults caused during modeling. By using the formally defined language, machine-readable production process models can be created and automatically transformed into executable resource instructions. We expect that the creation of an appropriate novel DSML would speed up the specification of production processes, decrease the number of faults during the specification, and enable faster changes in models.

#### 4.2.2.3 Combining Modeling Languages to Model Production Processes

Using different modeling languages to specify various aspects of production processes is one way to deal with the complexity of production process models and the extensive volume of modeling concepts a single language needs to have to specify production processes. Researchers have created



various approaches in which they used a combination of modeling languages to specify production processes, as discussed in this section.

Bork et al. [58] proposed a procedural framework that guides the design and development of combined conceptual modeling methods in the context of Industrial Business Process Management (IBPM). The framework is composed of three abstraction levels:

- approach level – abstract meta-model building blocks;
- concept level – model and functional building blocks; and
- execution level – execution building blocks.

The authors evaluated the proposed framework using the ADOxx meta-modeling platform to implement a multi-stage manufacturing process simulation environment. ADOxx [172] is a meta-modeling and configuration platform used to conceptualize modeling methods [173]. A formal definition for ADOxx meta-modeling concepts can be found in the work of Fill et al. [174]. As the ideas of IBPM are that there is no single approach applicable to a complex domain and that a combination of different approaches is needed, the authors proposed the usage of different languages in the multi-stage manufacturing process simulation environment. Various models of a production process should be flexible and loosely coupled, representing different views on the production process. The idea is that different modeling languages are based on the same abstract meta-model building blocks. Therefore, the authors implemented abstract and concrete building blocks for Graph-based Simulation of Multi-Stage Production Processes (GSMSP), and they used simplified BPMN and simplified block diagrams of concrete building blocks to model production processes. Based on the presented meta-models, some production process modeling concepts are not covered, such as resources and products. Modeled processes can be simulated, but it would be hard to use the models for dynamic orchestration and automatic execution. In this framework implementation, some flow types are missing, such as material flow, but the framework could be extended with different languages to support various flows. However, using different modeling languages comes with risks in consistency, usability, and intuitive understanding [58]. Therefore, it may be hard for process designers to know how to use different languages, especially as these languages may not be from the domain they are familiar with. These are some of the main reasons we propose the creation of a novel DSML – so that process designers use a single language with modeling concepts familiar to them. The whole domain of production process modeling is complex, and this justifies the authors' statement that one modeling language is not enough to capture all the requirements and that multiple modeling languages should be used. However, our opinion is that the usage of a single language with different levels of detail and different modeling layers can be applied to cope with the domain complexity problem.

Jeong et al. [59] proposed a Process-Based Modeling (PBM) method to describe the production processes of ship block assembly planning. This method uses different modeling approaches, languages, and documents to describe production processes. It consists of four modeling steps:

- creation of a unit model that includes products, processes, and resources for a block;
- design of an integrated network of processes by linking unit models based on a BOM;
- creation of a process-based model describing production processes by combining unit models; and
- generation of a PN model from a process-based model to simulate and analyze the productivity of a ship block assembly process and to support model validation.

The PBM method can be used to model Products, Processes, Resources, Schedules, and Spaces (PPR-SS) needed for block production, and it can identify complex interrelations between processes. The method combines data from different sources, such as a unit product, BOM, activity data, and resource data, creating a unit model. Therefore, a process designer must be familiar with all these sources to specify production process models. Being aware of all these details may be burdensome for process designers. Since the PBM method does not have a formal description [59], it would be hard to execute its models.

Petrasch and Hentschke [60] presented IoT-Aware Process Modeling Method (IAPMM) that uses languages, such as SysML requirement diagrams, UML use cases, and BPMN extensions named IoT-Aware Process Modeling Notation (IAPMN) in order to model IoT-aware processes. The goal of this method is to enable the modeling of software systems and software applications like sensing and actuation. As IoT-aware process models are needed to specify system and software design requirements, a new modeling method is introduced. The authors used existing modeling concepts and added new ones to model IoT devices, physical entities, IoT activities such as actuation and sensing tasks, real-world data objects and stores, mobility aspects, and indicators of whether human beings are involved. Therefore, resources and activities may be modeled, but products and the material flow modeling concepts are missing.

The same authors created Industry 4.0 Process Modeling Language (I4PML) and Industry 4.0 Process Modeling Method (I4PMM) [61] by extending IAPMN and IAPMM with Cloud Computing applications. The language represents an extension of BPMN. By using I4PML, it is not possible to model all the technological details as its purpose is to model production processes in a requirements specification and analysis phase. Also, there are no details on how to execute models of both methods presented by the authors [42]. However, as I4PML represents a UML profile, it has a formal definition and may be integrated with existing validation and simulation tools and used as a source for M2M or M2T transformations within code generators.

As many modeling approaches are insufficient to support collaborative tasks of human workers and machines, Lindorfer et al. [62] presented the Asset-Decision-Action-Property-relationship (ADAPT) modeling approach to model variable work tasks of humans and machines. The approach is based on a newly created meta-model, and the authors presented the first implementation of a BPMN-based workflow designer. Such an approach aims to enable the modeling of assembly workflows, collaboration between human workers and machines, decision-making, and assembly variations. The authors plan to extend the tool to use a combination of Gantt and flow process charts to model various assembly tasks. Currently, they only developed a workflow modeler based on the extensions of the BPMN language to support ADAPT models, enabling the modeling of actions, decisions, and assets. Assets are used to define many concepts, such as skill profiles, resources, products, parameters, images, and work instructions. Therefore, it may confuse users on how to use such a modeling concept. Likewise, a decision concept is used to model both conditional workflows and product variations. A model may also be specified as an Extensible Markup Language (XML) file, later used to generate PSMs for different software applications, such as a visualization or a specific robot system. A code generator is used to transform an XML-based model into an HTML webpage with work instructions and images that are sent to a worker's smart device. Specific robots and workers are not modeled by users, indicating that it is possible to separate some of the production system details from production process models, as they use skill profiles and match process steps with workers. The material flow modeling concept is missing, and it is unclear how a collaboration and message exchange between human workers and machines may be modeled. Also, the authors stated that the Programmable Logic Controller (PLC) program might be generated from models for different kinds of robots but is considered a future work. Similar to the skill profiles used in the ADAPT approach, we plan to utilize skill-based engineering to match process steps with resources but also to separate specific storage, and transportation and configuration activities from resource-agnostic production process models.

Combining different modeling languages to cover various concepts of production process modeling is a justified approach. Due to the complexity of the production process modeling domain, authors used different modeling languages to model various production process aspects, which would hardly be achieved by using only a single language. However, this approach requires knowledge from process designers to apply multiple modeling languages. Moreover, they would need to be trained to actively use different modeling languages, which can be particularly burdensome.

#### 4.2.2.4 Modeling Languages to Support Production Process or Production System Modeling

The final category of modeling languages represents the ones that can be referred to as DSMLs in the domain of production process modeling or production system modeling. Even if some DSMLs are created to support the production system modeling, these languages usually cover modeling concepts such as operations, activities, or capabilities, thus allowing the specification of production processes as well. The DSMLs we investigated were created to cover various aspects of the production domain, as presented in this section.

Value-Stream Mapping (VSM) [63] is a language used to model production processes, having modeling concepts that can express material and information flows in detail. VSM models are used to create a production path from customer to supplier, and they are used to identify issues and reduce waste during production. However, according to Zor et al. [31], VSM models serve documentation purposes only and cannot be executed automatically. Additionally, VSM is limited to simple linear process sequences [66] and describes tasks at a high level [212].

Salmi et al. [64] presented a novel Assembly Sequence Modeling Language (ASML) to support the decision-making process of determining to which level to automate a factory in assembly manufacturing. ASML is used to represent different assembly sequences and to schedule their tasks in different ways to determine the optimal level of automation and estimate the assembly time and cost. The language is built so a user can work on assembly tasks without knowing the technological details. A resource is defined as a block of process tasks, but it is not presented in the paper whether any resource details can be specified. An automation optimization algorithm, a process designer, or a manufacturer will decide the specific resources. Thus, an assembly sequence without resources and their schedule is given as input to the decision-making process. After testing different scheduling scenarios, the output is an assembly sequence with resources, their scheduling, technological details, and cost estimations. However, products, storage, and material flow would be hard to specify, making dynamic and automatic production orchestration difficult. Also, the ASML modeling tool was not implemented and is considered future work. As assembly sequences are modeled without specifying particular resources, and then different scheduling scenarios are tested, this is one way to create resource-agnostic production process models and their scheduling. We also aim to create resource-agnostic production process models, but semantically richer in order to be used by an orchestrator to automatically match resources with process steps and schedule their allocation automatically. Additionally, an orchestrator should also be able to add production logistics steps, including the material flow in a facility, and configuration process steps when needed.

Since manufacturing systems aim to be flexible, reacting quickly to customer changes, Keddis et al. [65] proposed a DSL for production workflows to address such needs. The authors stated that flexibility could be increased by separating process descriptions from production systems. Thus, production process descriptions should not be based on available resources. Instead, a generic description of processes should be defined once and reused with different factory setups. The goal of creating such a language is to use workflows in automated planning and scheduling systems to determine which resources are to execute process steps. This allocation of resources needs to be done automatically without user intervention. The authors implemented a meta-model of the language by using EMF. Therefore, production process models are machine-readable, making it possible to execute such models in the future, but this has not yet been implemented. Process steps do not include resource-related information but only process operations, input and output products, and materials. Besides sequence, parallelism, and decision workflows, the language also supports an arbitrary order of process steps. The error handling modeling concept is considered a future work. Similar to the language and approach presented by Keddis et al., we also plan to create generic process descriptions, by using different detail levels to separate resource-agnostic from resource-aware production process models. However, besides creating resource-agnostic process models as generic process descriptions, we also plan to separate storage, and transportation and configuration process steps from such models.

To overcome the usual lack of the material flow modeling concept, Lütjen and Rippel [66] proposed the GRaphical MOdeling and Simulation-based Analysis (GRAMOSA) integrated approach. GRAMOSA is a novel material flow-oriented production process modeling language that is used to transform factory data models into executable simulation models. The authors implemented an automatic generation of executable simulation models from factory data models to reduce the time and errors of manual transformation. GRAMOSA uses symbols from UML class diagrams but has a profiled UML notation instead of a regular UML one. Processes can be specified in sequence or parallel, but operations can only be specified in a linear sequence. Materials, products, and storage can be specified, thus creating a material flow. However, as the authors stated in the paper, the material flow-oriented approach is complex [66].

Since there is a need for clear and accurate manufacturing process information modeling, Yang et al. [67] proposed a four-layer framework based on meta-modeling, and extended it two years later [68]. The framework is formally defined, providing a systematic and standardized method for manufacturing process information modeling at the process planning stage. The framework is based on UML and has four layers: meta-meta-model, meta-model, model, and data layer. As the framework is constructed via UML, the meta-meta-model is represented by the OMG's MOF modeling infrastructure. The authors presented the meta-model named Manufacturing Process Information MetaModel (MPIMM) to create models independent of specific products and processes with a wide range of applications. Based on the MPIMM and the UML profile extension mechanism, models named Manufacturing Process Information Models (MPIMs) can be created, thus specifying concepts closer to the specific manufacturing domain. With the MDA principles, MPIM can be transformed into specific data models. Process steps can be fully modeled using the framework, and for each step, different resources may be specified that have the same or similar capabilities needed to execute an operation. Therefore, there is no mention of separating production system details from production process models. Also, the framework does not support modeling collaboration between humans and robots [62], and the material flow modeling concept is missing.

To support rapid semi-automatic system design, reconfiguration, and auto-configuration of heterogeneous multi-vendor production systems, Järvenpää et al. [69] presented Manufacturing Resource Capability Ontology (MaRCO). The proposed ontology is formally based on Web Ontology Language (OWL) [223], a semantic markup language for publishing and sharing ontologies on the World Wide Web (WWW). Resource vendors may use MaRCO to describe offered capabilities, while end-users may use it to identify resource candidates for a specific production. The presented ontology is used to formally model the capabilities of manufacturing resources in a vendor-independent manner. MaRCO aims to allow automatic matching and suggests appropriate resources for specific product requirements. By using the ontology, products, process steps to create products, resources, and their set of capabilities can be modeled, as well as matchmaking between process step required functionalities and resource capabilities, i.e., provided functionalities. The authors presented the Device Blueprint modeling concept representing types of devices with nominal capabilities. As nominal capabilities and parameters may change during the usage of resources, there are actual capabilities modeled for individual devices that can refer to a Device Blueprint. By introducing Device Blueprint and nominal capabilities, the independence of production system details may be partially achieved. Process variations may be represented through combined capabilities with the *OR* operator that may be modeled to represent different variations of similar capabilities, but product variations are not described. The authors created a capability catalog, defining a pool of generic capabilities and parameters that are assigned to resources with resource-specific parameter values. We also consider creating the capability repository, as capabilities are not standardized yet. Therefore, process designers could use capabilities and parameters from the repository when specifying production processes. Otherwise, process designers may use synonyms or slang for the names of capabilities and parameters, or they can misspell the name unintentionally. All these scenarios lead to the inability of matching process steps and resources by an orchestrator.

Cramer et al. [70] established the Meta-Model for Production Data (MMPD) for efficient data integration with multi-step production processes, as usually, meta-information about processes is

missing. Creating data-driven models may be used to predict the occurrence of defects in production, which is called Predictive Quality (PQ). MMPD is a product-centric meta-model that can be used to specify production data for automated PQ methods. The language contains modeling concepts, such as products, process steps, machines, tools, and human workers. A process flow determines a sequence of atomic process steps, meaning even changing a machine tool should be represented as a new process step. However, MMPD does not support modeling concepts such as material and message flows. It would be hard to execute MMPD models due to the lack of details, and such models depend on a specific production system, making the modeling difficult for users. We also plan to model production processes consisting of atomic process steps in order to allow the orchestration of each production process step, and not the high-level orchestration. For example, if a gripper needs to be changed or a robot needs to determine its position, these are all considered distinct process steps.

Brunoe et al. [71] proposed an approach and a meta-model for modeling components, processes, equipment, and their relationships. The approach is called an Integrated Product-Process Modeling Approach (IPPMA), aiming to contribute to increasing manufacturing changeability. The product-process modeling approach is based on the modeling of products and processes at two distinct levels:

- generic level – forms a company-specific ontology of components and processes; and
- specific level – based on the generic ontology, specific components and processes in a company are described.

A generic level is introduced, so that specific components and processes are always described using the same attributes and relationships between them. However, production system details are not separated from product-process models. The approach does not include a sequence of operations, implying that it does not support automated routing. It is only possible to assign operations needed to produce a component. Also, the approach was not validated in the industry [71]. Differing from the abstraction levels presented in this approach, we aim to create detail levels to separate equipment, i.e., smart resources, devices, and storage, from production process models, alongside the configuration of the equipment and production logistics.

Qiao et al. [73] presented a manufacturing process information modeling method based on Process Specification Language (PSL) with its extensions related to activities, materials, and resources. PSL is an interchange format used in various manufacturing application systems, such as process modeling, process planning, scheduling, simulation, workflow, project management, and business process re-engineering [72]. The language aims to exchange process information automatically between such manufacturing application systems by transforming their native format into PSL. It was created by National Institute of Standards and Technology (NIST) [224] with the goal to make a standard language with formal semantic definitions to serve as an integration format between different process-related application systems throughout the manufacturing life cycle. The authors [73] extended PSL version 2.5 to create a generic representation of manufacturing process information. The language can represent activities, materials, and resources, with different relationships between such concepts. One activity may be assigned to multiple resource options in a process plan, as different resources may perform such a manufacturing task, but only one will perform it during the process execution. A situation where several resources may be used to execute a single activity can also be modeled. In addition, there is a possibility to create resource sets that may substitute each other, as using one resource leads to using other specific resources. Therefore, these kinds of relationships can be used to create process models that are partially independent of specific resources, but there are still relations to resources from process models. As PSL models are machine-interpretable, they can be exchanged and understood by different application systems. However, certain modeling concepts are missing, such as storage, message flow, and error handling.

Witsch and Vogel-Heuser [74] presented Manufacturing Execution System Modeling Language (MES-ML), whose purpose is to specify MES through different views so that the model complexity can be reduced. MES-ML is based on BPMN and covers the modeling of:

- a technical system, i.e., the technological structure of a plant;

- production processes; and
- MES/IT functions.

Integration of these three views is done with an MES-ML linking model. By using links, it is possible to connect process steps with production system elements, i.e., smart resources, that will execute the steps. However, such links create a dependency between production process models and a production system. Due to this dependency, a process designer must take care of connecting process steps with production system elements during the production process modeling. Such a task makes the production process modeling more complex and could lead to a higher number of created errors during the modeling and higher model complexity. The models could be independent if MES is not implemented for the existing plant but is projected in the early engineering phases – there are process steps that are not yet linked to the technical system. MES-ML is used for specification, standardization, testing, and documentation of the MES software, but the automatic execution of the models would be hard to achieve. In the presented approach, links to connect process steps and resources need to be defined manually. To avoid connecting process steps and resources manually, we aim to utilize skill-based engineering in production process modeling and use an orchestrator to automatically assign process steps to resources and add storage, and transportation and configuration steps.

According to Weißenberger et al. [75], MES-ML does not support the creation of generic production process specifications as the semantics of process tasks are insufficiently specified, and process models are not suitable for code generation. MES-ML models are used as human-readable text-based MES specifications. To enable the modeling of machine-usable MES specifications suitable for code generation, the authors implemented a DSML by extending MES-ML. This new language aims to enable higher independence of production process models from a production system during the process modeling. Instead of the link used to connect a process step with a resource of a production system, the authors proposed a list of links to be used for each process step. At runtime, resources that execute process steps will be determined. However, the dependency between process steps and production system resources remains, and it is ambiguous which resources will execute process steps until the runtime. Therefore, process steps are not entirely independent as they have direct links to the resources of a production system. Such a way to achieve independence is similar to resource options and resource sets presented by Qiao et al. [73] when modeling production processes using PSL. Instead of manually specifying a list of links that connect a process step with resource candidates that may execute the step, we plan to connect process steps and resources through required and offered capabilities. Thus, we plan to provide the independence of a production process model from a specific production system.

As an MDE approach for the food and beverage industry domain has not been established for MES, Chen et al. [76] extended MES-ML further to use it in such a domain. The authors focused on specifying modeling elements of MES functions that would enable the automatic transformation of models into operational MES. The MES-ML language [74] and its previous extensions [75] are further extended with energy consumption and overall equipment effectiveness concepts, as the food and beverage industry creates a product for human consumption, requiring high quality and safety standards. The authors applied their approach in the beer brewing context by adding domain-specific elements to the existing meta-model. However, the authors have not yet implemented a transformation tool from MES specifications to operational MES solutions.

Afterward, Chen et al. [77] applied the approach in two additional use cases in the processing and packaging areas of the food and beverage industry to prove its usability. The authors also presented generated graphical user interfaces from MES-ML models. By using such an MDE approach, MES customizations do not depend on high programming efforts but on domain-specific modeling, which is more suitable for MES users.

Considering manufacturing processes' sustainability or energy consumption, Lee et al. [78] proposed the Green information-based Manufacturing Process Modeling (GMPM) methodology to support the design of MES functions. The authors also presented a procedure for extracting functional requirements from GMPM models and mapping them to standardized MES functions.

The methodology can be used to effectively design and introduce MES by recommending proper MES functions based on information gathered from manufacturing process models. GMPM is designed to include detailed activity information and green information, i.e., manufacturing processes' sustainability or energy consumption information. It can be used to model process flows, activities, and energy consumption simultaneously. However, GMPM cannot be used to model concepts such as material flow, message flow, and error handling.

Lee et al. [79,80] also created the Part-flow based Manufacturing Process Modeling (PMPM) language to express a process flow and a part flow at the same time. This language has similar modeling concepts to the GMPM language created by the same authors. The authors used PMPM in their conceptual framework for the Process Model-based Human-Robot Collaboration (PM-HRC) system. The system is used for human-robot collaboration in a semi-automation process of electric motor production [79]. In such a process, a robot assists a human worker during material handling. As a human worker does not know in advance what part type a robot has picked, there is a delay when the human worker receives the part as it needs to be identified. Therefore, the authors applied the AR technology worn by human workers to identify parts in such collaborative operations. The technology is used to visualize process workflow during the execution and part information modeled with the PMPM language.

In their subsequent paper [80], Lee et al. also used the PM-HRC system so they could respond to changes in customer demands and process plans through a collaboration of human workers and robots. PM-HRC is based on MDE principles and can be used to automatically transform PMPM models into BOMs, assembly instructions for workers, control codes for cobots, and PN verification models. Assembly instructions are sent via mobile devices to human workers. Control codes are sent to cobots and are verified through automatically generated PN models. PMPM is based on the PPR concept. Therefore, operations, products, and resources can be modeled. By using PMPM, tasks performed by automated machines and collaborative works of robots and human workers can be modeled. However, a human-robot collaboration is achieved through modeled process steps and the usage of AR devices during a part handover at an assembly workstation. A message exchange between process steps or resources cannot be modeled. Input and output parts of each task can be modeled, as well as a part flow through process steps, but the storage modeling concept is missing. Additionally, as operations and parameters of control codes are not synchronized when activity or part flows change, manual debugging and manual parameter synchronization are necessary [80]. Compared with the authors' approach, we also plan to use code generators and predefined templates to generate various manufacturing documentation, but also instructions to execute process steps.

VDI/VDE 3682 [81,82] is a standard for formal process descriptions created by the Association of German Engineers and the Association for Electrical, Electronic & Information Technologies, VDI/VDE (German: Verein Deutscher Ingenieure/Verband der Elektrotechnik Elektronik und Informationstechnik) Society for Measurement and Automatic Control (German: VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, GMA). Currently, there are two parts of this standard: (i) Concept and graphic representation [81]; and (ii) Information model [82]. The other three parts are still in preparation: (i) XML representation; (ii) Application in process industry; and (iii) Application in the manufacturing industry. The standard aims to apply formalized process descriptions in technical and non-technical processes, continuous, batch and discrete processes, and in different manufacturing domains. Requirements are specified in this standard, and thus a process description should be:

- simple;
- neutral as regards the branch of industry;
- easily understandable for all the functional areas involved; and
- usable throughout the life cycle of the system.

Regarding the requirements described in the VDI/VDE 3682 standard, the DSML we propose also aims to fulfill those requirements. In addition, it is mentioned in the standard that having a formalized process description creates a basis for:

- checking the quality of processes and products;

- analyzing different metrics;
- simulating, validating, and verifying process models; and
- easily importing/exporting information with computer aided systems.

These are some of the reasons why we also propose a formal description of production processes. When using the VDI/VDE 3682 standard, a process is represented graphically with symbols only, and additional information, such as attribute values, is added afterward. These symbols are visually simple, and there are only a few different symbol types, as one of the aims of the standard's language is to be easily used by users unfamiliar with IT. Production process models are created using states and process operators connected via directed links. States can represent a product, energy, or information, and they are changed by process operators with the help of technical resources, i.e., assets. Connecting products and process operators creates a sequence of process steps. The standard uses separate flows between process operators and states to represent process alternatives or partially shared flows between them to represent parallelism. It does not include additional modeling concepts to represent different control flow types. Therefore, separating and sharing flows could make the process model harder to read, but the idea of not creating additional modeling concepts is justified. Process variations can be modeled using alternative processes, but there is no information on distinguishing decision flows and variations since there is only the alternative process flow concept. Also, by using the standard, alternative products can be specified. As for the material flow modeling concept, there is no mention of storage and locations, but the technical resource modeling concept could probably be used to represent storage, and process operators could be used to represent activities of transporting materials. The standard covers different production process modeling concepts, but machines could hardly process this kind of process representation, especially as PPR constraints are hard to model [83,84].

Meixner et al. [83] proposed a DSL to model and evaluate PPR concepts and constraints between them. The authors analyzed the VDI/VDE 3682 standard used for visual PPR modeling in the basic planning phase. They used the standard and part of its extensions as a base for their language and extended it to create a new textual DSL – PPR DSL. The proposed DSL is used for modeling PPR and constraints between them, whose models should be readable by domain experts and processable by machines. The authors also introduced three requirements for the representation of PPR constraints in CPPS engineering:

- expressiveness of the PPR concepts – the language should support products, processes, and resources; relationships between them to model process alternatives; and constraints between PPR;
- computer processability – the language should be readable by a computer, processable for verification and evaluation, and easy to exchange between platforms; and
- usability and usefulness – the language should be usable by and useful for domain experts. By usability, the authors mean the language is easy to understand, learn and apply, and by usefulness, they mean the language should provide benefits that exceed the effort to use it.

Meixner et al. focused on the first requirement in this paper and left other requirements for future research. The PPR DSL tool implementation is considered a future work as well. Regarding the DSML we propose to create, we also aim to fulfill such requirements by:

- supporting the production process modeling concepts, such as products, process steps, resources, relationships, and constraints;
- creating the abstract syntax – a formal approach to specify production process models that can be processable by computers and machines; and
- creating such a concrete syntax that can be easily used by domain experts and by evaluating the language's usability and usefulness with different user groups.

A year after, Meixner et al. [84] presented a meta-model of the PPR DSL, which is built on the VDI/VDE 3682 standard and its extensions. As VDI/VDE 3682 process models are not machine-readable and insufficiently formally defined, the PPR DSL aims to formalize PPR models. Therefore, the language can specify PPR aspects – products, processes, resources, attributes,



relationships, and constraints. The assembly sequence concept can be used to form a production sequence, but there is no mention of other control flow patterns. PPR aspects can be abstractly specified with high-level attributes so they can be reused and instantiated in a process model. Attributes can be specified at the beginning of the textual model, and afterward, they can be used and reused in PPR concepts. As PPR DSL models are machine-readable, the authors implemented parsers to map the PPR DSL constraint modeling concept into Structured Query Language (SQL) queries. They observed whether constraints are violated by executing generated SQL queries in a relational database. Thus, the authors aimed to evaluate the language by using a technology frequently applied in industry. The presented design of PPR DSL does not cover all the modeling concepts covered by the VDI/VDE 3682 standard. However, the advantage of PPR DSL in comparison to VDI/VDE 3682 is that PPR DSL supports the constraint modeling concept.

A Digital Process Twin (DPT) is a type of digital twin related to manufacturing processes. Such a digital twin aims to optimize a manufacturing process, thus producing optimal products and energy consumption and fulfilling certain quality requirements. Caesar et al. [85] presented an information model of DPT for machining processes. The information model describes the properties and relationships of relevant data needed to perform a processing task. Both planning and processing, i.e., execution data, can be used for process analysis and optimization. The proposed information model is based on the VDI/VDE 3682 standard, which is extended to support machining operations. According to the presented information model, a detailed specification of processes, materials, products, resources, tools, parameters, and constraints can be achieved. The information model has process steps specified in detail, but other modeling concepts are not considered in this approach, such as the control flow, material flow, and message flow.

Most workflow languages lack the expressiveness and flexibility to model CPS processes. Therefore, Seiger et al. [86] introduced a novel DSML – an object-oriented workflow language for formalizing processes in heterogeneous and dynamic environments, enabling a hierarchical composition of processes and process variations. The language can be used to model process steps and service calls at the type level, i.e., a service type should be invoked, not a specific service. Thus, concrete services or devices that will execute process steps do not need to be known during process modeling. Accordingly, the authors aimed to separate a specific production system from a process model, as process models may be used in different non-distributed or distributed environments. Also, resources to execute activities can be specified at both the instance level and the type level, and it may be a device, a service, or a human worker. Specifying a resource type supports the authors' future goal – to enable distributed process execution across many ubiquitous systems with different devices. However, to execute such process models, information about service names, server Internet Protocol (IP) addresses, HTTP methods, and configuration files need to be specified inside process steps. The authors created the language and the modeling tool using EMF and Graphiti [225]. As process models are stored in the XML Metadata Interchange (XMI) format, they can be used for process execution and monitoring. Thus, the authors plan to develop a process repository and a process engine for distributed execution of process models. However, this language does not cover modeling concepts such as material and message flows.

The matching of production resources with production processes needed for creating a product is usually done manually. Brovkina and Riedel [87] proposed a meta-model for CPS, which provides high-granularity skill descriptions that, when combined with the manufacturing process descriptions, allow for automating the matching of process steps with resources. The skill-based CPS meta-model is designed to model skills, primarily for the assembly industry.

Brovkina and Riedel [88] then presented a data model and an automated system for the design of assembly lines based on the Model-Based System Engineering (MBSE) principles. Their solution aims to improve resource selection for an assembly line design, defining which skills are needed to create a product, which resources have such skills, and which transport elements are connecting resources. The presented system uses a graph-based description, specifying assembly sequences with products and capabilities. There are also abstract storage and sources that can be matched with equipment having a storage skill. The presented graph-based model example of assembly sequences seems complex, with many nodes and relationships between them. Also, a node can represent any

of the modeling concepts, and they are all represented as a circle but with different colors, which can be hard to read, especially for someone who has difficulties distinguishing colors.

Afterward, Brovkina and Riedel [89] presented a data model for skill definitions of heterogeneous machines, focusing on standardizing a capability representation. The data model is abstracted to define a generic assembly process model. Usually, assembly process models are specified using flow charts and diagrams or using the same data model applied for skill definitions, allowing automated matching between resources and process steps. However, a common issue is when having different machines that can perform a process step in different ways, leading to the specification of multiple models for the same process. Therefore, an abstract process model is needed in which process steps can be matched with different machine skills. Creating such an abstract assembly process model and matching it with the skills of resources is the primary goal of the presented data model. However, the presented data model does not cover modeling concepts, such as the material and message flows. The authors also defined four requirements for process descriptions:

- abstraction – a process model is an abstract description of required skills;
- machine-independence – a process model is not attached to any specific machine;
- product-derivability – a process model includes product-relevant parameters that can be derived from a product description; and
- standardization – a process model is a formal description of a process or is based on existing standards.

The DSML we propose also aims to fulfill these requirements as:

- a production process model should be composed of process steps that contain abstract capabilities;
- resource-agnostic production process models and their process steps should not contain any specific resources that are to execute such process steps;
- process steps should contain products with their set of constraints, which can be extracted from a product description, but such automatic extraction from a product description is considered as future work; and
- production process models should be formally specified through the defined meta-model of the language.

As DSLs for CPPS are not much focused on production-specific details, and process-specific information from Asset Administration Shell (AAS) has not been integrated with DSL tools so far, Lehnert et al. [90] proposed a new service-oriented hierarchical DSL for CPPS automation software design. The language contains elements and rules for modeling service-oriented process tasks, including information from AAS. This DSL consists of four layers with different abstraction levels and degrees of detail in models. These layers allow different users, such as process and software engineers, to work together during the software development process. The AAS contains process parameters, such as material properties, configurations, and control parameters, and it is used and integrated within the DSL for parametrization purposes. Thus, program code can be generated using a code generator leading to the executable services. The authors stated that sequential process steps could be created, but other control flow patterns are not mentioned. Information about materials and plant properties can be specified, thus making it possible to specify process steps fully. Similar to the presented multi-abstraction level approach, we also plan to utilize multiple modeling layers and levels of detail in order to make the process modeling easier and to include different users when specifying process models. In addition, as the authors integrated AAS with the DSL, similarly, we plan to use a capability repository integrated with a modeling tool associated with the DSML we propose, so that process designers can choose capabilities and appropriate parameters when creating production process models.

Gamboa Quintanilla et al. [91] proposed a framework for customizable product-process specifications based on Manufacturing Services (MServices) in Holonic Manufacturing Systems (HMSs), suitable for product-driven applications. As services were initially built for web

applications and needed a new model to be applied in the manufacturing domain, the authors proposed the usage of a new information model based on the design principles of Service-Oriented Architectures (SOAs). Manufacturing processes are formed by defining relations between the composing MServices. Process steps are represented by MServices that are executed by one or several resources, and they can be reused in multiple processes, as they are independent of a production system. Resources offer a collection of MServices, while process steps request them. Service providers and service requests are matched through MServices, similar to what can be done with skill-based engineering, matching skills requested by process steps and skills offered by resources. However, there is no mention of separating resources from process models in the proposed framework, and resources are equipped with all the information needed to execute operations. The authors did not discuss an invocation of MServices in the framework, but the services could probably be executed like any other.

Indamutsa et al. [92] proposed a Low-Code Development Platform (LCDP) to support the planning, development, and execution of model-management workflows of complex systems. The proposed platform aims to support developers not to manage low-level details, such as discovery, orchestration, and integration of model management services needed to develop a process. LCDP is an event-driven platform based on trigger-action programming that supports high-level abstraction and automation of model management services offered by different providers. By using a DSL, a user can programmatically specify complex expressions of the workflow, and by using a graphical modeling tool, it is possible to model nodes that represent microservices orchestrated when a workflow is executed. A workflow represented in the XML or JavaScript Object Notation (JSON) format can be transformed and executed by an engine. However, in the context of production process modeling, the language does not support modeling concepts such as the material and message flows. Also, LCDP does not have a mechanism to check the terminology used for naming workflow tasks. We plan to cope with such an issue by using a capability repository when modeling production process steps.

Languages created to support the production process or production system modeling cover various aspects of production. They are mainly created to be specific for a particular manufacturing domain, i.e., they can be classified as DSMLs. Nearly all the presented languages allow the specification of production steps, operations or capabilities, and the production flow. However, they usually specify production processes at a high abstraction level. Therefore, the execution details, or how to perform process steps, are usually omitted, and thus models cannot be used for the automatic execution or resource instruction generation. Even if some execution details are included in production process models, they depend on a specific production system.

### 4.2.3 Discussion on Production Process Modeling

In this section, we analyze the literature found and presented in detail in Section 4.2.2 and discuss answers to the research questions formulated in the same section. As part of the answers to the research questions, we also present the investigated languages and approaches and their fulfillment of the requirements presented in Section 4.2.1, as well as the directions for future development of DSMLs for the production process modeling domain.

*RQ1: In which areas process modeling languages have been applied, and which concerns do they address?* Process modeling languages are mostly used in the domains of MES, IoT, CPS/CPSS, and for production process modeling in general, covering various aspects in such domains. The researchers utilized process modeling languages to create models that would be used for simulation, execution, monitoring, comparison and evaluation of process performances, and for documentation purposes.

Various languages utilize the PPR concept, as products, process operations, and resources are essential concepts a production process modeling language needs to have. Recently, researchers are trying to integrate skill-based engineering with production process modeling, as it allows the specification of production processes independently of a specific production system, yet production

process models can be matched with resources of a chosen system when needed through required and offered skills or capabilities. Also, the MD paradigm is applied nowadays in production process modeling, allowing the transformation of production process models into code used for simulation or execution purposes, documentation of various types, or models of different types.

Most applied languages support modeling concepts such as process operations, control flows, and sub-processes but lack support for material flow or collaboration between resources. This lack of support motivated researchers to create novel modeling concepts for existing languages or to create entirely new languages to support these missing concepts. In addition, as flexible production is gaining much attention recently, novel modeling concepts are created to support the modeling of product and process variations. The Health, Safety, and Environment (HSE) management and energy consumption modeling concepts are also researched and applied in a few languages, as human physical risk factors, safety, and energy consumption are particularly considered in Industry 4.0. As for the production process domain coverage, many languages are applied in discrete production, especially in the assembly industry, but languages are also applied in the process industry.

*RQ2: What are the related languages and approaches used for production process modeling?* We have found many languages, approaches, and methods applied in the manufacturing domain, as presented in Table 4.3, and discussed in detail in Section 4.2.2. Researchers have either applied existing languages in their original form, extended them, or created novel languages from scratch to enable production process modeling. Some researchers have applied a combined approach by using different languages to cover various concepts of production process modeling, as such a domain is complex. Therefore, various production process aspects are modeled using different languages. However, this approach requires knowledge from process designers to apply multiple modeling languages, which can be burdensome.

Traditional methods to specify production processes usually lack certain modeling concepts needed for the automatic generation of instructions. Also, such methods are usually not formally defined. They are mostly presented as tables, spreadsheets, arbitrary text, or flow charts with graphical symbols only and without formal semantics, making it hard to transform such specifications into executable instructions.

We also analyzed different process modeling languages that are not primarily created for the manufacturing domain but the business domain or the process modeling in general, and thus they cannot be used to specify all the production details. Researchers have used and extended such languages to cover certain aspects of production process modeling and usually not all the details required for process execution. Due to the complexity of production process modeling, many different extensions would need to be used to specify models that are suitable for execution or automatic instruction generation. Such a language would be hard to use by process designers as it has its base modeling concepts that are not tailored for the manufacturing domain. Even if process designers are trained to model production processes using these languages, none of the language concepts are conceptually familiar to them. Therefore, they would need to invest a lot of time and effort in learning to use any of these languages and their modeling tools. Difficulties in learning to use a modeling tool that supports one of these languages are not caused by the complexity of the domain but by the complexity of the modeling language and the tool. Accordingly, if the languages are extended to support production process modeling concepts they are missing, these languages are still not tailored for such a domain. As too many extensions need to be created, it would be better to create a novel language from scratch, adapting it for the production process modeling domain.

Therefore, to solve this issue, some researchers created domain-specific process modeling languages for the manufacturing domain. However, process modeling languages created for the manufacturing domain usually specify production processes at a high abstraction level. Thus, the execution details are not included, and even if some execution details are included, process models are dependent on a production system. Accordingly, such resource-aware process models cannot

be used in various production systems, and process designers need to put much effort into specifying such process models that are dependent on a production system, covering all the execution details.

*RQ3: Which languages and approaches are most frequently used and extended for the production domain?* According to the reviewed literature, BPMN, UML, PN, MES-ML, and VDI/VDE 3682 are usually extended or used as a base for other languages. BPMN is the most extended language, with numerous extensions and applications in the context of production process modeling. It has many extensions created in the production domain, and if they are used together, many of the presented requirements could be fulfilled. However, too many extensions would need to be applied simultaneously to specify instruction-generation-ready production process models fully. The researchers mostly extended BPMN so that production process models could be integrated with currently used enterprise systems and together be specified with business process details. MES-ML has been developed for years and influenced researchers to extend it multiple times. Recently, the VDI/VDE 3682 standard has gained attention from different researchers and has been extended or used as a base for different languages.

*RQ4: To what degree are identified requirements (c.f. Section 4.2.1) fulfilled by languages and approaches?* A comparative analysis of the reviewed languages and approaches with the specified requirements is presented in Table 4.4. Each cell is filled with both a character and a shading pattern. The character represents whether a language supports the requirement fully – Y (Yes), partially – P (Partially), or does not support it at all – N (No). The shading pattern represents the language's main advantages (horizontal strips) or disadvantages (vertical strips). To distinguish between different extensions of the same language, the name of the first author of the extension is presented in parentheses. At the bottom of the table, a percentage of languages and approaches fulfilling each requirement is presented.

In the following text, we discuss the fulfillment of the requirements presented in Table 4.4:

- **R1 (process step):** this requirement is the only one that is at least partially covered by each of the presented languages. Each language has a modeling concept, such as a process step, an operation, a skill, or a capability. Product and resource modeling concepts are not always covered. Even if products, resources, and capabilities are covered, only some languages (21.15%) cover such concepts' attributes, constraints, and parameters. Usually, attributes are not covered as products, resources, or operations are presented with graphical symbols only, without additional semantics.
- **R2 (control flow):** most languages (92.31%) cover at least a sequence of process steps. Parallelism, decision, or iteration control flow patterns are not supported by all the reviewed languages, as they usually cover only a sequence of steps, because most of the processes are sequential in mass production. However, due to the increased flexibility of processes and a need to change process step execution in runtime, control flow patterns, such as parallelism, decision, and iteration are required and covered by many existing process modeling languages (65.39%).
- **R3 (material flow):** the material flow modeling concept is fully covered only by a few reviewed languages (5.77%). Such a modeling concept is usually not considered at all by languages (69.23%), and some languages are created especially to model material flows, as it is essential to know whether a product needs to be retrieved from storage or is a result of a previous step. As for the partial coverage of such modeling concepts (25.00%), some languages lack a connection between products, or the storage modeling concept is missing.
- **R4 (message flow):** more than half of the reviewed languages (53.85%) do not cover collaboration of process steps or resources. However, in recent years, especially in the context of Industry 4.0, a collaboration between resources, particularly a human-machine collaboration, is covered by languages. Therefore, the collaboration or message flow modeling concept will be needed to fully model and perform production processes in the Industry 4.0 environment. Most reviewed languages that cover such a modeling concept are based on BPMN.

- **R5 (unordered steps):** this requirement does not have partial fulfillment, as a set of unordered steps is a simple modeling concept. Only some reviewed languages support such a modeling concept (25.00%), most of which are based on BPMN. The concept is not crucial for production process modeling but has different applications whenever the order of process step execution is not important. Therefore, it allows workers to discover the optimal order of process steps to execute, if there is one, and thus improve process execution.
- **R6 (product and process variations):** product and process variations are the modeling concepts that are the most unsupported in the reviewed languages (90.39%). These modeling concepts have a special role in the Industry 4.0 context, as production flexibility implies multiple product and process variations. Such variations need to be formally specified as they will be part of dynamic orchestration and production. The VDI/VDE 3682 standard presents product and process variations and will probably influence the further development of languages.
- **R7 (sub-process):** similar to the unordered steps modeling concept, this requirement does not have partial fulfillment due to the simplicity of the modeling concept. Many process modeling languages support the sub-process modeling concept (61.54%) as it is essential in reducing the diagram's complexity, increasing the reusability of processes, and decreasing the redundancy of model elements.
- **R8 (error handler):** more than half of the reviewed languages (57.69%) do not support error handling in production processes. Without such a modeling concept, production process models can be orchestrated and executed, but if an error occurs during production, an execution system does not have formally defined corrective steps to repair the consequences of the error. Consequently, this would increase costs during production as it will be stopped until a production expert arrives to solve the issue. Most of the reviewed languages that cover such a modeling concept are based on BPMN.
- **R9 (executable or suitable for automatic instruction generation):** many languages are created to model production processes for the planning phase or documentation purposes only. Therefore, they are not suitable for execution or automatic instruction generation (69.23%). In recent years, machine-readable models have been created as MD principles are applied and formal languages are developed. Human workers perform the execution of process models by sending them a textual description of each process step. These kinds of instructions are informal. However, a formal specification of process steps is required to send instructions to machines and robots, or specific instructions or services need to be added to process steps. Most languages that support the execution of process models integrate service information into process models.
- **R10 (production system independence):** the independence of production process models from a specific production system is still not fully supported by the reviewed languages. Most languages do not support such independence (71.15%), and some support it partially (28.85%). Production system details are required to create process models suitable for automatic execution or instruction generation. Such required details make it hard for process designers to create process models and make these models not usable in different production systems. Therefore, some languages apply different abstraction levels of modeling concepts, modeling layers, or services on different levels. Researchers usually create independence from production resources by creating sets of resources that can execute process steps, links to specific production systems, or capability-based process models. However, production logistics, transportation steps, specific storage, and steps to configure machines are not part of such production process independence. Languages should also consider these concepts to create an independent production process model.

**Table 4.4.** A comparison of process modeling languages.

Language	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
<b>Traditional ways to specify production processes</b>										
BOM [24]	P	N [34,35]	P	N	N	N	N	N	N	N
BOO [24]	P	P	N	N	N	N	N	N	N	N
BOMO [24]	P	P	P	N	N	N	N	N	N	N
ASME FPC [25]	P	Y	P	N	N	N	N	N	N	N
FMEA/PFMEA [27–29]	P	N	N	N	N	N	N	Y	N	N
<b>Process modeling languages that are not primarily created for the production process modeling and their extensions</b>										
BPMN [30]	P [39]	Y	N	Y	Y	N [86]	Y	Y	N	N
BPMN ext. (Zor) [31]	P	Y	N	Y	Y	N	Y	Y	N	N
BPMN ext. (Zor) [32]	P	Y	N	Y	Y	N	Y	Y	N	N [65]
BPMN ext. (Polderdijk) [33]	P	Y	N	Y	Y	N	Y	Y	N	N
BPMN ext. (Ahn) [34,35]	P	P	N	N	N	N	N	N	P	N
BPMN ext. (Abouzid) [36]	P	Y	N	Y	Y	N	Y	Y	N	N
BPMN ext. (Michalik) [37]	P	Y	N	Y	Y	N	Y	Y	N	N
BPMN4CPS [38]	P	Y	N	Y	Y	N	Y	Y	N [42]	N
BPMN ext. (Bocciarelli) [39]	P	Y	N	Y	Y	N	Y	Y	N	N
BPMN ext. (Meyer) [40,41]	P	Y	N	Y	Y	N	Y	Y	N [42]	N
BPMN ext. (Schönig) [42]	P	Y	N	Y	Y	N	Y	Y	Y	N
UML AD [43]	P	Y	P	P	N	N	Y	Y	N	N
SysML AD [44,45]	P	Y	P	P	N	N	Y	Y	N	N
PN [46,47]	P	Y	N	N	N	N	N	N	N	N
PN and PN-like models [48]	P	Y	N	N	N	N	N	N	P	P
Object PN [49,50]	P	Y	Y	N	N	N	N	N	N	N
IDEF3 [51,52]	P [73]	Y	P	N	N	N	Y	N	N	N
EPC [53]	P	Y	N	N	N	N	N	N	N	N
S-BPM [54–56]	P	Y	N	Y	N	N	Y	Y	N	P
CT [57]	P	P	P	N	N	N	N	N	N	N
<b>A combination of different modeling languages used to model production processes</b>										
GSMSP [58]	P	Y	P	P	N	N	Y	P	P	N
PBM for ship block assembly planning [59]	P	Y	P	N	N	N	Y	N	N	N

Language	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
IAPMM [60]	P	Y	N	Y	Y	N	Y	Y	N [42]	N
I4PML [61]	P	Y	N	Y	Y	N	Y	Y	N [42]	N
ADAPT [62]	P	P	N	P	N	P	Y	N	N	P
<b>Modeling languages created to support production process or production system modeling</b>										
VSM [63]	P	P	Y [31]	Y [31]	N	N	N	N	N [31]	N
ASML [64]	P	Y	N	Y	N	N	Y	N	N	P
DSL for production workflows [65]	P	P	N	N	Y	N	N	N	N	P
GRAMOSA [66]	P	P	Y	N	N	N	N	Y	P	N
MPIMM/MPIM [67,68]	Y	Y	N	N	N	N	Y	N	N	N
MaRCO [69]	Y	P	N	N	N	P	N	N	P	P
MMPD [70]	Y	P	N	N	N	N	N	N	N	N
IPPMA [71]	Y	N	N	N	N	N	N	N	N	N
PSL ext. (Qiao) [73]	Y	Y	P	N	N	N	Y	N	P	P
MES-ML [74]	P	Y	N	Y	N	N	Y	Y	N [75]	P
MES-ML ext. (Weißenberger) [75]	P	Y	N	Y	N	N	Y	Y	P	P
MES-ML ext. (Chen) [76,77]	P	Y	N	Y	N	N	Y	Y	P	P
GMPPM [78]	P	Y	N	N	N	N	N	N	N	N
PMPM [79,80]	Y	Y	P	P	N	N	Y	N	P	N
VDI/VDE 3682 [81,82]	P [83,84]	P	P	N	N	Y	Y	N	N	N
PPR DSL [83,84]	Y	P	P	N	N	N	Y	N	P	P
Information model of DPT [85]	Y	N	N	N	N	N	N	N	N	N
DSML for CPS processes [86]	P	Y	N	N	N	P	Y	N	Y	P
Skill-based meta-model for assembly processes [87–89]	Y	P	N	N	N	P	Y	N	P	P
Hierarchical DSL for CPPS [90]	Y	P	N	P	N	N	Y	N	Y	P
MService HMS [91]	Y	Y	N	N	N	N	Y	N	Y	P
LCDP [92]	P	Y	N	N	N	N	N	P	Y	N
<b>No</b>	<b>0.00 %</b>	<b>7.69 %</b>	<b>69.23 %</b>	<b>53.85 %</b>	<b>75.00 %</b>	<b>90.39 %</b>	<b>38.46 %</b>	<b>57.69 %</b>	<b>69.23 %</b>	<b>71.15 %</b>
<b>Partial</b>	<b>78.85 %</b>	<b>26.92 %</b>	<b>25.00 %</b>	<b>11.54 %</b>	<b>0.00 %</b>	<b>7.69 %</b>	<b>0.00 %</b>	<b>3.85 %</b>	<b>21.15 %</b>	<b>28.85 %</b>
<b>Yes</b>	<b>21.15 %</b>	<b>65.39 %</b>	<b>5.77 %</b>	<b>34.61 %</b>	<b>25.00 %</b>	<b>1.92 %</b>	<b>61.54 %</b>	<b>38.46 %</b>	<b>9.62 %</b>	<b>0.00 %</b>



*RQ4.1: How production process models are executed?* In recent years, many researchers have applied MD principles in their approaches supporting different aspects of production processes. Such languages have been formally defined with a meta-model so that models could be machine-readable and used as input to code generators or model interpreters. The code generators allow to automatically generate human workers' instructions, machines' program code, simulations, and manufacturing documentation.

To execute production process models, researchers usually apply their languages in SOA, adding service information inside process models or making a language domain-specific for the SOA domain. Integrating service information inside production process models, such as HTTP, service name, and port makes it possible to execute such models but also creates a dependency on a specific production system. Also, such service-integrated production process models would be hard to orchestrate, as process steps are coupled with specific services. In addition, process designers would struggle to create production process models with all the service information integrated into production steps.

As for the BPMN extensions, several researchers aimed to use such models in BPM engines and execute them, but production system details or specific services need to be added to process models.

*RQ4.2: In which way is the production system independence achieved in production process models?* Some of the researchers aimed to separate production process models from a specific production system. There are four approaches identified in the reviewed papers:

- a process step has a set of resources that may execute it;
- a process step has a set of links that reference resources from a production system;
- a process step has a resource type, i.e., abstract resource or abstract service needed for execution; and
- a process step has a skill or a capability needed for its execution, and there are resources with a set of skills or capabilities they have. At runtime, a human or an intelligent system is needed to determine which specific resources will execute each process step.

The first and second approaches still store information about production systems, the third approach stores resource types in process models, while the fourth approach makes process models independent of a specific production system. However, for the fourth approach, capabilities may be arbitrarily specified by a process designer, which may be prone to errors, or a capability repository may be created to store the available capabilities of production systems. The latter could make process models partially dependent on the production systems under consideration, as the repository or a dictionary contains currently available capabilities that may be used for production process modeling. This issue could be solved in the future when capabilities and their parameters are standardized.

The researchers also created languages to support the matching and scheduling of production processes, as any of the four mentioned approaches requires matching process steps to specific resources before a process model execution. Also, to create optimal execution in a production system, the scheduling of production process execution and resources needs to be defined.

*RQ4.3: Is there a language that fulfills all the identified requirements?* Based on our survey, we have not identified a modeling language capable of fulfilling all the production process modeling requirements for dynamic production orchestration and automatic execution of production processes, formulated in Section 4.2.1. We have found that many languages lack support for modeling concepts, such as material flow, collaboration and message exchange between resources, product and process variations, and error handling. These modeling concepts need additional attention in the future by researchers if production process languages are to be applied in production, especially in the context of Industry 4.0.

Additionally, human risk factors and safety aspects are most rarely modeled within production processes. As for the human risk factors, they should be specified so that an intelligent system in charge of production orchestration and execution can conclude whether a human worker or a

machine should execute a production task. Also, as human-machine interaction is gaining attention, especially in the Industry 4.0 context, the safety aspect should be considered as well. The energy consumption and time estimations of production process execution also need special attention in the future in order to allow the evaluation of production process execution in various production systems, choosing the one with performances that are preferred.

As the reviewed languages do not cover all identified production process modeling concepts, the specification of production process models with all the execution details would be hard to achieve. Although some examined languages can be used to specify execution details, they still incur dependency between production process models and production system details. Thus, production process models become more complex, burdensome to model by process designers, and hard to read by any interested parties.

Formal languages with machine-readable models should be utilized to support dynamic production orchestration and automatic execution of production processes. However, as such execution-ready or instruction-generation-ready production process models are coupled with a specific production system, the independence of a production system should be achieved. This production system independence does not only consider the independence of specific resources that will execute process steps but also of production logistics and configuration of machines. Therefore, based on the reviewed literature, capability-based process modeling promises the possibility of creating production process models suitable for execution but also independent of a specific production system. Such production process models would be used by matching and scheduling algorithms of an intelligent system, such as an orchestrator, automatically and dynamically creating instructions for the resources of a chosen production system.

Every conclusion discussed in this section is based on the reviewed languages. Such conclusions may provide directions on what should be considered before creating a language whose models are suitable for dynamic orchestration and automatic execution of production processes in the Industry 4.0 environment.

### 4.3 Summary

In this section, we first discussed the research and application of the MD paradigm and DSLs in ISs and Industry 4.0. The largest part of the state-of-the-art investigation is related to the languages and approaches used for production process modeling. We formulated the requirements a production process modeling language needs to fulfill in order to be used in the Industry 4.0 context and analyzed the languages and approaches we found based on these requirements.

According to the state-of-the-art analysis and conclusions made in Section 4.2.3, and as we could not find a modeling language that fulfills the formulated requirements, we decided to create a novel DSML for production process modeling. It should be a formal, capability-based modeling language whose models should be independent of any production system yet suitable to be automatically enriched with details needed for the automatic generation of executable resource instructions. Therefore, the DSML should allow the creation of generic production process models suitable for dynamic production orchestration and automatic generation of resource instructions and manufacturing documentation, unifying different production process aspects. In this way, process designers should not need to take care of production system details during the production process modeling, and they should be entirely focused on modeling process steps. Furthermore, it should be possible to automatically connect process steps with smart resources at runtime without additional load to process designers by using an orchestrator.

The design, development, application, and evaluation of the proposed DSML, named MultiProLan, are discussed in the following sections of the rest of this thesis. The MD solution that enables the transformation of MultiProLan resource-agnostic process models into MultiProLan resource-aware process models and then into executable resource instructions and manufacturing documentation is presented in the following section.

## 5 MD Solution for Modeling and Automatic Execution of Production Processes

To contribute to flexible and automatic production, we propose a novel MD solution for modeling and automatic execution of production processes. Our MD solution was first proposed in [8,9] and extended in [10,11,13,14]. This MD solution comprises two parts: the MD approach and the MD system. The MD approach consists of several steps to convert production process models into manufacturing documentation and executable resource instructions, executed in a digital twin and on a shop floor. To support the proposed MD approach, our MD system consists of several components, such as *Resource Modeling Tool*, *Process Modeling Tool*, *Orchestrator*, *Knowledge Base*, *Instruction Generator*, *Digital Twin*, *Production System*, *Documentation Generators*, and *Documentation Storage*. The research presented in this thesis focuses on Process Modeling Tool and Instruction and Documentation Generators, as well as transformation steps from production process models to executable resource instructions and manufacturing documentation of different types.

A central component of the MD solution is a novel DSML for production process modeling, named *Multi-Level Production Process Modeling Language (MultiProLan)*. The aim of MultiProLan is to help process designers specify production processes in a formal manner, thus creating process models that can lead the execution of production processes and contribute to production flexibility. The language is mainly focused on modeling discrete product manufacturing or, to be more precise, the assembly of discrete products.

This section is structured as follows. The MD system architecture with its components is discussed in Section 5.1, and the steps used in our MD approach are presented in Section 5.2. The main objectives of our MD solution, MultiProLan and Process Modeling Tool are outlined in Section 5.3. The summary of the proposed solution is presented in Section 5.4.

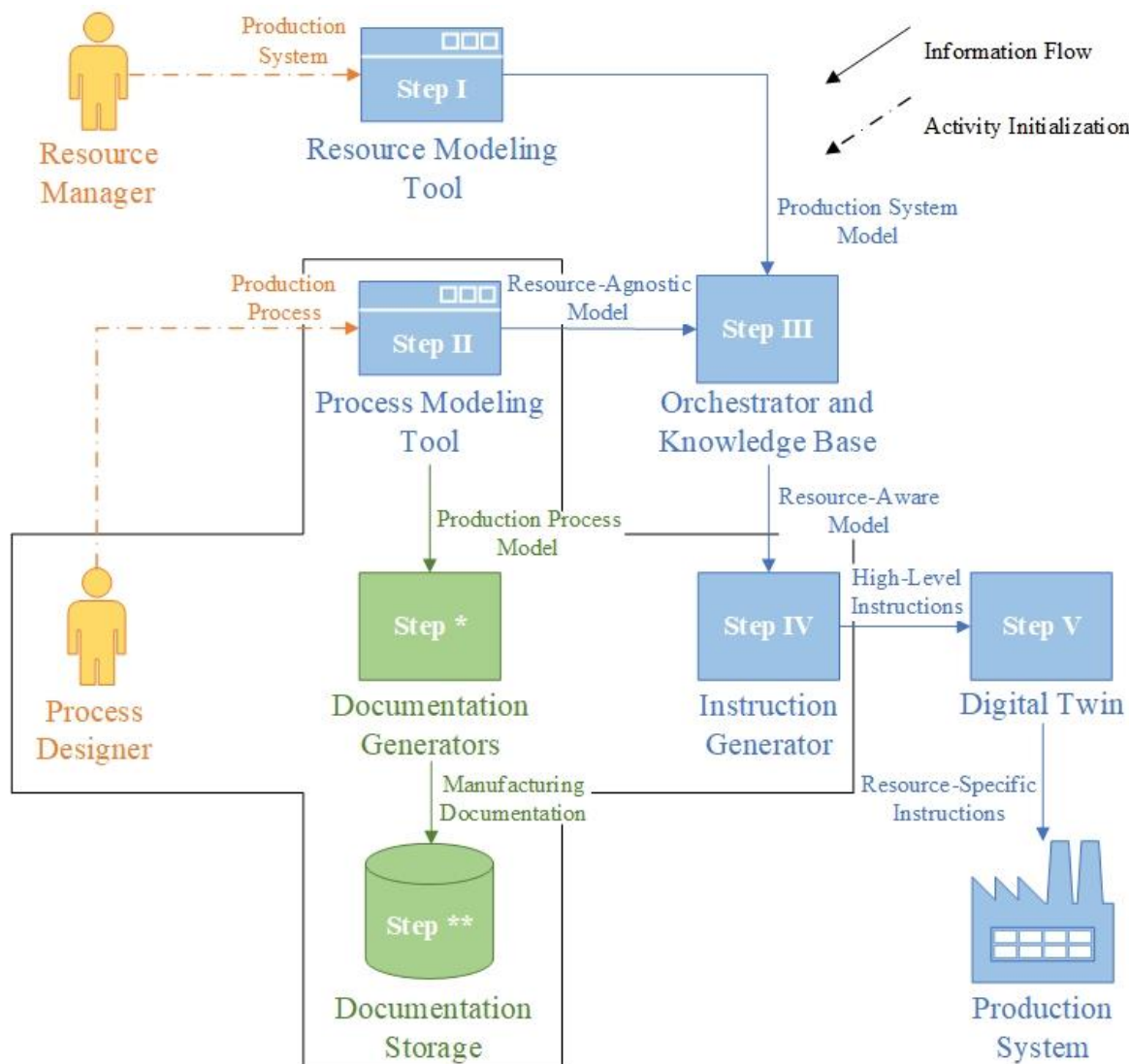
### 5.1 Architecture of the MD System

In Figure 5.1, we present the architecture of the MD system for production process modeling and execution. The architecture components, encapsulated with a solid black line in Figure 5.1, are the main focus of our research, while the components presented outside the black solid enclosure are given to outline the context of our research and they are part of another research. All these components are presented in short in this section, while the main components are discussed further in the following sections. The architecture contains the following components:

- **Resource Modeling Tool.** To create production system models, Resource Modeling Tool is used by resource managers. We denote a resource manager as a person in charge of specifying shop floor resources, their capabilities, production logistics, and available materials and products on the shop floor. Production system models contain all the shop

floor resources and their capabilities, among other details that can be specified. Resource Modeling Tool contains a DSML for production system modeling, and it is a part of another research described in [5].

- Process Modeling Tool.** To specify production process models, including process steps and capabilities needed for their execution, production Process Modeling Tool is used by process designers. Process Modeling Tool contains the MultiProLan language, and as the tool and the language are the main components of our solution, they are discussed further in this thesis. Both Process Modeling Tool and MultiProLan are implemented using a framework for rapid tool prototyping. Frameworks, such as EMF [94,95], are effective in creating tool prototypes in a relatively short time while allowing for the more detailed and customized specification of necessary features. We used the Ecore meta-meta-model to create an abstract syntax of MultiProLan. As process designers are familiar with flow charts and not so familiar with textual notations, we created a graphical syntax for MultiProLan using the Eclipse Sirius framework [100]. In addition to graphical concrete syntax creation, the Sirius framework also enables rapid implementation of a prototype tool [99]. As MultiProLan is a *capability-based* process modeling language, our MD solution uses a *capability repository* for the specification of process steps. We created the capability repository, storing capabilities and their parameters, so they can be reused in production process models. Once the capabilities are standardized by international committees, we will replace our capability taxonomy stored in the repository with a standard one.



**Figure 5.1.** The architecture of the MD solution for production process modeling and execution.

- **Orchestrator.** Orchestrator is utilized to automatically match process steps with resources through required and offered capabilities and to schedule production. Thus, Orchestrator automatically transforms resource-agnostic into resource-aware production process models. The development of Orchestrator is not part of this research, but we used the one presented by Pisarić et al. [4,5], which entirely fits our needs.
- **Knowledge Base.** To store data about production systems in which process models can be executed, Knowledge Base is needed. Orchestrator uses stored knowledge for its matching and scheduling algorithms. Both resource-agnostic and resource-aware production process models are stored in Knowledge Base, as well as the capability repository. We utilized Vaticle TypeDB [226] to store knowledge about production systems, capabilities and their parameters, and production processes, and to enable reasoning about the connections within such a knowledge graph.
- **Instruction Generator.** To automatically transform resource-aware process models into high-level instructions, Instruction Generator is used. We developed Instruction Generator and M2T transformations from scratch in the Java programming language.
- **Digital Twin.** A production system needs to be connected with our MD solution through its Digital Twin. Digital Twin can be used in two different modes: production and simulation-only. In the production mode, Digital Twin is used to obtain high-level instructions sent by Instruction Generator, transform the instructions into machine-specific or human-readable instructions by using *protocol transformation components*, and send such instructions to Production System. In the simulation-only mode, Digital Twin is used as a simulation environment to test created process models when needed, without sending instructions to Production System. We utilized the Robot Operating System (ROS) [227,228] framework to create a bridge across machine-specific instructions that are to be sent to machines. Human workers use a custom-made mobile application to get human-readable instructions on their tablets or smart watches. The Gazebo simulator [229,230] is utilized to create a simulation that is a core part of Digital Twin. Both Digital Twin and the simulation are part of another research and development [4,5]. Therefore, we use them as a black box.
- **Production System.** Production System contains smart resources, both human workers and machines, working together to produce various products. Smart resources are connected with our MD solution through Digital Twin, receiving executable instructions based on production process models.
- **Documentation Generators.** Manufacturing documents of various types are automatically generated from process models by using Documentation Generators. We used the Xtend [198] language to develop Documentation Generators and M2T transformations.
- **Documentation Storage.** The automatically generated manufacturing documentation is permanently stored in a factory Documentation Storage. The generated documentation can be used by any relevant stakeholder.

In the following section, we discuss the usage of our MD approach, including the main steps to get executable instructions and manufacturing documentation from MultiProLan models. These steps are presented in Figure 5.1.

## 5.2 Main Steps of the MD Approach

To create a novel MD approach for production process modeling and automatic generation of executable resource instructions and manufacturing documentation, we applied the MDSEA framework. Production process models independent of a production system – *resource-agnostic* process models, can be seen as TIMs, while production process models enriched with resource information – *resource-aware* process models, can be seen as TSMs. To automatically transform resource-agnostic into resource-aware process models, we need M2M transformation rules. Similarly, to automatically transform resource-aware process models into instructions to execute

process operations and to transform process models into manufacturing documentation, we need M2T transformation rules. In this section, we discuss our MD approach in which production process models are used in such M2M and M2T transformations.

The MD approach comprises the five steps of modeling and automatic execution of production processes and the two additional steps of transforming process models into manufacturing documentation, described in the following subsections.

### 5.2.1 Main Steps of Modeling and Automatic Execution of Production Processes

The developed MD approach comprises the following five steps of modeling production processes and automatically transforming them into executable resource instructions.

**Step I – Specification of production system models.** Resource managers use *Resource Modeling Tool* to create models of production systems. A production system model is stored in *Orchestrator's Knowledge Base* and later used for matching and scheduling algorithms.

Detailed information on a production system needs to be specified first, as Orchestrator requires them when enhancing existing production process models. Such information is required by Orchestrator to automatically transform resource-agnostic into resource-aware production process models. A production system model includes:

- resources of a production system;
- capabilities that are offered by the resources;
- prerequisites needed by a resource to perform a capability;
- interactions between the resources, e.g., whether the resources can cooperate or hand over materials between themselves;
- constraints of various types in the production system;
- different interfaces and protocols used by the resources;
- storage and their locations; and
- materials and products stored in the storage.

The production system details can be added manually to Knowledge Base by specifying them in the form of a Knowledge Base script and executing this script through *Knowledge Base Management System*. However, by using Resource Modeling Tool and its DSML, a resource manager can create a graphical production system model, transform the model into a Knowledge Base script, and execute the script on Knowledge Base Management System, saving the production system details in Knowledge Base.

**Step II – Specification of resource-agnostic production process models.** Process designers use *Process Modeling Tool* that utilizes MultiProLan to create *resource-agnostic* production process models, which we denote as *Master-Level (MasL)* models. These process models may be applied in various production systems with different production resources. As resource-agnostic models are independent of any production system, Step I and Step II of our MD approach may be performed in parallel.

A MasL production process model represents a technical description of a production process. It includes specification of process steps without details required for the automatic production, such as: smart resources required to execute process steps; production logistic activities; specific storage where products and parts are stored; and machine configuration activities. Therefore, MasL models do not depend on a specific technological platform, i.e., on a production system in which modeled production processes are to be executed.

We have implemented a graphical Process Modeling Tool to allow the modeling of production processes by using MultiProLan. Process designers use the modeling tool to model production processes without the need to specify execution details. These MasL models include:

- process steps;
- capabilities required to execute process steps, with their parameters and constraints;
- input and output products, i.e., raw materials, components, or finished goods, with constraints and the material flow;
- the control flow, i.e., sequence, parallelism, selection, and iteration patterns;
- unordered process steps;
- collaboration needed between process steps;
- product and process variations; and
- sub-processes.

In addition to the above, MultiProLan contains modeling concepts aimed at modeling errors and corrective process steps used to handle these errors. By the notion of a production error, we denote errors that may occur during the execution of a process. This part of a MasL model is optional, and if it is specified, it can be visible or hidden within Process Modeling Tool on demand.

By using the services of Knowledge Base Management System, MasL models can be stored in Knowledge Base, which represents a central place to store process and resource semantics. These stored MasL models can be reused at any time, as they can be imported into Process Modeling Tool through Knowledge Base Management System, thus enabling any required MasL model modifications and optimizations.

**Step III – Enrichment of resource-agnostic production process models.** *Orchestrator* can automatically enrich *resource-agnostic* process models with details needed for the process execution. It uses knowledge about a production system in which processes are to be executed and enriches resource-agnostic process models, creating *resource-aware* production process models, which we denote as *Detail-Level (DetL)* models.

A production process model created by a process designer needs to be executed within a chosen production system. Additional information must be placed in a MasL model to be used for the automatic generation of executable instructions. A MasL model needs to be enriched with the following elements of a chosen production system:

- specific resources, such as robots, machines, and human workers, that are to perform process steps;
- production logistic activities, which represent transportation of materials, products, and resources;
- specific storage where materials, parts, and products are stored; and
- configurations of machines and robots, such as software setup, changing grippers, position calibration, and plugging into a charger or a workstation.

DetL models can be created either manually or automatically. A process designer conducts the manual DetL creation by using MasL elements and DetL elements related to a chosen production system. DetL models are similarly stored in Knowledge Base as it is with MasL models. A process designer can make additional changes to the existing MasL/DetL model, which is imported from Knowledge Base, or create a DetL model from scratch using Process Modeling Tool. A process designer can also import the existing DetL model to check the model, improve, refine, and optimize it before execution, or monitor its execution. Both import and export of MasL/DetL models are done via Knowledge Base Management System.

Although manual creation of DetL models is possible, it would be a complex and time-consuming task as it requires specification of all the technological and production system details needed to execute the process. During the DetL modeling, a process designer must think about the production system details – specific resources, storage, production logistics and configuration steps already specified in Knowledge Base. Therefore, the full potential of our system is reached if *Orchestrator* is used to automate this burdensome process.

In our vision of Industry 4.0 production process modeling, production system and production process models should be separated to enable a high level of product customization. Thus, the

automatic creation of DetL models from the existing MasL models is also supported in our MD solution and is conducted by means of the Orchestrator software. Our Orchestrator [4,5] comprises a matching mechanism that connects resources with process steps and storage with products, as well as a scheduling algorithm to provide an optimal match between resources and process steps. Therefore, matching and scheduling algorithms of Orchestrator can be seen as M2M transformations, as MasL models independent of a production system are transformed automatically into DetL models dependent on a specific, chosen production system. Orchestrator is also responsible for adding transportation and configuration process steps to existing process models. Based on a MasL model and the production system details gathered from Knowledge Base with the help of Knowledge Base Management System, our Orchestrator can automatically generate a DetL model. Then, the generated DetL model is sent to Knowledge Base Management System to store it in Knowledge Base. In the following text, the process of automatic creation of DetL models is explained.

Knowledge Base needs to provide all the necessary information about a given production system for Orchestrator to be able to generate DetL models from MasL models automatically. Every process step specified in a MasL model contains a capability required to execute a process step, alongside various constraints that need to be fulfilled. It is necessary to add information about a resource that will execute a process step within the chosen production system. This resource cannot be just any resource but the resource with the required capability in its set of offered capabilities and that can fulfill defined constraints. By using Knowledge Base, our Orchestrator can match a capability that is required in a process step with a capability that a specific resource offers and, in that way, match the process step with the resource. A capability of one process step could be matched with the same capability of multiple resources. Orchestrator needs to use optimization techniques and scheduling algorithms to choose one resource for every process step and to optimize the work of resources in a factory. Therefore, Orchestrator needs to determine the allocation of operations to matched resources based on various optimization criteria, supporting real-time orchestration and capability-based process execution. A process step that is ready to be executed is composed of:

- input products that are to be used during the process step execution;
- a capability that a resource must have in order to execute the process step;
- a smart resource that has the required capability and is to execute the process step activity on the input products; and
- output products that are the result of executing the process step activity on input products.

Orchestrator also needs to take care of production logistics. It needs to identify and add storage containing required materials, parts, and products into a DetL process model and process steps that facilitate the transportation of materials, parts, and products, and the movement of resources between storage and workstations. Production logistic activities significantly impact production processes as they require a lot of time to execute [231], so these activities must be well organized. Orchestrator also takes care of machine configurations. Based on the knowledge gathered from Knowledge Base, our Orchestrator can infer whether machine configuration steps need to be added to the process to enable further activities, e.g., calibrating an Automated Guided Vehicle (AGV) after movement so the mounted gripper has the required precision. If several process variations are modeled (e.g., a hole can be made by borer drilling and some sanding, or by laser drilling), Orchestrator also needs to choose the best option in a current state based on a factory topology, available resources, or certain custom-made optimization criteria.

In this Ph.D. thesis, we look at Orchestrator as a black box, as it is a part of another research. It is introduced to provide a context in which MultiProLan is used. An internal structure of Orchestrator used in our MD solution can be found in [4,5].

**Step IV – Generation of high-level instructions.** *Instruction Generator* automatically generates *high-level instructions* from a *resource-aware* production process model. Such an M2T transformation creates instructions with the same structure, independent of specific resources.



Therefore, if new resource types are added to the production system or some resource types are replaced with other ones, these high-level instructions do not need to be changed.

The aim of MultiProLan models is not only to serve documentation purposes but also to lead production process execution in a smart factory. As MultiProLan is created to be a formal language with exact and precise semantics, modeling concepts are machine-readable and understandable, enabling the automatic transformation of modeling concepts into instructions to execute process operations. DetL models have all the technological details needed for the automatic generation of instructions that resources will execute. A process designer can initiate the automatic generation of instructions from a DetL model through Process Modeling Tool by choosing the existing model from Knowledge Base. Another option is the automatic generation of instructions that Orchestrator can initiate after it finishes the automatic transformation of a MasL model to a DetL model. Knowledge Base Management System obtains a DetL model from Knowledge Base and sends it to Instruction Generator, which transforms the model into a set of high-level, generic instructions. These instructions are serialized as JSON objects that contain information on:

- process steps;
- required process steps' capabilities with parameters;
- input and output products;
- specific storage where products are stored or need to be stored, i.e., the material flow; and
- smart resources that are to perform process steps with resources' protocols on which instructions need to be sent;

The high-level, generic instructions are sent to Digital Twin. In our case, Digital Twin can be used for the simulation only or for forwarding instructions to the Production System's smart resources through appropriate communication protocols. Using a digital twin in the simulation-only mode can decrease production failures, provide insight into poorly modeled process steps, and enable the optimization of resources and processes [122]. Furthermore, running simulations makes it possible to predict the impact of process steps on the final product [121].

**Step V – Transformation of high-level instructions into resource-specific instructions.** *Protocol transformation components* stored in *Digital Twin* transform *high-level instructions* into *resource-specific instructions*. Such a Text-to-Text (T2T) transformation creates instructions that are to be executed by resources. Each resource supports a *communication protocol*, and if a new resource type appears in *Production System*, a new protocol transformation component will be created.

Whenever Digital Twin is used to forward instructions to Production System, a transformation of high-level, generic instructions into resource-specific instructions needs to be performed by protocol transformation components. These resource-specific instructions are then passed to real-world resources, whether machines or humans, in the order defined by the control flow through an appropriate communication protocol. An appropriate protocol transformation component is selected to do the transformation based on a protocol defined for each smart resource in a DetL model.

Digital Twin has different protocol transformation components that transform high-level instructions into machine-specific instructions depending on the protocols of the machines. Whenever a new type of machine is added to the shop floor, a new protocol transformation component needs to be added to Digital Twin to support the transformation of high-level instructions into machine-specific instructions for the newly added machine. Also, the knowledge about a newly added resource needs to be stored in Knowledge Base through Knowledge Base Management System. The machine-specific instructions are sent wirelessly to machines and robots to execute them on the shop floor, after machines and robots send feedback about the execution performance. The feedback is forwarded from Digital Twin to Process Modeling Tool, enabling monitoring of the process execution. Each process step can be in the execution phase, executed successfully, or an error can occur. Whenever an error occurs, Digital Twin also forwards information about it to Orchestrator so the error can be handled, and the executed process can be reorchestrated.

A human protocol transformation component is also embedded into Digital Twin, transforming high-level, generic instructions into human-readable instructions. Human workers receive these instructions on their tablets, monitors, or smart watches, and the workers respond through these devices and send feedback on whether instructions are executed successfully or an error occurred. Digital Twin also updates the digital footprint of all resources it manages.

### 5.2.2 Main Steps of Transforming Production Process Models into Manufacturing Documentation

Our MD approach comprises the following two steps of automatic generation of manufacturing documentation. As a precondition to these steps, MultiProLan models have to be created.

**Step \*** – **Generation of manufacturing documentation of different types.** *Documentation Generators* automatically generate *manufacturing documentation* of different types from *resource-agnostic* or *resource-aware* production process models. Such an M2T transformation creates new manufacturing documents or updates existing documents and creates new versions of them.

Documents of various types need to be created and stored when specifying production processes in manufacturing companies. Production processes are mostly described in a textual form, often without an accompanying graphical model. MultiProLan aims to unify the content of various manufacturing documentation into a single model, thus providing a single point of knowledge. Due to different procedures, legal regulations, and internal or external standards in a factory, it may still be required for the documentation to be stored in a textual form. Therefore, the automatic generation of manufacturing documents from MultiProLan models is implemented.

The manufacturing documentation needs to be updated whenever a process or a product changes. In the context of Industry 4.0, where mass customization replaces mass production, product and process changes are common, so the documentation needs to be updated frequently. In traditional manufacturing, these changes are often stored as separate documents [202]. This increases the possibility of human errors while introducing the changes and decreases the level of documentation quality, as vast amounts of content from previously created documents need to be rewritten or copied into new documents. Therefore, the automatic generation of manufacturing documents from MultiProLan models can:

- make the documentation modifications easier;
- decrease the documentation creation time;
- increase the documentation consistency and synchronization with actual production process models; and
- reduce the likelihood of human error.

There are five Documentation Generators implemented to enable M2T transformations – the generation of: Bill of Materials (BOM), Bill of Materials and Operations (BOMO), Flow Process Charts (FPCs), Process Failure Mode and Effect Analysis (PFMEA), and user manuals [12]. By using Process Modeling Tool, a process designer can initiate the generation of the documents based on the newly created MasL/DetL model or the existing MasL/DetL model imported from Knowledge Base. Depending on the document chosen to be generated from a MultiProLan model, appropriate Documentation Generator is initiated, after which the automatically generated document is ready to be stored. Therefore, it is not necessary to manually create all the documents, reducing the time needed for the process design phase. As MultiProLan unifies different documents into a single model, process and quality engineers can work together on modeling processes with a single language while simultaneously getting the documents required in a company.

**Step \*\*** – **Manufacturing documentation storage.** Generated *manufacturing documentation* is stored in *Documentation Storage*, allowing various stakeholders to access the documentation when needed.

Generating and storing the manufacturing documentation from MultiProLan models serves for subsequent production analysis and compliance with procedures, legal regulations, and standards. The automatic generation of manufacturing documentation can help keep the documentation up to date while lowering the costs needed for manual creation and update of the documentation.

In the following section, we summarize the objectives our MD solution aims to achieve, with a special focus on its two main components – MultiProLan and Process Modeling Tool.

### 5.3 Objectives of the MD Solution

Our MD solution, including MultiProLan and Process Modeling Tool, has multiple objectives it needs to achieve. These objectives are summarized to better denote the reasons for creating such a solution and to point out what are the goals of our MD solution and its components. The MD solution should fulfill the following objectives:

- The automatic transformation of MasL into DetL models should help process designers when modeling processes, as the manual creation of DetL production process models is a burdensome task. Also, the automatic transformation should enable fast switching between different production systems as a single MasL model is used, and DetL models can be automatically created. Therefore, the time-consuming task of manually adapting process specifications for different production systems can be replaced with the automatic one. Additionally, an analysis and a comparison of different DetL process models of the same product should be possible, choosing a production system in which the product will be produced optimally.
- The automatic generation of high-level instructions from DetL models should enable manufacturing products based on their production process models. Additionally, high-level instructions that are to be transformed into textual descriptions with images and videos and sent to human workers one by one via tablets or smart watches should enable guided production. Such production should help workers, especially the novice ones, to better understand production processes and perform each process step more easily. Also, such production can lower the time spent by experts helping novice workers, thus improving performance in a factory of both these worker categories.
- The solution should support a process monitoring feature by gathering feedback from resources while executing process steps and sending the feedback to Process Modeling Tool. Therefore, the execution of process steps can be presented in the tool, making it possible to present successfully executed process steps, currently executed process steps, and errors that occur during production. Thus, better control of the process execution and a faster response to occurred errors can be utilized. Also, a current state of the process execution can be presented to human workers, indicating which exact step they currently execute and what are the following process steps, gaining an overview of the whole process. Gathered feedback will also be stored in a database, creating a possibility for process analysis, which can lead to the detection of process anomalies, bottlenecks, and failures. Based on these findings, production processes can be optimized, and unscheduled system shutdowns can be prevented as some failures can be mitigated.
- Instead of a time-consuming manual creation of manufacturing documentation of different types, an automatic generation of such documentation should decrease the time invested by process designers. Also, the manual creation of documentation is usually error-prone, as many documents need to be created. The volume of needed documentation increases significantly in the Industry 4.0 context, where multiple process and product variations exist. Additionally, as a product or its process may change over time, all manufacturing documents related to the product must be updated. By using manufacturing Documentation Generators, versioning and keeping the manufacturing documentation up to date should be done automatically, reducing the time needed for such tasks and possible errors that may occur during manual writing.

- The MD system should contribute to flexible production in the context of Industry 4.0, introducing factories to the digital transformation process.

The developed DSML – MultiProLan should fulfill the following objectives:

- MultiProLan should be a capability-based production process modeling language, allowing Orchestrator to match process steps with resources automatically, thus contributing to flexible production.
- MultiProLan should have a core set of concepts to describe production process models suitable for the automatic generation of executable instructions. Among the core concepts, the language should cover modeling concepts such as material flow, error handling, and product and process variations, making models ready to cope with the production flexibility challenge. In addition, the requirements presented in Section 4.2.1 should be fulfilled by the language, allowing various processes to be modeled.
- MultiProLan should support the modeling of all production details required for the automatic instruction generation and execution but not be too complex for a human to comprehend. It would be hard to model production processes with all the details required for the execution while keeping models clear, concise, and appropriate for initial process analysis. Accordingly, two levels of detail should be implemented to distinguish between MasL and DetL process models. By creating two levels of detail, production process models will become independent of the production system details, and thus efforts needed during the production process modeling will be reduced, as they will be modeled in a generic way. Also, such generic process models can be utilized in multiple different production systems.
- MultiProLan should include and unify concepts from different types of manufacturing documentation to support automatic documentation generation from a single model. A production process model should store knowledge that is usually stored across different sheets and documents, making such a model a single point of knowledge. Such unification should allow various users, such as process and quality engineers, to collaborate on creating the same model from different viewpoints.
- MultiProLan should:
  - speed up and increase the precision with which production processes are designed;
  - decrease the number of faults during process design; and
  - enable faster changes in production process models.

Process Modeling Tool should fulfill the following objectives:

- The tool should support process monitoring, being able to get process execution feedback. The process model's production steps should change color depending on the execution status. By presenting execution status to users, they can monitor process execution and intervene if needed, solving issues during production.
- The tool should support mechanisms to lower the complexity of process diagrams, making them more readable:
  - The tool should support different modeling layers, being able to show or hide a set of modeling concepts related to the specific layer. The usage of layers would allow different users, such as process and quality engineers, to work together on the same process model, showing just the modeling concepts relevant to them. Therefore, the amount of information and modeling elements on a process diagram would be lower for each class of users, allowing them to model processes more easily and to be focused only on their modeling task.
  - Process modeling elements that are containers for other elements should have a mechanism to show or hide contained elements, allowing to have more or fewer details on a process diagram.
  - A process model should contain sub-processes, allowing the decomposition of a process model and the reusability of existing processes. The modeling tool should

allow for easy usage of sub-processes, being able to easily open and present sub-processes when needed.

- As there can be multiple process and product variations presented on a process diagram, the complexity and volume of the diagram can increase significantly. Therefore, the tool should have a filter function that would allow a user to choose which process or product variation to present, hiding all other variations from the diagram. Such a feature would allow users to be focused only on variations they want to create, view, or update.
- A zoom feature should also be implemented to allow an overview of the whole process diagram or focus on the part of the diagram.

In order to use MultiProLan, Process Modeling Tool, and the whole MD solution effectively, the presented objectives should be achieved. Therefore, the full potential of the proposed solution can be reached.

## 5.4 Summary

In this section, we presented our MD solution comprising a novel system and an approach for modeling and automatic execution of production processes. The presented MD solution can be used by process designers to specify resource-agnostic production process models. These models can be enriched with details of a chosen production system in which a production process is to be executed. Such enrichment of resource-agnostic production process models can be done automatically by Orchestrator, based on production system models stored in Knowledge Base, creating resource-aware production process models. These models can be automatically transformed into executable resource instructions by Instruction Generator and protocol transformation components stored in Digital Twin. Instructions can be sent to the shop floor resources to perform specified production process steps. Our MD solution can also be used to automatically transform production process models into manufacturing documentation.

The main component of our MD solution is MultiProLan, a novel DSML for production process modeling. Before creating a novel DSML, modeling concepts need to be carefully analyzed and selected. Therefore, a production process modeling domain analysis is conducted and presented in the following section.



## 6 Analysis of the Production Process Modeling Domain

Creating a new language requires formulating its abstract syntax based on the relevant identified domain concepts. Therefore, we analyzed the domain of production process modeling and identified its main concepts before creating the MultiProLan language. Domain knowledge is gathered from research papers, technical documentation, use cases, and by talking to domain experts. Feature-Oriented Domain Analysis (FODA) is used as a domain analysis method [93]. The FODA method is used to analyze a domain and identify features that are usually expected to support software development.

A feature model comprises the typical features of a family of systems in the domain and their relationships [93]. A feature represents a property of a system that directly affects stakeholders that use the system. Features are connected via structural relationship, meaning features consist of other features, representing a logical grouping of features. Inside a feature group, alternative features can exist, representing a specialization of a more general category and indicating that no more than one specialization can be used. There are also mandatory and optional features, indicating which features must exist and which ones are optional.

In this thesis, we use the standard FODA notation [93,155] extended to support feature and group cardinalities in order to create cardinality-based feature models [232]. A feature cardinality represents how often a sub-feature or an entire sub-tree can be replicated. The feature cardinality is presented in a form  $[n..m]$ , indicating that at least  $n$  and at most  $m$  sub-features need to be replicated. If no cardinality is presented for a sub-feature, an unfilled circle of a relation represents the  $[0..1]$  cardinality, while a filled circle represents the  $[1..1]$  cardinality. A group cardinality can be specified for a whole group of sub-features, indicating how many sub-features can be selected. The group cardinality is presented in a form  $\langle n..m \rangle$ , indicating that at least  $n$  and at most  $m$  group features need to be selected. If no cardinality is presented for a group, the  $\langle 1..1 \rangle$  cardinality is assumed.

To create cardinality-based feature models, we used Yet Another Feature Modeling Tool (YAFMT) [233], representing a collection of Eclipse plug-ins for feature modeling. This tool was also used by Samimi-Dehkordi et al. [234] when they analyzed a domain for the Model-Driven Engineering of Bidirectional Transformations via Epsilon (MoDEBiTE) language. The only difference in the syntax of FODA models that we encountered when using YAFMT compared to the standard FODA notation is that when specifying a group of features, a square is used instead of a circle at the end of relations. There are also other feature modeling tools, such as FeatureIDE [235,236] – an Eclipse-based open-source framework for Feature-Oriented Software Development (FOSD), a paradigm for the construction, customization, and synthesis of software systems. However, we chose the YAFMT tool due to the good-looking graphical appearance of feature model diagrams, easiness of use, and cardinality-based feature modeling.

Production processes typically follow a strict set of rules or guidelines in order to turn raw materials into a quality finished product. They can be described in various ways and at different

levels of detail. As we aim to create production process models that can be used for the automatic instruction generation and execution of the generated instructions, production processes need to be described with all the details required for the execution. A production process can be understood from several different perspectives. Here we focus on the operational, resource, and control-flow perspectives and present them as cardinality-based feature models. It should be noted that these feature models do not cover all the details of the production processes domain, but a minimal set of features needed for dynamic production orchestration and automatic process execution and error handling. The domain analysis of production process modeling and related FODA models were published in [11].

This section is composed of the following three subsections. In Section 6.1, we present the cardinality-based feature model of the operational and resource perspectives of production process steps, while in Section 6.2, we present the control-flow perspective of production processes as the cardinality-based feature model. The summary of the domain analysis is outlined in Section 6.3.

## 6.1 Operational and Resource Perspectives

The operational and resource perspectives are presented in Figure 6.1 as a cardinality-based feature model of a production process step. The operational perspective describes elementary units of work (*Process Step*), and the resource perspective provides an organizational structure in the form of human and machine roles responsible for executing process steps. A process step feature is the root of the FODA tree in Figure 6.1. A process step requires a capability (*Capability*) that represents the "implementation-independent potential of an Industry 4.0 component to achieve an effect within a domain" [142], i.e., a skill needed to execute the process step. For example, a robot can have different capabilities depending on which tools are mounted, and a human worker can have different capabilities depending on their expertise or completed training. A process step capability can have different parameters (*Capability Parameter*) or constraints (*Capability Constraint*). A capability parameter is used to specify additionally how a capability needs to be used. Parameters usually represent conditions, attributes, or settings that can be varied to affect the execution of process steps [141]. For example, if a product needs to be placed on a pedestal, the coordinates of where the product is to be placed need to be specified. A capability constraint indicates what is an additional requirement to execute a process step. For example, for the *pick* capability, a constraint can be that a gripper needs to be able to pick up objects wider than 0.5 m. Different robots can all have the *pick* capability, but they can pick objects of different sizes, as their grippers are not the same. Also, a constraint for the *pick* capability can be defined as a minimal mass that the resource can handle, e.g., 100 kg. Human workers and some robots can pick objects, but there is a limitation on how much weight they can lift. Thus, only some of the robots can lift such objects.

A process step can exist without any products (*Product*). For example, if a process step is a movement operation of a resource or some of the resources just need to be configured, none of the products are included as these operations do not require them. A process step can include multiple products of different kinds (*Product Kind*): input (*Input*) and output (*Output*) products. A process step capability is used on input products to create output products of the process step. Products can be of different types (*Product Type*) [141]:

- raw material (*Raw Material*) – represents any physical material used in a product creation;
- an intermediate product, a part, or a component (*Component*) – represents an output of a process step that is an input to another process step and requires further transformations within a production system; and
- a finished good or a final product (*Finished Good*) – represents any good or service offered to satisfy someone's needs.



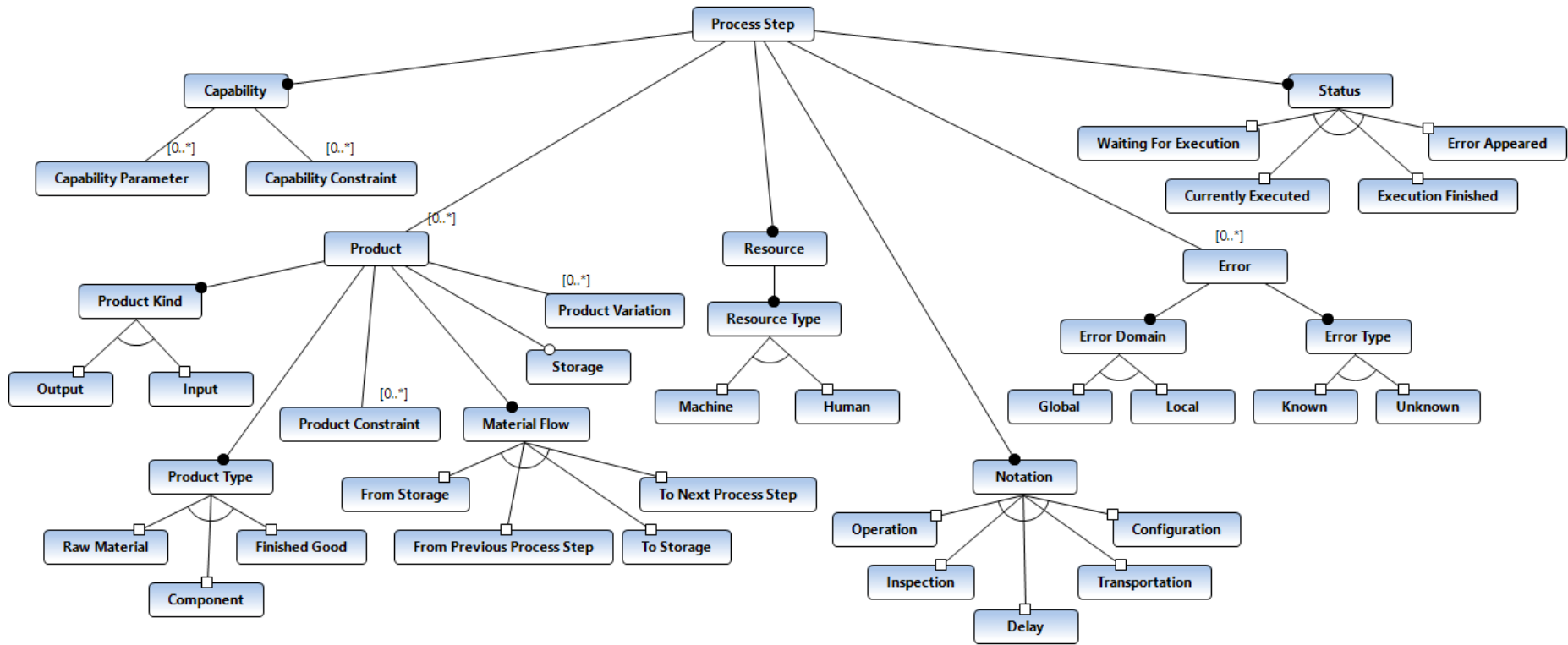


Figure 6.1. A FODA model of a production process step suitable for an execution.

For each product, different constraints (*Product Constraint*) can be specified. Like capability constraints, product constraints can be used to determine which resources can execute a process step. A product constraint can be seen as an attribute, as it specifies a product's property, such as height, width, depth, mass, or color.

It is also important to specify a flow of materials or products (*Material Flow*) between storage and execution resources of process steps. A material flow represents products entering from or leaving to another production system [141], or in the case of a single production process, it represents materials and products entering from or leaving to another process steps. For each input product, it must be known whether it needs to be taken from storage (*From Storage*) or is a result of a previous process step (*From Previous Process Step*). Also, for each output product, it must be known whether it needs to be placed in storage (*To Storage*) or it will be used in some further process steps (*To Next Process Step*). If an input product needs to be taken from storage or an output product placed in storage, the product needs to include information on which storage (*Storage*) that should be.

Each product can have many variations (*Product Variation*), especially in the context of Industry 4.0 and lot-size-one production. As customers require various personalized products in such production, each product variation can differ in the parts and materials it is made. Therefore, a product with all its variations belongs to the same product family.

A smart resource (*Resource*) needs to be specified for each process step to enable its execution. Smart resources can be of different types (*Resource Type*): a robot or a machine (*Machine*) or a human worker (*Human*). A machine represents a device, a piece of equipment, an instrument, a tool, a single machine, or a robot, that performs elementary activities, is used for elementary activities, or makes work easier [141].

Process steps can be represented by using different notations (*Notation*): (i) operation (*Operation*), (ii) inspection (*Inspection*), (iii) delay (*Delay*), (iv) transportation (*Transportation*), or (v) configuration (*Configuration*). An operation is an activity needed to change input products and create output products, e.g., cutting a metal bar. An inspection is an activity needed to check whether a product fulfills certain requirements, e.g., visually inspecting assembled parts. A delay is an activity designed to wait for some other process to be finished, e.g., waiting for a metal bar to cool down. Transportation is an activity needed for changing the location or position of resources or products between storage, e.g., moving a robot to a smart shelf. These four notations are also present in the American Society of Mechanical Engineers (ASME) Flow Process Charts (FPCs) [25], as well as in [2], but without the delay notation. To specify production processes that are suitable for execution, configuration activities also need to be defined. Configuration is an activity needed to configure resources in order to finish different tasks. For example, a robot must determine its position before it executes an operation, change a gripper, or plug itself into a charger.

Various errors (*Error*) can occur when executing process steps, so error handlers need to be specified. An error handler has its domain (*Error Domain*): global (*Global*) – the same error handler can be used for different errors, or local (*Local*) – the error handler is specific to the error and cannot be used for other errors. For example, a global error handler can be utilized to specify steps to recycle a plastic product and a local error handler can be used to specify steps for disassembling a specific product into components. Also, error handlers can be specified for errors of the following types (*Error Type*): known errors (*Known*) – identified errors that can occur in process steps and process designers are aware of them, or unknown errors (*Unknown*) – unidentified errors that can occur in process steps. There can be only one unidentified error handler for each process step and many identified error handlers.

The status (*Status*) of each process step needs to be monitored during the execution of a production process. A process step can be in the following states: (i) not yet executed (*Waiting For Execution*), (ii) currently being executed (*Currently Executed*), (iii) executed successfully (*Execution Finished*), and (iv) an error occurred during the execution (*Error Appeared*).

## 6.2 Control-Flow Perspective

The control-flow perspective is presented in Figure 6.2 as a cardinality-based feature model of a production process. The control-flow perspective describes activities and their execution ordering through different constructors (*Control Structure*), which permit a flow of execution control, e.g., sequence, parallelism, decision, and iteration. Activities in an elementary form are atomic units of work (*Process Step*), and in a compound form, they are a set of activities that modularize an execution order.

A production process or a manufacturing process (*Process*) is the root concept of the FODA tree presented in Figure 6.2. A process has a starting point (*Start*), an ending point (*End*), and a non-empty array of control structures (*Control Structure*). Control structures represent different types of execution flows that contain branches and process steps. They are: (i) sequence (*Sequence*), (ii) parallelism (*Parallelism*), (iii) decision (*Decision*), (iv) iteration (*Iteration*), (v) collaboration (*Collaboration*), (vi) variation (*Variation*), and (vii) unordered set of steps (*Unordered Steps*). During the domain analysis, we encountered multiple examples of these control structures, and some of these examples are discussed in the following paragraphs.

A sequence (*Sequence*) is a non-empty array of process steps (*Process Step*) that must be executed in the exact order they are presented. For example, a robot needs to move to an assembly table and proceed to assemble two parts.

Parallelism (*Parallelism*) has a starting point (*Start Parallelism (Fork)*), an ending point (*End Parallelism (Join)*), and an array of at least two parallel branches (*Parallel Branch*). Each parallel branch is a non-empty array of control structures (*Control Structure*). The branches are to be executed in parallel. For example, there are different independent parts that can be assembled in parallel.

A decision (*Decision*) has two or more branches (*Decision Branch*), and each of them is an array of control structures (*Control Structure*) that can be empty. Every branch has a condition (*Decision Condition*) that needs to be met for the branch to be executed. For example, after inspecting a product, a decision is made whether the product needs to be stored or discarded, depending on the inspection results.

An iteration (*Iteration*) has an iteration branch (*Iteration Branch*) that is a non-empty array of control structures (*Control Structure*) placed between a check point (*Check Point*) and a return point (*Return Point*) that needs to be executed as part of the current iteration. The check point comprises a condition (*Iteration Condition*) that can be checked before the execution of the iteration branch (*Pre-Condition*) or after the execution of the iteration branch (*Post-Condition*). Thus, it is possible to create *while* or *do-until* loops. For example, a metal bar needs to be heated several times.

A collaboration (*Collaboration*) has two or more collaboration branches (*Collaboration Branch*), and one or more messages (*Message*). Each collaboration branch is a non-empty array of control structures (*Control Structure*). Messages are exchanged between process steps contained in some of the control structures. Each message has a source process step (*Source Process Step*) that sends the message to a target process step (*Target Process Step*) that receives the message. For example, one part needs to be held while another one is getting attached to it. The attachment process step should not start before a message arrives that the first part is being held. The holding process step should not end until a message arrives that the second part becomes fully attached to the first one. A collaboration between resources is essential in the context of Industry 4.0, especially a human-machine or human-robot collaboration. Due to the need for highly flexible production, resources are meant to collaborate frequently to produce different products and their variations. Accordingly, a human-robot collaboration combines benefits from both human workers (e.g., sensorimotor skills) and robots (e.g., high-precision skills), making production more effective and efficient [237].

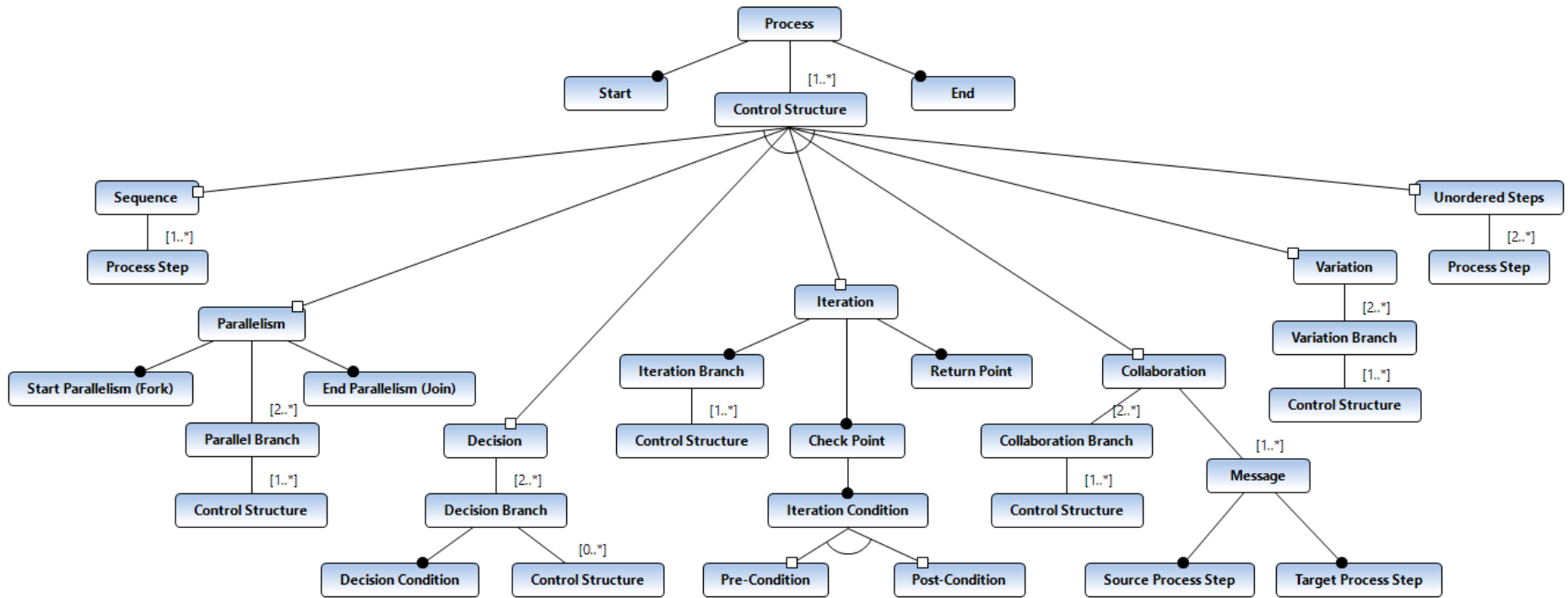


Figure 6.2. A FODA model of a production process suitable for an execution.

A variation (*Variation*) represents a part of a process with two or more variation branches (*Variation Branch*), leading to different product variations in the same product family or the same product but processed in different ways. Each variation branch is a non-empty array of control structures (*Control Structure*). Different product variations can belong to the same product family but differ in some materials or parts. Each variation branch can represent different product variations, containing different activities and products that participate in the process. The exact product variation is chosen before the execution of the process. In addition, an intermediate product or a finished good can be created in different ways, which is what process variations represent. For example, a hole in a component can be created by a drilling process step followed by a sanding process step, or the hole can be made with a laser without needing the sanding afterward. There are many other ways to create a hole, such as water cutting, plasma cutting, or punching [71]. All these variations can be specified, and just before the execution of a process, an optimal variation can be chosen.

An unordered set of steps (*Unordered Steps*) represents a collection of two or more process steps (*Process Step*) that may be executed in any order. For example, two parts need to be picked from the same storage, and it is not important which one should be picked first. Such arbitrarily ordered steps can be useful in cases when workers are allowed to assemble certain parts in any order, and at some time, workers may find out the fastest way to assemble a product, optimizing their performance and needed time. Arbitrarily ordered steps may also be applied for quality assurance steps when several tests need to be done for a part or a product [65].

### 6.3 Summary

To execute a production process model, it should have at least the following features: specified starting and ending points of the process and, between these two points, a sequence control flow with at least one specified process step. The process step comprises a single input product, a capability, a single output product, and a resource. All of the aforementioned represent a minimal set of features needed to execute a process, assuming that a resource already has an input product at a workspace and that transportation of the product is not required. However, in practice, storage and transportation process steps should be specified in the process, as well as the configuration of resources when needed. Additionally, product and capability constraints and capability parameters should be specified when they are required.

The domain of production process modeling is much broader and semantically richer than presented in this thesis. We settled on describing only the given concepts as we focused on a minimal set of features needed for the automatic execution of production processes and error handling. Based on the presented features, we created a novel DSML – MultiProLan that covers all the aforementioned domain concepts, discussed in the following section.



## 7 Multi-Level Production Process Modeling Language

In this section, we present the abstract and concrete syntaxes of *Multi-Level Production Process Modeling Language (MultiProLan)* – a DSML for modeling production processes suitable for dynamic production orchestration and automatic generation of executable resource instructions and manufacturing documentation.

We used the Ecore meta-meta-model, which is a part of EMF [94,95], to create the abstract syntax of MultiProLan based on the domain analysis presented in the previous section. Additional constraints that cannot be expressed by the meta-meta-model concepts, but exist in the production process modeling domain, are expressed by using Object Constraint Language (OCL) [96]. OCL is a standard language created by OMG to overcome the limitations of UML by defining various expressions clearly and unambiguously [97]. It was primarily made to express various constraints of a system, but the newer OCL versions are also used to define different queries, data manipulation, business rules, and model transformations [98].

To create the graphical concrete syntax and to enable the simple implementation of a prototype tool [99], we used the Eclipse Sirius framework [100]. Both MultiProLan and Process Modeling Tool were developed iteratively.

During the design of MultiProLan, several process engineers were involved as domain experts and provided us with different model examples specified as ASME FPCs, BOMs, and textual descriptions. With their help, we identified domain concepts and validated the modeling concepts of our language.

This section is structured as follows. First, an overview and usage of MultiProLan are discussed in Section 7.1. Then, the abstract syntax of MultiProLan is presented in Section 7.2, while its concrete graphical syntax is presented in Section 7.3. Process Modeling Tool, introduced as a part of our MD system in Section 5.1, is discussed in detail in Section 7.4. The summary of developing MultiProLan and Process Modeling Tool is outlined in Section 7.5.

A detailed description of the MultiProLan usage was presented in [11] and [14]. The abstract syntax of MultiProLan was presented for the first time in [10] and extended in [11], while the MultiProLan's concrete syntax was presented through model examples and figures in [10,11,13] and in a more detailed manner in [14].

### 7.1 Overview and Usage of MultiProLan

The main goal of creating MultiProLan is to enable formal specification of production processes that will allow the automatic generation of executable instructions. This way, we want to automate

process execution and thus improve production flexibility. The language is created according to the requirements presented in Section 4.2.1.

MultiProLan is a production process modeling language primarily used in the domain of hardware assembly. Currently, it does not support other production domains, but it can be extended to support them in the future. The language unifies different production process aspects and, therefore, allows the unification of the work from different user groups. At this point of the MultiProLan development, these user groups comprise process and quality engineers. They use MultiProLan together to create production process models that are:

- suitable for automatic generation of executable instructions;
- independent of any production system;
- used to handle errors when they occur;
- comprised of multiple variations of the process and final product; and
- suitable for automatic generation and update of manufacturing documentation.

As manufacturing companies need to have an overview of their processes from different viewpoints [212], the MultiProLan modeling concepts, at the present stage, are grouped across two detail levels and two layers, as seen in Figure 7.1. Two detail levels – *Master-Level* (MasL, i.e., resource-agnostic) and *Detail-Level* (DetL, i.e., resource-aware) – are used to separate modeling concepts independent of an execution platform, i.e., a specific production system, from those that are execution platform-specific, respectively. Furthermore, as MultiProLan follows the principles of skill-based engineering, i.e., process steps are specified through capabilities, MasL models can be automatically transformed into DetL models. Such an automatic transformation is possible by using matching mechanisms implemented in Orchestrator, connecting required capabilities from process steps with offered capabilities from production resources. Assigning resources to MasL process steps and adding transportation and configuration process steps as well, creates DetL process models. As capabilities are independent of any specific production system, these two levels participate in one of the key features of the MultiProLan languages – production process models need to be independent of any production system so that they can be used in various systems, but also, process models need to be executable in a chosen production system.

To ease language extensibility, modeling concepts are additionally grouped across so-called layers. *Execution Layer* is the default layer. It comprises modeling concepts that are crucial for production process modeling and execution. This layer is always visible within the graphical representation of a MasL/DetL model and is required for other layers to be applied. In Execution

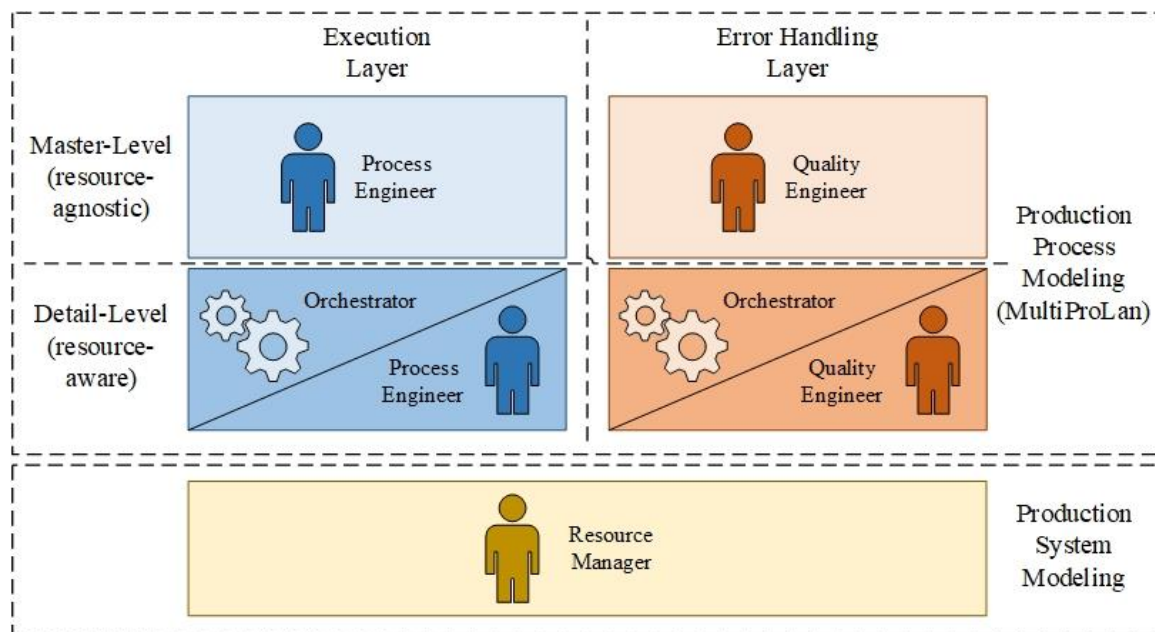


Figure 7.1. Context and architecture of MultiProLan.



Layer, process steps are modeled with input and output products, capabilities, and resources. Also, the control flow of process steps and the material and message flows are modeled in this layer, as well as different process variations. Errors that might occur during production and process steps needed for their handling are grouped within *Error Handling Layer*, and these modeling concepts are used mainly by quality engineers. Most of the time, process and quality engineers collaborate to create a process model. However, sometimes process and quality engineers want to be focused only on their part of process modeling. Also, as there may be a significant number of details on a process model diagram, a mechanism should exist to reduce the level of detail and enable process and quality engineers to be more focused on their work. Thus, Error Handling Layer can be hidden from a process engineer and can be made visible to a quality engineer. Hiding layers from other users enables them only to be focused on their part of the process modeling. However, both layers can be presented simultaneously so all users can work together and optimize a production process model. In Figure 7.1, we present user groups that use modeling concepts grouped across detail levels and currently supported layers. Additional layers are under development, such as *Risk Management Layer*, which presents a part of a production process model specifying different risk factors for each process step and the overall risk associated with resources. This layer will be used mainly by HSE managers. Modeling various production safety and energy consumption aspects needs to be considered as well but it is part of our future work.

The two detail levels and two different layers discussed are part of the production process modeling. To create DetL models from MasL models, a specification of a production system also needs to be possible. The production system modeling is done by a resource manager, as seen in the bottom part of Figure 7.1. The language for production system modeling is another language used in Resource Modeling Tool (introduced in Section 5.1), but it has similar modeling concepts to MultiProLan, such as capabilities, resources, storage, and material flows. Thus, we can say that they both belong to the same language family – the languages are compatible, have overlapping concepts, and complement each other, but they were developed and are considered separately. The production system modeling language and Resource Modeling Tool are part of another research [5] and will not be discussed in detail in this thesis.

Four main characteristics that differentiate MultiProLan from the languages and approaches presented in Section 4.2.2 are:

- **A core set of concepts needed to describe production process models suitable for automatic generation of executable instructions (C1).** The first characteristic enables process designers to use MultiProLan to model production processes ready for automatic instruction generation and execution.
- **Two detail levels that allow distinction between resource-agnostic and resource-aware production process models (C2).** As it is hard to model production processes with all the details required for the execution while keeping models clear, concise, and appropriate for initial process analysis, two detail levels are introduced. The second characteristic makes modeling easier for process designers by allowing them to specify production processes at a high level, independent of the exact resources that will execute the process. Furthermore, to automatically enrich these abstract models with specific shop floor details, methods to transform resource-agnostic into resource-aware models are required.
- **A core set of concepts needed to model error handling (C3).** The third characteristic enables process designers to use MultiProLan to model production process errors that may occur. During the process execution, various errors may occur, and they need to be modeled alongside the corrective steps to remove the damage caused by the errors.
- **Unification of concepts from different manufacturing documents, such as BOM, FPC, and PFMEA, into a single, uniform production process model, allowing automatic generation and update of manufacturing documentation from the single model (C4).** The fourth characteristic allows various users, such as process and quality engineers, to collaborate on creating the same model from different viewpoints. Thus, different aspects of a production process can be integrated into a uniform model instead of having several documents covering various aspects. Accordingly, having a uniform process model with

information usually stored in multiple documents makes it possible to automatically generate and update the manufacturing documentation from such a model.

When modeling a production process by using MultiProLan, there are several assumptions or constraints in place that a modeler must be aware of:

- A production process may be modeled for execution in a single facility or a single smart factory. However, multi-facility collaboration is not yet supported.
- Existing factory resources executing modeled process steps are smart enough to understand basic high-level instructions, such as *pick*, *place*, *move*, or *assemble*.
- Once online, resources introduce themselves to the system and provide the needed semantics on how to use them.
- Storage in a production facility, such as smart shelves, is already filled with materials and parts and cannot be depleted.

In the following sections, the abstract and concrete syntaxes of MultiProLan are described, as well as Process Modeling Tool.

## 7.2 Abstract Syntax of MultiProLan

To make the MultiProLan meta-model diagram more concise and easier to understand, we divided its concepts into four parts. The first two parts of the meta-model cover modeling concepts of MasL and DetL at Execution Layer. The third part of the meta-model covers modeling concepts at Error Handling Layer, while the fourth part of the meta-model covers the extensions needed for the automatic generation of manufacturing documentation. Therefore, we divided this section into four subsections based on these four meta-model parts.

### 7.2.1 Master-Level Modeling Concepts at Execution Layer

The MasL part of the MultiProLan meta-model used at Execution Layer is depicted in Figure 7.2. Process designers use these modeling concepts to create MasL models. A production process is modeled using the *Process* class, which represents the root model element. A process version (*version*) can be specified as models are stored in Knowledge Base and can be changed or reused at any time. A process is composed of at least one process element (*ProcessElement*), and process elements can come in the form of process steps (*ProcessStep*), gates (*Gate*), unordered sets of steps (*UnorderedSteps*), or sub-processes (*SubProcess*). Each process also comprises relationships (*Relationship*) between process elements. As the knowledge of the execution starting point is needed, a start process step must be referenced from a process (*startStep*).

There are two types (*type*) of relationships (*ERelationshipType*):

- *FLOW* – representing a control flow between process elements; and
- *COLLABORATION* – representing a message flow between process steps.

Relationships have the message attribute (*message*) specified whenever a message needs to be sent between collaboration process steps. Also, relationships have the logical condition (*logicalCondition*) specified whenever they are used in selection or iteration patterns. There are input (*inRelationship*) and output (*outRelationship*) relationships connected to a process element, and each relationship must have its source (*source*) and target (*target*) process element.

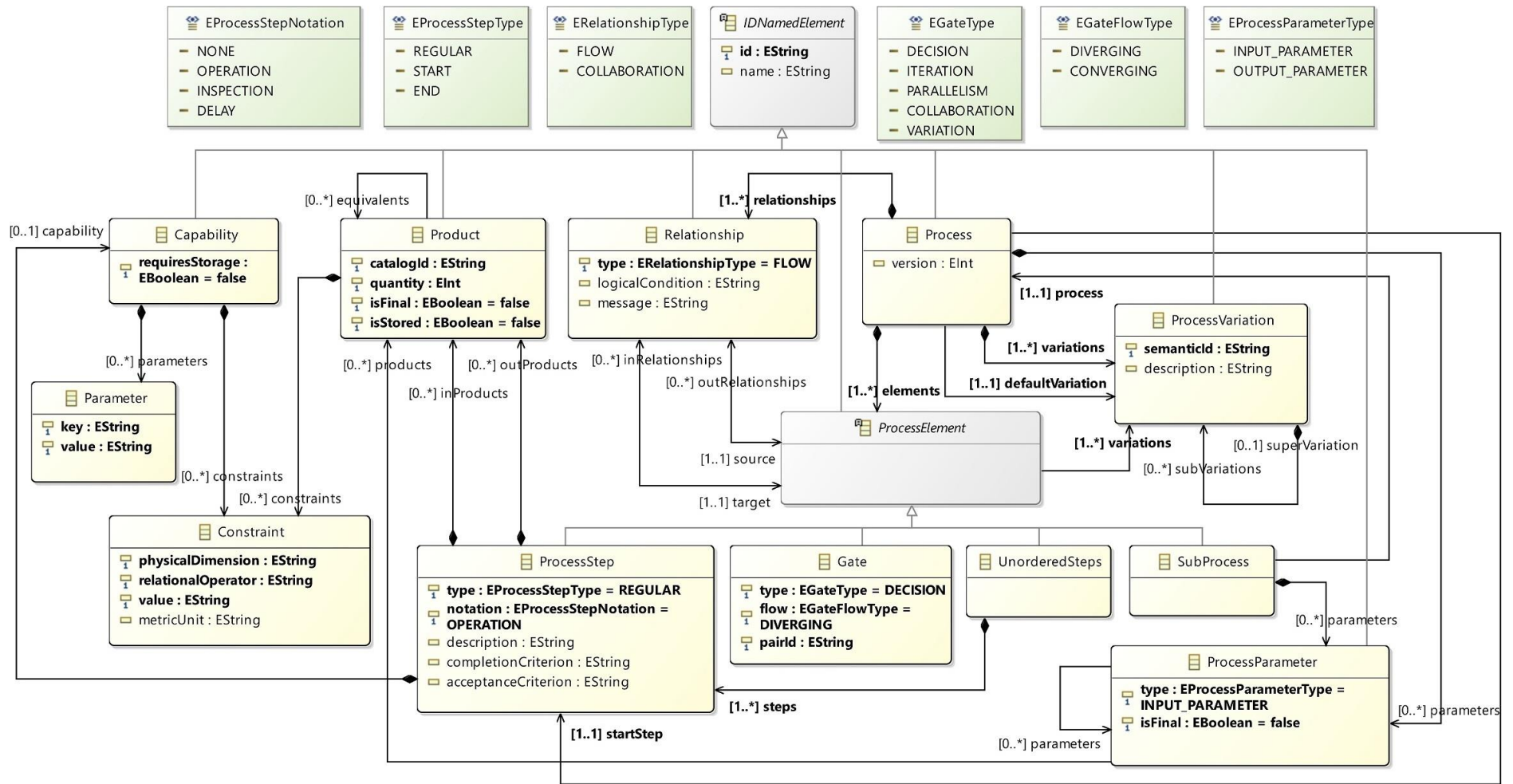


Figure 7.2. The first part of the meta-model used for MasL model creation at Execution Layer.

Additional constraints exist when specifying a relationship:

- relationship's source and target cannot reference the same process element;
- there cannot be two flow-type relationships that have the same source and target elements;
- each flow-type relationship cannot have the *message* attribute specified;
- each collaboration-type relationship must have the *message* attribute specified; and
- each collaboration-type relationship cannot have the *logicalCondition* attribute specified.

These constraints are specified in OCL, which code is implemented inside the Ecore-based meta-model. The code of these constraints is presented in Listing 7.1.

```

1  invariant relationshipsCannotBeRecursive('Source and target elements of each
    relationship must be different.'):
    Relationship.allInstances()->forAll(r | r.source <> r.target);
2  invariant singleFlowRelationshipBetweenElements('There cannot be more than a
    single flow-type relationship between the same source and target elements.'):
    Relationship.allInstances()->forAll(r1, r2 | ((r1 <> r2) and
    (r1.type = ERelationshipType::FLOW and r2.type = ERelationshipType::FLOW))
    implies ((r1.source <> r2.source) or (r1.target <> r2.target));
3  invariant flowRelationshipCannotHaveMessage('The message attribute cannot be
    specified for a flow-type relationship.'):
    Relationship.allInstances()->forAll(r | r.type = (ERelationshipType::FLOW)
    implies (r.message = null or r.message = ''));
4  invariant collaborationRelationshipMustHaveMessage('The message attribute must be
    specified for each collaboration-type relationship.'):
    Relationship.allInstances()->forAll(r |
    r.type = (ERelationshipType::COLLABORATION) implies (r.message <> null
    and r.message <> ''));
5  invariant collaborationRelationshipCannotHaveLogicalCondition('The logical
    condition attribute cannot be specified for a collaboration-type relationship.'):
    Relationship.allInstances()->forAll(r |
    r.type = (ERelationshipType::COLLABORATION) implies
    (r.logicalCondition = null or r.logicalCondition = ''));

```

**Listing 7.1.** Constraints related to flow-type and collaboration-type relationships.

A process step is composed of a capability (*Capability*) required to perform an activity and products (*Product*) on which the activity is to be performed. A capability has an indicator of whether it requires storage (*requiresStorage*) to be executed in the real world.

Each product has an identifier of a product type it belongs to (*catalogId*), as well as an indicator of whether it is a final product (*isFinal*) or an intermediate one. A process step can have input products (*inProducts*), representing products on which an activity is to be performed, and output products (*outProducts*), representing products that are the result of performing the activity. There is also the quantity attribute (*quantity*), representing the number of products used as input to a process step or produced as output of the process step.

A material flow should be specified for every product. An input product can be considered as equivalent (*equivalents*) to products of some previous process steps, or it can be brought from storage (*isStored*). An output product can be used in the following process steps or stored in storage.

Every product and capability may have constraints (*Constraint*), such as dimensions, color, and mass. Constraints will be considered by Orchestrator when it decides which smart resources can perform a process step. A constraint has a physical dimension (*physicalDimension*), such as mass and width, a relational operator (*relationalOperator*), such as =, !=, <, >, <=, >=, and a value (*value*). Optionally, it has a metric unit (*metricUnit*), such as gram and meter, but there is no metric unit for constraints such as color. A product has constraints with the equivalency relational operator, as they describe its characteristics (e.g., width = 0.5 meters), while capability constraints can have any relational operator. For example, if the *pick* capability has the following constraint: width < 1 meter, it means that a resource with the *pick* capability is needed to pick a product, and not just any resource, but the resource that can pick objects whose width is less than 1 meter. The

*relationalOperator* attribute domains for product and capability constraints are defined in the *Product* and *Capability* classes by using OCL and they are presented in Listing 7.2.

1	<b>invariant</b> productConstraintRelationalOperatorDomain('The equivalency (=) relational operator must be used in product constraints.'): self.constraints->forAll(c   c.relationalOperator = '=');
2	<b>invariant</b> capabilityConstraintRelationalOperatorDomain('A capability constraint relational operator must have one of the following values: =, !=, <, >, <=, >='): self.constraints->forAll(c   Set{'=', '!=', '<', '>', '<=', '>='} ->includes(c.relationalOperator));

**Listing 7.2.** The specification of the relational operator domain in product and capability constraints.

Some capabilities may require parameters (*Parameter*) to be specified. For example, the drilling position must be specified to drill a hole. A parameter has two attributes: a key or a name (*key*) of the parameter and a value (*value*) of such a parameter.

Process steps can be of different types (*type*), having one of the following values (*EProcessStepType*):

- *START* – the first process step;
- *END* – the last process step; and
- *REGULAR* – other process steps that may contain capabilities and products.

The start and end process steps have additional constraints implemented using OCL and presented in Listing 7.3:

- the start and end process steps do not have any capabilities or products;
- exactly one start process step and one end process step have to exist per each production process model;
- the start process step cannot have any input flow-type relationships (but it is possible for the start process step to contain input error-type relationships, as discussed in Section 7.2.3), and must contain a single output flow-type relationship; and
- the end process step cannot contain output relationships and must contain a single input flow-type relationship.

1	<b>invariant</b> startProcessStepCannotContainProductsOrCapability('The start process step cannot contain products or a capability.'): ProcessStep.allInstances()->forAll(ps   (ps.type = EProcessStepType::START) implies (ps.inProducts->isEmpty() and ps.outProducts->isEmpty() and ps.capability = null));
2	<b>invariant</b> endProcessStepCannotContainProductsOrCapability('The end process step cannot contain products or a capability.'): ProcessStep.allInstances()->forAll(ps   (ps.type = EProcessStepType::END) implies (ps.inProducts->isEmpty() and ps.outProducts->isEmpty() and ps.capability = null));
3	<b>invariant</b> startProcessStepMustExist('A single start process step must exist.'): ProcessStep.allInstances()->select(ps   ps.type = EProcessStepType::START) ->size() = 1;
4	<b>invariant</b> endProcessStepMustExist('A single end process step must exist.'): ProcessStep.allInstances()->select(ps   ps.type = EProcessStepType::END) ->size() = 1;
5	<b>invariant</b> startProcessStepCannotContainInputFlowRelationship('The start process step cannot contain an input flow-type relationship.'): ProcessStep.allInstances()->forAll(ps   (ps.type = EProcessStepType::START) implies (ps.inRelationships->forAll(r   r.type <> ERelationshipType::FLOW)));
6	<b>invariant</b> startProcessStepMustHaveSingleOutputRelationship('The start process step must have a single output flow-type relationship.'): ProcessStep.allInstances()->forAll(ps   (ps.type = EProcessStepType::START) implies (ps.outRelationships->size() = 1));
7	<b>invariant</b> endProcessStepCannotHaveOutputRelationship('The end process step cannot have an output relationship.');

8	<pre> ProcessStep.allInstances()-&gt;forall(ps   (ps.type = EProcessStepType::END)   implies (ps.outRelationships-&gt;isEmpty())); invariant endProcessStepMustHaveSingleInputFlowRelationship('The end process step must have a single input flow-type relationship.'):   ProcessStep.allInstances()-&gt;forall(ps   (ps.type = EProcessStepType::END)   implies (ps.inRelationships-&gt;select(r   r.type = ERelationshipType::FLOW)   -&gt;size() = 1)); </pre>
---	--

**Listing 7.3.** Constraints related to the start and end process steps.

By introducing process step types, two more constraints need to be defined and they are implemented using OCL as presented in Listing 7.4:

- the *startStep* relation from the *Process* class must reference the start process step; and
- a collaboration-type relationship can only be created between regular process steps.

1	<pre> invariant startStepRelationMustReferenceStartProcessStep('The startStep relation from the process must reference the start process step.'):   self.startStep.type = EProcessStepType::START; </pre>
2	<pre> invariant collaborationRelationshipMustConnectRegularProcessSteps('Collaboration relationships must connect only regular process steps.'):   Relationship.allInstances()-&gt;forall(r     (r.type = ERelationshipType::COLLABORATION) implies   ((ProcessStep.allInstances()-&gt;exists(ps   ps.type = EProcessStepType::REGULAR   and ps = r.source)) and (ProcessStep.allInstances()-&gt;exists(ps     ps.type = EProcessStepType::REGULAR and ps = r.target)))); </pre>

**Listing 7.4.** Constraints related to the *startStep* relation and the collaboration-type relationship.

A process step has a notation (*notation*), having one of the following values (*EProcessStepNotation*):

- *NONE* – a notation needed for the start and end process steps;
- *OPERATION* – an activity that changes input products and creates output products;
- *INSPECTION* – an activity to perform various checks on products; and
- *DELAY* – necessary waiting activities.

A process step can have its description (*description*) to provide additional information about it and is used when a human worker executes a process step. A process step can have a completion criterion (*completionCriterion*), indicating when a process step is finished. It also can have an acceptance criterion (*acceptanceCriterion*), providing boundaries in which completion of the process step is acceptable. For example, a wooden plank needs to be cut, and its width should be 500 mm, representing the completion criterion. However, it is acceptable to have the wooden plank cut with a deviation of +/- 1 mm, representing the acceptance criterion – to have the wooden plank cut between 499 mm and 501 mm. These two attributes are related to production quality and are currently represented as string attributes. Our meta-model, especially the part on production quality, can be extended further and is considered a future work.

Besides process steps, gates (*Gate*) can be used as elements of a control flow. Gates of different types (*type*) are needed to specify entry and exit points of (*EGateType*):

- decision branches (*DECISION*) – to decide which structured set of process elements to execute based on a certain condition;
- iteration branches (*ITERATION*) – to repeat the execution of a structured set of process elements a certain number of times or until a certain condition is met;
- parallelism branches (*PARALLELISM*) – to execute structured sets of process elements in parallel;
- collaboration branches (*COLLABORATION*) – to execute structured sets of process elements in parallel, but there are process steps that must not start or finish their activities before they get a message that other process steps finished their activities; and



- variation branches (*VARIATION*) – the same intermediate or finished product can be produced by executing different structured sets of process elements, representing process variations, or different intermediate or finished products can be produced by executing different structured sets of process elements, representing product variations of the same product family.

Currently, the collaboration flow is based on a message exchange between process steps or resources and is modeled at a high level of abstraction using MultiProLan. With the collaboration modeling concept, it is possible to synchronize the actions of different resources. Whenever there is a collaboration between human workers and machines, safety aspects should also be considered, but this is part of our future work – to create an additional modeling layer to support the modeling of safety aspects. Also, as part of our future work, more complex communication between resources needs to be available for modeling using MultiProLan.

A gate has two additional attributes, which values need to be specified. Gates come in pairs, having the same pair identifier (*pairId*) and the same type. In addition, gates in the pair have different purposes regarding input and output branches (*EGateFlowType*). One gate is diverging (*DIVERGING*), meaning branches are going from the gate, and another one is converging (*CONVERGING*), meaning branches are going to the gate, distinguished by the *flow* attribute. These constraints are defined by using OCL and presented in Listing 7.5.

```

1  invariant gateMustHaveItsPair('Each gate must have only one paired gate with the
   same pairId.'):
   Gate.allInstances()->forAll(g | Gate.allInstances()->select(pairGate |
   g <> pairGate and g.pairId = pairGate.pairId)->size() = 1);
2  invariant pairGatesMustBeOfTheSameType('Gates in the pair must be of the same
   type.'):
   Gate.allInstances()->forAll(g1, g2 | (g1 <> g2 and g1.pairId = g2.pairId)
   implies (g1.type = g2.type));
3  invariant pairGatesMustHaveOppositeFlowPurposes('Gates in the pair must have
   opposite flow purposes.'):
   Gate.allInstances()->forAll(g1, g2 | (g1 <> g2 and g1.pairId = g2.pairId)
   implies (g1.flow <> g2.flow));

```

**Listing 7.5.** Constraints related to the pair of gates.

There are two constraints regarding diverging and converging gates, presented in Listing 7.6:

- each diverging gate that is not part of the error handling must have a single input flow-type relationship and at least two output flow-type relationships (diverging gates that are part of the error handling use error-type relationships, as discussed in Section 7.2.3); and
- each converging gate that is not part of the error handling must have at least two input flow-type relationships and a single output flow-type relationship.

```

1  invariant divergingGateRelationships('A diverging gate must have a single input
   and at least two output flow-type relationships.'):
   Gate.allInstances()->forAll(g | (not(g.isErrorGroup) and
   g.flow = EGateFlowType::DIVERGING) implies
   (g.inRelationships->select(r | r.type = ERelationshipType::FLOW)->size() = 1
   and g.outRelationships->select(r | r.type = ERelationshipType::FLOW)->
   size() >= 2));
2  invariant convergingGateRelationships('A converging gate must have at least two
   input and a single output flow-type relationships.'):
   Gate.allInstances()->forAll(g | (not(g.isErrorGroup) and
   g.flow = EGateFlowType::CONVERGING) implies
   (g.inRelationships->select(r | r.type = ERelationshipType::FLOW)->size() >= 2
   and g.outRelationships->select(r | r.type = ERelationshipType::FLOW)->
   size() = 1));

```

**Listing 7.6.** Constraints related to diverging and converging gates.

There is also an additional constraint regarding the gate's type. Whenever a diverging decision or a diverging iteration gate is created, all its output relationships must have the *logicalCondition* attribute specified, as selection and iteration patterns require a logical condition to be specified for each output branch. Such a constraint is presented in Listing 7.7.

1	<pre><b>invariant</b> selectionAndIterationPatternsMustHaveLogicalCondition('A logical condition must be specified for each output flow-type relationship of diverging decision and iteration gates.'):   Gate.allInstances()-&gt;forAll(g   (<b>not</b>(g.isErrorGroup) <b>and</b>     (g.type = EGateType::DECISION <b>or</b> g.type = EGateType::ITERATION) <b>and</b>     g.flow = EGateFlowType::DIVERGING) <b>implies</b> (g.outRelationships-&gt;forAll(r       (r.type = ERelationshipType::FLOW) <b>implies</b> (r.LogicalCondition &lt;&gt; <b>null</b> <b>and</b> r.LogicalCondition &lt;&gt; '')))));</pre>
---	---

**Listing 7.7.** A constraint related to selection and iteration patterns.

As stated before, variation branches and gates are used to represent different process or product variations. Therefore, each process has at least one variation (*ProcessVariation*), having semantic identifier (*semanticId*) and description (*description*) as attributes. Process variations are hierarchically organized; each variation has at most one parent variation (*superVariation*) and may have many child variations (*subVariations*). Each process element is assigned to at least one variation of the process it belongs to but may also be assigned to many variations, as a process element may appear in different variations. The constraint that restricts process element variations to only refer to variations of the parent process is implemented by using OCL inside the *Process* class and presented in Listing 7.8. Therefore, such a constraint forbids process element variations to refer to variations from other processes. Similar constraints are defined for other classes (e.g., the *startStep* relation from the *Process* class cannot reference the start step from another process) but are not presented in this thesis as they all have the same structure. A process has a default variation chosen from variations it has, which can help process designers speed up the process elements modeling by assigning the default variation to the elements automatically.

1	<pre><b>invariant</b> elementVariationMustReferToParentProcess('Each element variation must refer to a variation of the parent process.'):   self.elements-&gt;forAll(e   e.variations-&gt;forAll(v     self.variations-&gt;exists(processVariation   v = processVariation)));</pre>
---	--

**Listing 7.8.** A constraint related to a process element variation.

The unordered steps element (*UnorderedSteps*) contains different process steps that can be executed in any order. Each unordered set of steps that is not part of error handling must contain at least two regular process steps that are not part of the error handling elements. Process steps in an unordered set of steps cannot be connected with any relationships. These constraints are implemented by using OCL and presented in Listing 7.9.

1	<pre><b>invariant</b> unorderedSetMustContainAtLeastTwoProcessSteps('An unordered set of steps must contain at least two process steps.'):   UnorderedSteps.allInstances()-&gt;forAll(us   us.steps-&gt;size() &gt;= 2);</pre>
2	<pre><b>invariant</b> unorderedSetMustContainOnlyRegularProcessSteps('An unordered set of steps must contain only regular process steps that are not from the error group.'):   UnorderedSteps.allInstances()-&gt;forAll (us   (<b>not</b>(us.isErrorGroup)) <b>implies</b>     (us.steps-&gt;forAll(ps   <b>not</b>(ps.isErrorGroup) <b>and</b>     ps.type = EProcessStepType::REGULAR)));</pre>
3	<pre><b>invariant</b> unorderedSetCannotContainRelationships('There are no relationships in unordered steps.'):   UnorderedSteps.allInstances()-&gt;forAll(us   us.steps-&gt;forAll(ps     ps.inRelationships-&gt;isEmpty() <b>and</b> ps.outRelationships-&gt;isEmpty()));</pre>

**Listing 7.9.** Constraints related to an unordered set of steps.



There is also a sub-process (*SubProcess*) – a process element that references another process in order to:

- make process models more readable and less complex;
- increase the reusability of processes; and
- decrease the redundancy of process elements.

The material flow must be included in sub-processes, meaning that a sub-process can have parameters (*ProcessParameter*) of different types (*type*), having one of the following values (*EProcessParameterType*):

- *INPUT\_PARAMETER* – represents input parameters, i.e., materials or products that are coming to the sub-process; and
- *OUTPUT\_PARAMETER* – represents output parameters, i.e., materials or products that are coming from the sub-process as a result.

Sub-process parameters can reference products or other parameters, as sub-processes can be part of a larger control flow, just like any process element can. A process also contains parameters, which are referenced from sub-process parameters, and they are connected to other parameters or products in the process. As a sub-process may be the final process element in a process, its output parameter can represent the final product (*isFinal*) of the process.

Finally, most of the presented classes inherit the *IDNamedElement* class comprising identifier (*id*) and name (*name*) attributes.

## 7.2.2 Detail-Level Modeling Concepts at Execution Layer

The DetL part of the MultiProLan meta-model used at Execution Layer is depicted in Figure 7.3. This part of the meta-model is an extension of the MasL part, and together they are used to create DetL models.

As DetL process models include transportation and configuration activities, process step notations are extended by:

- *TRANSPORTATION* – production logistic activities of transporting materials or resources; and

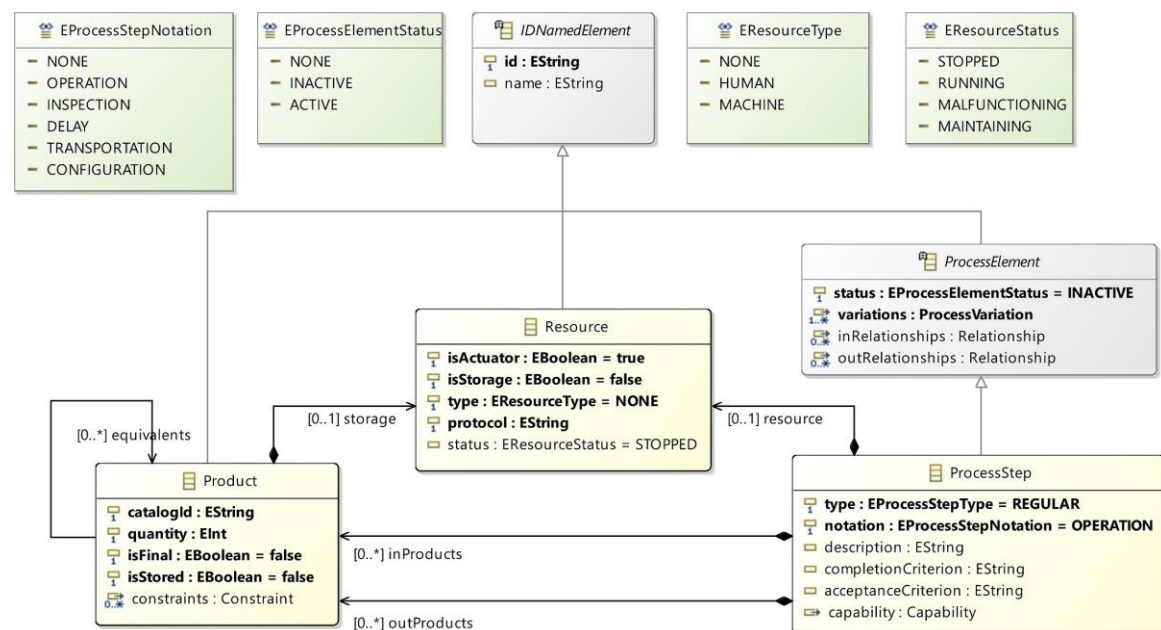


Figure 7.3. The second part of the meta-model used for DetL model creation at Execution Layer.

- *CONFIGURATION* – activities to configure resources, such as changing a griper or determining the robot's position.

A process step is extended with a resource (*resource*) that is to execute it using a required capability. A product is also extended with specific storage (*storage*) that needs to be defined for every input product retrieved from storage and for every output product placed in storage. Storage also represents a resource, as it can be equipped with sensors to monitor materials and products and with various devices.

A resource (*Resource*) inherits the *IDNamedElement* class, and it can be an actuator (*isActuator*) – an active resource, i.e., one that performs different activities during production, or storage (*isStorage*) – a passive resource, i.e., one that stores products. A resource can be both an actuator and storage; e.g., some robots can execute different tasks but also have a place on themselves to temporarily store products. Additional constraints must be defined for resources and they are implemented using OCL as presented in Listing 7.10:

- a process step can reference (*resource*) only a resource that is an actuator;
- a product can reference (*storage*) only a resource that is storage; and
- start and end process steps cannot reference a resource.

1	<pre>invariant processStepMustReferToActuator('A process step must refer to a resource that is an actuator.'):     ProcessStep.allInstances()-&gt;forAll(ps   (ps.resource &lt;&gt; null) implies     (ps.resource.isActuator));</pre>
2	<pre>invariant productMustReferToStorage('A product must refer to a resource that is storage.'):     Product.allInstances()-&gt;forAll(p   (p.storage &lt;&gt; null) implies     (p.storage.isStorage));</pre>
3	<pre>invariant startProcessStepCannotContainResource('The start process step cannot contain a resource.'):     ProcessStep.allInstances()-&gt;forAll(ps   (ps.type = EProcessStepType::START)     implies (ps.resource = null));</pre>
4	<pre>invariant endProcessStepCannotContainResource('The end process step cannot contain a resource.'):     ProcessStep.allInstances()-&gt;forAll(ps   (ps.type = EProcessStepType::END)     implies (ps.resource = null));</pre>

**Listing 7.10.** Constraints related to resources and storage.

Resources can also be classified (*type*) with the following values (*EResourceType*):

- *HUMAN* – a human worker;
- *MACHINE* – a machine or a robot; and
- *NONE* – a resource is neither a human nor a machine, e.g., a regular storage shelf with no smart devices or sensors attached.

Whenever a resource type is *NONE*, the resource must represent storage and it is not an actuator. Such a constraint is implemented and presented in Listing 7.11.

1	<pre>invariant noneTypeResourceMustBeRegularStorage('A resource of the NONE type must be storage and not an actuator.'):     Resource.allInstances()-&gt;forAll(r   (r.type = EResourceType::NONE) implies     (r.isStorage and not(r.isActuator)));</pre>
---	--

**Listing 7.11.** A constraint related to regular storage.

A protocol (*protocol*) must be specified for each resource, which is important information, especially during instruction generation, as instructions will be sent to protocol transformation components. Depending on the resource type and protocol, human-readable or machine-specific instructions will be generated for every process step. A resource's status (*status*) needs to be known before and during the process execution, and it can have the following values (*EResourceStatus*):

- *RUNNING* – a resource is running and ready for process steps execution;
- *STOPPED* – a resource has stopped working;
- *MALFUNCTIONING* – a resource is malfunctioning and cannot be used before it is repaired; and
- *MAINTAINING* – a resource is currently maintained.

A process element also has an execution status (*status*) through which process monitoring is implemented, and it can have the following values (*EProcessElementStatus*):

- *NONE* – a process element is not yet executed;
- *ACTIVE* – a process element is currently being executed; and
- *INACTIVE* – a process element has been successfully executed.

When extended with active and passive resources, production logistics, and configuration activities, process models are ready for the automatic generation of executable instructions.

### 7.2.3 Modeling Concepts at Error Handling Layer

In Figure 7.4, the third part of the meta-model is presented, representing modeling concepts needed for error handling at Error Handling Layer. This additional layer to production process models is used both at MasL and DetL. The central class of this part of the meta-model is an error (*Error*), which inherits the *IDNamedElement* class, and a process step can contain many errors related to it. However, the start and end process steps cannot contain errors, and such constraints are implemented using OCL and presented in Listing 7.12.

```

1  invariant startProcessStepCannotContainErrors('The start process step cannot
   contain errors.'):
   ProcessStep.allInstances()->forall(ps | (ps.type = EProcessStepType::START)
   implies (ps.errors->isEmpty()));
2  invariant endProcessStepCannotContainErrors('The end process step cannot contain
   errors.'):
   ProcessStep.allInstances()->forall(ps | (ps.type = EProcessStepType::END)
   implies (ps.errors->isEmpty()));
    
```

Listing 7.12. Constraints related to the start and end process step errors.

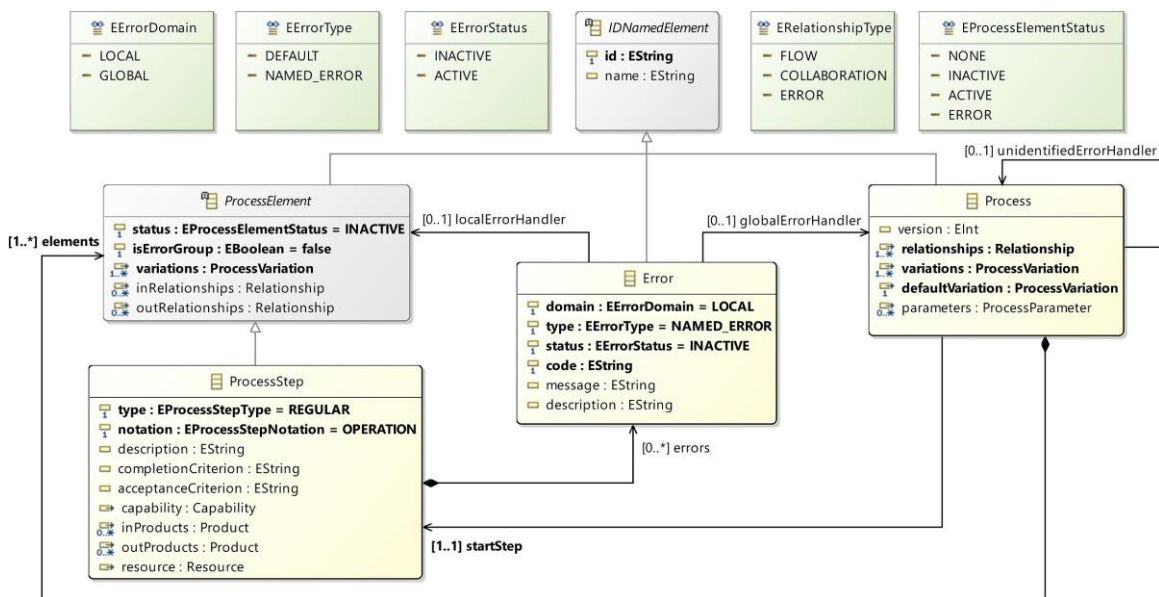


Figure 7.4. The third part of the meta-model used for the error handling modeling at Error Handling Layer.

An error has its domain (*domain*), which can be (*EErrorDomain*):

- *LOCAL* – representing a local error that has the specific handler for the process in which it occurs; and
- *GLOBAL* – representing a global error that can occur in multiple process steps of the same or other processes.

Depending on the selected domain, an error can reference a process element (*localErrorHandler*) – starts a control flow of process elements that handle the error, or an error can reference another process (*globalErrorHandler*) – a process that is already modeled and represents a handler for multiple errors. The constraints determining *localErrorHandler* or *globalErrorHandler* to be specified if the *LOCAL* or *GLOBAL* domain is chosen are implemented using OCL and presented in Listing 7.13.

1	<pre><b>invariant</b> localErrorHandlerDependsOnDomain('A local error handler must be specified for the local domain.'):     Error.allInstances()-&gt;forAll(e   (e.domain = EErrorDomain::LOCAL) <b>implies</b>     (e.localErrorHandler &lt;&gt; <b>null</b> and e.globalErrorHandler = <b>null</b>));</pre>
2	<pre><b>invariant</b> globalErrorHandlerDependsOnDomain('A global error handler must be specified for the global domain.'):     Error.allInstances()-&gt;forAll(e   (e.domain = EErrorDomain::GLOBAL) <b>implies</b>     (e.globalErrorHandler &lt;&gt; <b>null</b> and e.localErrorHandler = <b>null</b>));</pre>

**Listing 7.13.** Constraints related to local and global error handlers.

An error also has a type (*type*), and it can be (*EErrorType*):

- *NAMED\_ERROR* – an error that is known to process designers and can be expected; and
- *DEFAULT* – an unknown error with a default error handler.

Only a single default error handler can be specified per process element, and such a constraint is implemented using OCL and presented in Listing 7.14. If a default error handler is not specified for a process element, a default error handler at the process level (*unidentifiedErrorHandler*) will be used.

1	<pre><b>invariant</b> singleDefaultErrorHandlerPerStepCanExist('Only a single default error handler per process step can exist.'):     ProcessStep.allInstances()-&gt;forAll(ps   ps.errors-&gt;select(e       e.type = EErrorType::DEFAULT)-&gt;size() &lt;= 1);</pre>
---	---

**Listing 7.14.** A constraint related to a default error handler.

There is also an error status (*status*) during process execution, having one of the following values (*EErrorStatus*):

- *INACTIVE* – an error has not occurred yet; and
- *ACTIVE* – an error has occurred and corrective steps are going to be applied.

Also, a process element has a new status value added (*ERROR*), and this status is active whenever an error occurs at the process element during the execution. To distinguish default process elements from process elements specified for local error handlers, there is an indicator (*isErrorGroup*) for each process element. Similar to the constraint related to the unordered set of steps that is not from the error group (discussed in Section 7.2.1), each unordered set of steps that is from the error group must contain regular process steps that are from the error group. Such a constraint is implemented using OCL and presented in Listing 7.15.

1	<pre> <b>invariant</b> errorUnorderedSetMustContainOnlyRegularErrorProcessSteps('An unordered set of steps from the error group must contain only regular process steps that are from the error group.'):   UnorderedSteps.allInstances()-&gt;forALL(us   (us.isErrorGroup) <b>implies</b> (us.steps-&gt;forALL(ps   ps.isErrorGroup <b>and</b> ps.type = EProcessStepType::REGULAR))); </pre>
---	--

**Listing 7.15.** A constraint related to an unordered set of error steps.

An additional relationship type (*ERROR*) is added to connect error process elements. There are also constraints related to error-type and flow-type relationships, presented in Listing 7.16:

- there cannot be two error-type relationships that have the same source and target elements;
- an error-type relationship source element must be from the error group, but a target element does not have to be from the error group (e.g., after some corrective steps, an error flow returns to the main process steps);
- each error-type relationship cannot have the *message* attribute specified; and
- a flow-type relationship source and target elements cannot be from the error group.

1	<pre> <b>invariant</b> singleErrorRelationshipBetweenElements('There cannot be more than a single error-type relationship between the same source and target elements.'):   Relationship.allInstances()-&gt;forALL(r1, r2   ((r1 &lt;&gt; r2) <b>and</b> (r1.type = ERelationshipType::ERROR <b>and</b> r2.type = ERelationshipType::ERROR)) <b>implies</b> ((r1.source &lt;&gt; r2.source) <b>or</b> (r1.target &lt;&gt; r2.target)); </pre>
2	<pre> <b>invariant</b> errorRelationshipSourceMustBeErrorElement('A source of an error-type relationship must be an element from the error group.'):   Relationship.allInstances()-&gt;forALL(r   (r.type = ERelationshipType::ERROR) <b>implies</b> (r.source.isErrorGroup)); </pre>
3	<pre> <b>invariant</b> errorRelationshipCannotHaveMessage('The message attribute cannot be specified for an error-type relationship.'):   Relationship.allInstances()-&gt;forALL(r   r.type = (ERelationshipType::ERROR) <b>implies</b> (r.message = null <b>or</b> r.message = '')); </pre>
4	<pre> <b>invariant</b> flowRelationshipCannotConnectErrorElements('A flow-type relationship cannot connect elements from the error group.'):   Relationship.allInstances()-&gt;forALL(r   (r.type = ERelationshipType::FLOW) <b>implies</b> (<b>not</b>(r.source.isErrorGroup <b>or</b> r.target.isErrorGroup))); </pre>

**Listing 7.16.** Constraints related to error-type and flow-type relationships.

Gates from the error group have similar constraints to the gates that are not from the error group (discussed in Section 7.2.1), and they are presented in Listing 7.17:

- Each diverging gate from the error group must have at least two output error-type relationships. There is no additional constraint for input relationships as a diverging gate from the error group can have no input relationships whenever it is only referenced from an error, or it can have multiple input relationships if some corrective steps are going back to the gate to repeat some steps.
- Each converging gate from the error group must have at least two input error-type relationships and a single output error-type relationship.
- Output error-type relationships from a diverging decision or a diverging iteration gate must have the *logicalCondition* attribute specified.

1	<pre> <b>invariant</b> errorDivergingGateRelationships('A diverging gate from the error group must have at least two output error-type relationships.'):   Gate.allInstances()-&gt;forALL(g   (g.isErrorGroup <b>and</b> g.flow = EGateFlowType::DIVERGING) <b>implies</b> (g.outRelationships-&gt;select(r   r.type = ERelationshipType::ERROR)-&gt;size() &gt;= 2)); </pre>
2	<pre> <b>invariant</b> errorConvergingGateRelationships('A converging gate from the error group must have at least two input and a single output error-type relationships.'):   Gate.allInstances()-&gt;forALL(g   (g.isErrorGroup <b>and</b> g.flow = EGateFlowType::CONVERGING) <b>implies</b> (g.inRelationships-&gt;select(r   </pre>



3	<pre> r.type = ERelationshipType::ERROR)-&gt;size() &gt;= 2 and g.outRelationships-&gt;select(r   r.type = ERelationshipType::ERROR)-&gt; size() = 1)); <b>invariant</b> errorSelectionAndIterationPatternsMustHaveLogicalCondition('A logical condition must be specified for each output error-type relationship of diverging decision and iteration gates from the error group.'): Gate.allInstances()-&gt;forALL(g   (g.isErrorGroup and (g.type = EGateType::DECISION or g.type = EGateType::ITERATION) and g.flow = EGateFlowType::DIVERGING) <b>implies</b> (g.outRelationships-&gt;forALL(r   (r.type = ERelationshipType::ERROR) <b>implies</b> (r.LogicalCondition &lt;&gt; null and r.LogicalCondition &lt;&gt; '))))); </pre>
---	---

**Listing 7.17.** Constraints related to gates from the error group.

Each error has a code – an identifier of an error type it belongs to (*code*), a message (*message*), and a description (*description*) that will be presented to supervisors who monitor process execution after the error occurs.

## 7.2.4 Modeling Concepts for the Automatic Generation of Manufacturing Documentation and Guided Production

In Figure 7.5, the fourth and the last part of the meta-model is presented, representing modeling concepts or meta-model extensions needed for the automatic generation of manufacturing documentation of different types and for guided production.

The *Process* class is extended with the following attributes, which are used in most generated documents:

- *company* – a name of a company in which a process model is created;
- *author* – a name of an author or authors who created a process model;
- *date* – a date when a process model is created; and
- *isProposedProcess* – an indicator of whether a process model is a proposal or is already finished and approved by stakeholders.

The *Process* class is also extended with image (*image*) and video (*video*) links related to the whole process of producing the final product. These attributes are needed for guided production to export images and videos and send them to human workers. The *ProcessStep*, *UnorderedSteps*, and *SubProcess* classes are also extended with the image (*image*) and video (*video*) attributes for the same purpose, but also for the worker-manual-like documentation, such as Job Breakdown Sheets (JBSs) [238].

Expert workers or instructors use JBS to train new workers to create products. Therefore, each process step needs to be described, and images need to be included to visually present what needs to be done. Also, each process step can include key points (*KeyPoint*), indicating the details workers need to pay attention to while performing process steps. However, the start and end process steps cannot contain key points. These constraints are implemented using OCL and presented in Listing 7.18.

1	<pre> <b>invariant</b> startProcessStepCannotContainKeyPoints('The start process step cannot contain key points.'): ProcessStep.allInstances()-&gt;forALL(ps   (ps.type = EProcessStepType::START) <b>implies</b> (ps.keypoints-&gt;isEmpty())); </pre>
2	<pre> <b>invariant</b> endProcessStepCannotContainKeyPoints('The end process step cannot contain key points.'): ProcessStep.allInstances()-&gt;forALL(ps   (ps.type = EProcessStepType::END) <b>implies</b> (ps.keypoints-&gt;isEmpty())); </pre>

**Listing 7.18.** Constraints related to the start and end process steps and key points.

A key point has a type (*type*), including one of the following values (*EKeyPointType*):

- *COMPLETE\_JOB* – how to perform a process step to complete the task;
- *COMPLETE\_JOB\_EASIER* – how to perform a process step to complete it easier;
- *BREAK\_JOB* – what and how not to do as it could compromise the successful finishing of the task; and
- *INJURE\_WORKER* – what could cause an injury to a worker.

The *KeyPoint* class inherits the *IDNamedElement* class, and each key point has a reason (*reason*) to be considered during a process step execution. Key points can also be used in guided production to help workers during production.

The *ProcessStep* class is extended with time (*time*) and work center (*workCenter*) attributes. Estimation of the process step's duration and the name of the work center in which the process step is to be performed are needed by documents, such as BOMO.

To automatically generate PFMEA spreadsheets, the *Error* class needs to be extended with the following attributes [28]:

- *mode* – a potential failure mode representing a manner in which process step requirements may potentially fail to be met;
- *effect* – a potential effect of a failure on the product, customer, worker, machine, or the following operations;

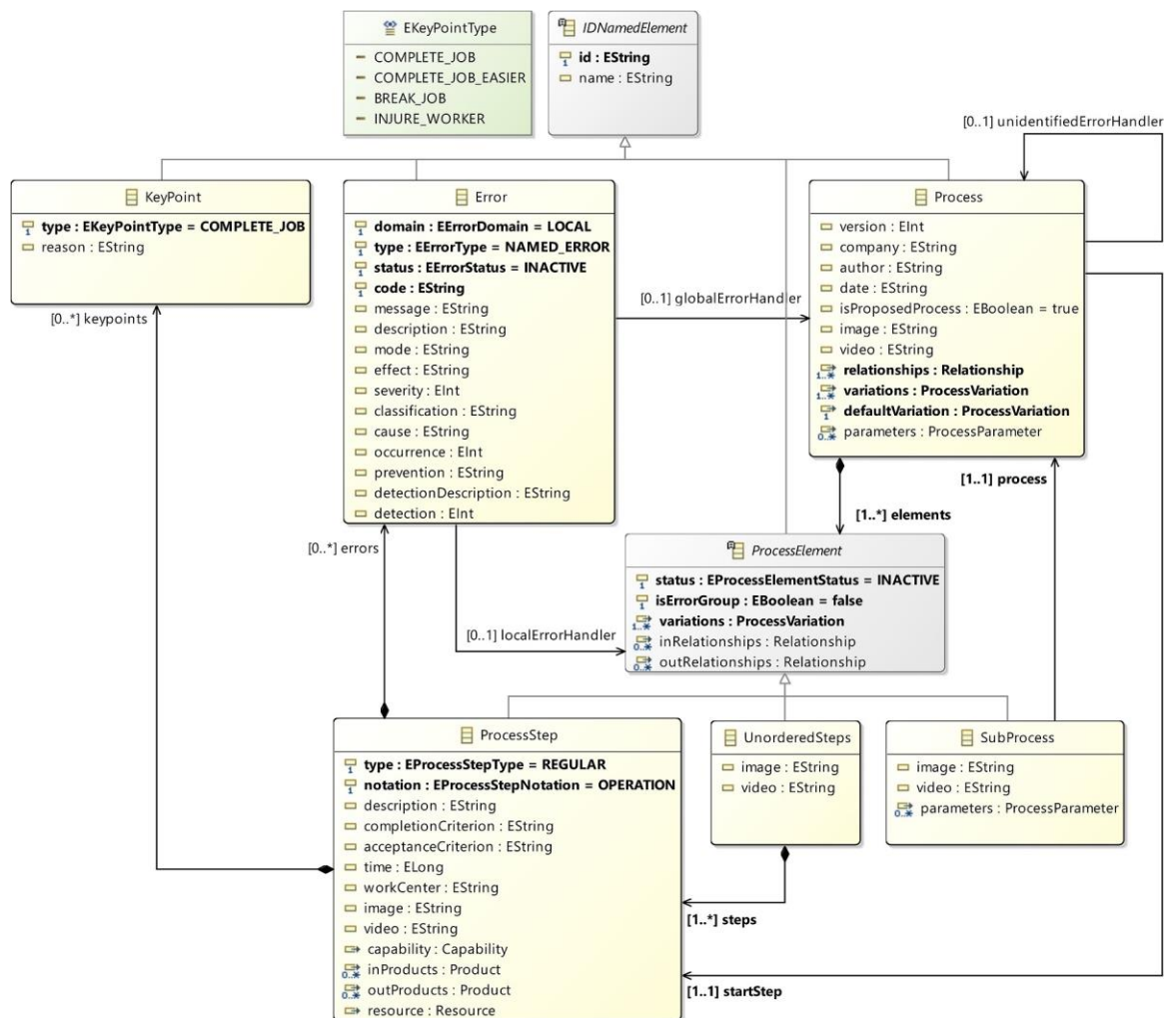


Figure 7.5. The fourth part of the meta-model used for the automatic generation of manufacturing documentation and guided production.

- *severity* – represents a relative ranking of how serious the effect is, usually ranging from 1 to 10;
- *classification* – used to classify any product or process characteristic;
- *cause* – a potential cause of a failure represents how the failure can occur;
- *occurrence* – represents a relative ranking of the likelihood of a failure to occur, usually ranging from 1 to 10;
- *prevention* – a description of control to prevent a failure from occurring;
- *detectionDescription* – a description of how to detect a failure; and
- *detection* – represents a relative ranking of a failure to be detected, usually ranging from 1 to 10.





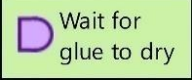







In the following section, the concrete graphical syntax of MultiProLan is presented, which is created based on the abstract syntax presented in this section.

### 7.3 Concrete Syntax of MultiProLan

There are two types of concrete syntaxes – textual and graphical, but there is no general answer to which one is better overall [239]. We decided to create a graphical syntax for MultiProLan to make the modeling easier for process designers as they are already familiar with other graphical languages, such as ASME FPC [25]. The decision was also made to enable visualized process monitoring, as well as to enable visualization of detected errors during production. As BPMN [30] is commonly used to model processes of different kinds, and as it is easy to interpret its models [240], some BPMN concepts, such as activities and gateways, are used in the graphical syntax of MultiProLan. The graphical syntax is also inspired by ASME FPC as process designers are used to these charts. Some FPC elements are used in process step notations, such as operation, transportation, inspection, and delay. Also, the storage element is used within a product, indicating that a product should be retrieved from storage or placed in storage.





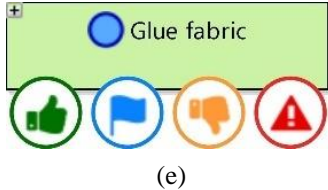
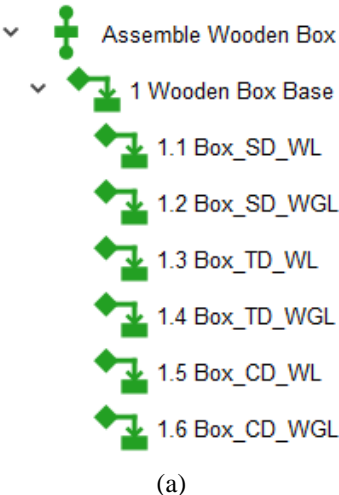

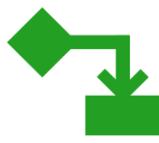







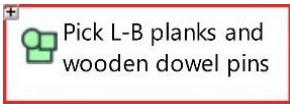
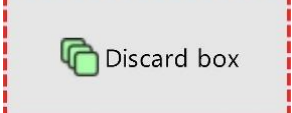
The graphical symbols used for the MultiProLan's concrete syntax are presented and described in Table 7.1, starting with MasL modeling concepts at Execution Layer and Error Handling Layer and ending with DetL modeling concepts at both layers. All the MasL modeling concepts are used for DetL modeling as well.


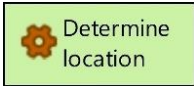





**Table 7.1.** The basic modeling concepts of MultiProLan.

Description	Symbol				
<b>Master-Level modeling concepts at Execution Layer</b>					
The <i>Start</i> (a) and <i>End</i> (b) process steps indicate the starting and finishing points of the process execution.	 (a)		 (b)		
Regular process steps can be an operation (a), inspection (b), or delay (c). These process step notations indicate to process designers what kinds of activities need execution.	 (a)	 (b)	 (c)		
Gates are used as a control flow mechanism, forming a decision (a), iteration (b), parallelism (c), collaboration (d), or product/process variations (e).	 (a)	 (b)	 (c)	 (d)	 (e)
Relationships are used to form a control flow (a) or a collaboration (b).	 (a)		 (b)		



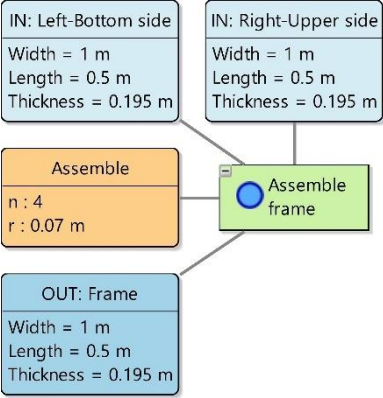
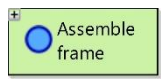
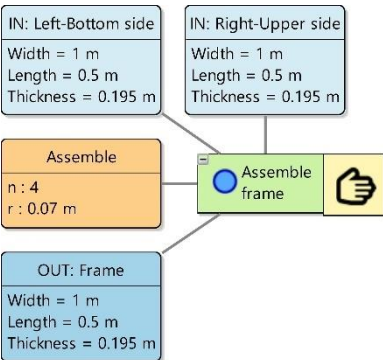
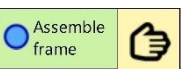
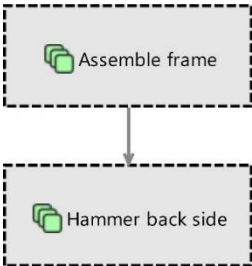
<p>A capability is specified for each regular process step. Different parameters can be specified as key-value properties. A key represents a parameter's name.</p>	<div style="border: 1px solid black; padding: 5px; text-align: center;"> <p style="background-color: #fde9d9; display: inline-block; padding: 2px 10px;">Assemble</p> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 5px;"> <p>n : 2 r : 0.07 m</p> </div>	
<p>An input product represents a material, a part, or an intermediate product. It can be an output from a previous process step (a) or be retrieved from storage, which is indicated by a triangle icon (b). Different constraints can be specified for an input product.</p>	<div style="border: 1px solid black; padding: 5px; text-align: center;"> <p>IN: Wooden lid</p> <hr/> <p>Width = 1.06 m Length = 0.56 m Thickness = 0.03 m</p> </div> <p style="text-align: center;">(a)</p>	<div style="border: 1px solid black; padding: 5px; text-align: center;"> <p> IN: Wooden lid</p> <hr/> <p>Width = 1.06 m Length = 0.56 m Thickness = 0.03 m</p> </div> <p style="text-align: center;">(b)</p>
<p>An output product represents an intermediate or finished product. It can be used as an input for a subsequent step (a) or stored in storage, which is indicated by a triangle icon (b). Different constraints can be specified for an output product.</p>	<div style="border: 1px solid black; padding: 5px; text-align: center;"> <p>OUT: Box</p> <hr/> <p>Width = 1.06 m Length = 0.56 m Thickness = 0.23 m</p> </div> <p style="text-align: center;">(a)</p>	<div style="border: 1px solid black; padding: 5px; text-align: center;"> <p> OUT: Box</p> <hr/> <p>Width = 1.06 m Length = 0.56 m Thickness = 0.23 m</p> </div> <p style="text-align: center;">(b)</p>
<p>A product equivalency link is used to represent equivalent products, i.e., when an input product of a process step is the same product that was present in a previous process step.</p>		
<p>An unordered set of steps includes process steps that can be executed in any order (a). Process steps can be hidden using a +/- button to reduce model complexity (b).</p>	<div style="border: 1px solid black; padding: 5px;"> <p> Pick L-B planks and wooden dowel pins</p> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> <p> Pick left side</p> </div> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> <p> Pick bottom side</p> </div> <div style="border: 1px solid black; padding: 2px; margin: 2px;"> <p> Pick wooden dowel pins</p> </div> </div> <p style="text-align: center;">(a)</p>	<div style="border: 1px solid black; padding: 5px;"> <p> Pick L-B planks and wooden dowel pins</p> </div> <p style="text-align: center;">(b)</p>
<p>A sub-process (a) is used to reference another process model. It is a mechanism for reducing the complexity of a model. A sub-process (b) can be equipped with input and output parameters.</p>	<div style="border: 1px dashed black; padding: 10px; text-align: center;"> <p> Discard box</p> </div> <p style="text-align: center;">(a)</p>	<div style="border: 1px dashed black; padding: 10px; text-align: center;"> <p> IN</p> <p> Discard box</p> <p> OUT</p> </div> <p style="text-align: center;">(b)</p>
<p>Input parameters (a) are used to send products to sub-processes, and output parameters (b) are used to receive products from sub-processes. They are also used in processes to connect with products or sub-process parameters.</p>	<div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; display: flex; align-items: center; justify-content: center; margin: 0 auto;"> <p style="margin: 0;">IN</p> </div> <p style="text-align: center;">(a)</p>	<div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; display: flex; align-items: center; justify-content: center; margin: 0 auto;"> <p style="margin: 0;">OUT</p> </div> <p style="text-align: center;">(b)</p>
<p>A parameter link connects a parameter with a product or with another parameter.</p>		

<p>Key points indicate to workers how to successfully perform a process step (a), how to perform a process step easier (b), what could make a process step fail (c), and what could injure workers while performing a process step (d). Key points can be added to a process step they are related to (e).</p>	 <p>(a)</p>	 <p>(b)</p>	 <p>(c)</p>	 <p>(d)</p>
<p>Product and process variations are hierarchically presented within a tree structure (a). The root node is a process model (b), having variations as sub-nodes (c). Each variation can have many sub-variations.</p>	 <p>(e)</p>			
<p>Production errors can be modeled within local error handlers with action steps as part of the current model (a) or within global error handlers as separate processes (b). Unidentified errors can also be modeled within local error handlers (c) or global error handlers (d).</p>	 <p>(a)</p>			
	 <p>(b)</p>	 <p>(c)</p>		
<p><b>Master-Level modeling concepts at Error Handling Layer</b></p>				
<p>Error relationships are used to link errors and error handler process elements.</p>	 <p>(a)</p>	 <p>(b)</p>	 <p>(c)</p>	 <p>(d)</p>
<p>Process elements that are part of a local error handler in MasL. These elements are from the error group, and they are marked with a red border.</p>				
				
				

Additional modeling concepts for Detail-Level at Execution Layer			
Additional types of process steps are added to support transportation (a) and configuration (b) activities.	 Move to smart shelf (a)	 Determine location (b)	
Active smart resources can be attached to process steps representing a human (a) or a machine/robot (b). Passive smart resources can be attached to products representing storage (c).	 (a)	 (b)	 (c)
Additional modeling concepts for Detail-Level at Error Handling Layer			
As new types of process steps are added in DetL, additional process step types are added to a local error handler as well.	 Move to smart shelf	 Determine location	

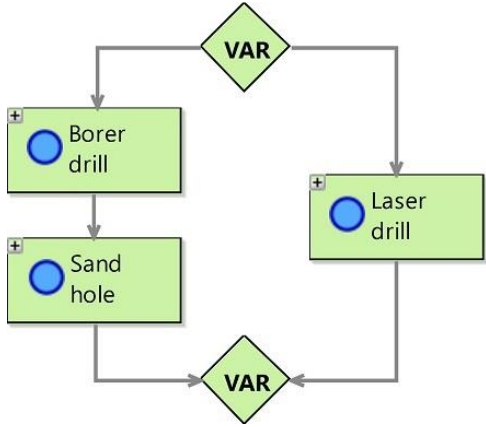

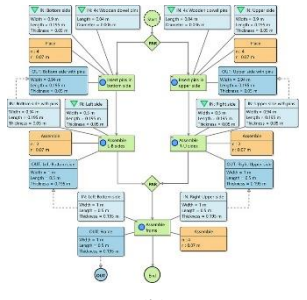
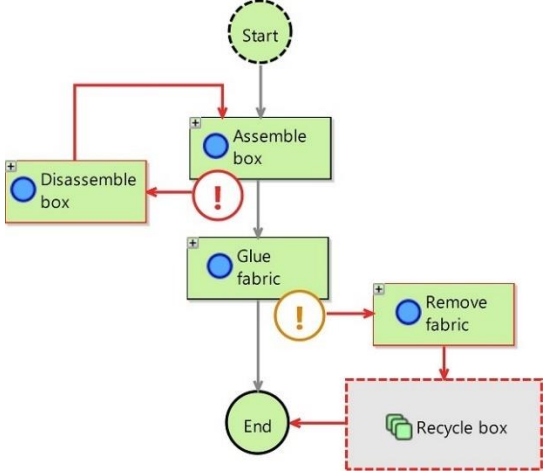
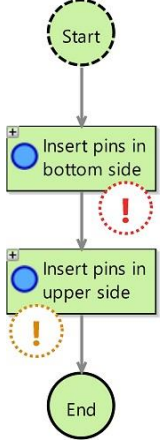
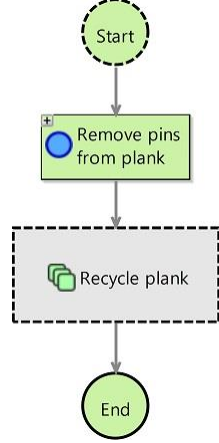
A classification of MultiProLan features according to already introduced production process modeling requirements in Section 4.2.1 is outlined in Table 7.2. Therefore, a way in which MultiProLan fulfills the requirements is presented in the table.

**Table 7.2.** MultiProLan requirements fulfilment.

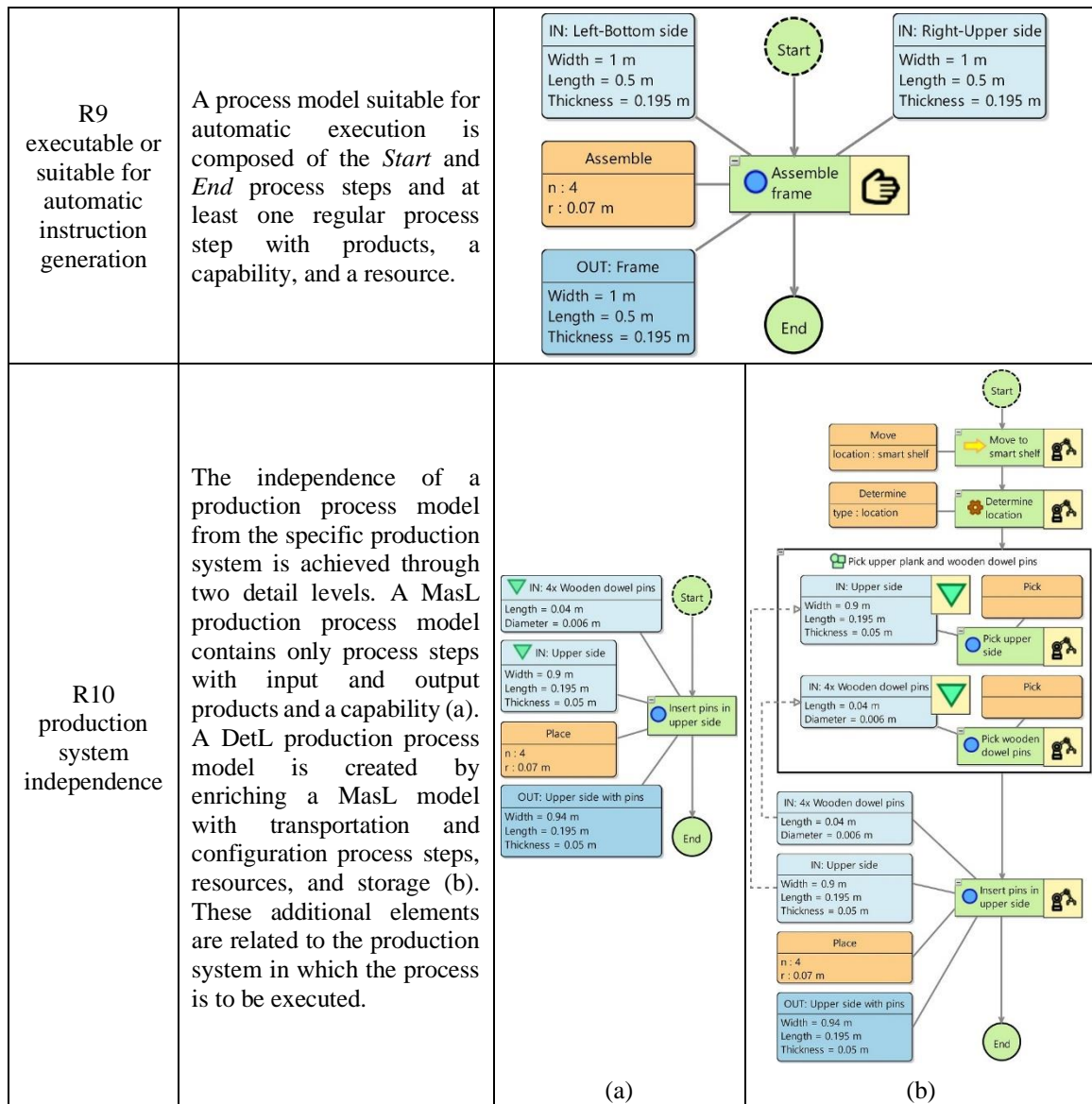
Requirement	Description	Figure	
R1 process step	A MasL process step contains input and output products and a capability (a). In addition to the MasL process step, a DetL process step also contains a smart resource (c). Products and a capability can be hidden by using a process step's +/- button (b, d) to make a process model diagram more readable.	 <p>(a)</p>	 <p>(b)</p>
		 <p>(c)</p>	 <p>(d)</p>
R2.1 sequence	A sequence is represented by two or more process elements connected with a flow-type relationship.		

<p>R2.2 decision</p>	<p>A decision is created between two decision gates with two or more conditional branches and process elements between them.</p>	
<p>R2.3 iteration</p>	<p>An iteration is created similarly to a decision, with one branch returning to the converging iteration gate.</p>	
<p>R2.4 parallelism</p>	<p>Parallelism is created between two parallelism gates with two or more parallel branches and process elements between them.</p>	
<p>R3 material flow</p>	<p>A material flow is represented by products retrieved from or placed in storage and product equivalency links, indicating that a product of one process step is the same product of another process step. In MasL models, particular storage is not specified but just an indicator that storage is needed (a), while in DetL models, storage is specified with all the necessary details (b).</p>	

<p>R4 message flow</p>	<p>A message flow is created between two collaboration gates with two or more branches and process steps between them. Process steps are also connected with collaboration-type relationships representing message exchange between them. For example, one process step cannot start before another one is started, or one process step cannot finish before another one is finished.</p>	
<p>R5 unordered steps</p>	<p>An unordered set of steps is created with an element containing different process steps that can be executed in any order.</p>	
<p>R6 product and process variations</p>	<p>Product and process variations are defined in a hierarchical structure (a). In a process model, variations are specified between two variation gates with two or more branches and process elements between them. Each process element in a process model references variations it belongs to from the previously defined tree structure. Therefore, each branch represents a different variation. There are two types of variations: product and process variations. Product variations (b) refer to differences in products from the same product family. For example, a wooden box can have a fully wooden lid or a wooden lid with a glass opening in the middle. Process variations (c) refer to different ways of</p>	<p>(a)</p> <p>(b)</p>

	<p>creating the same product. For example, a hole can be created by borer drilling and some sanding, or it can be made just by laser drilling.</p>	 <p>(c)</p>	
<p>R7 sub-process</p>	<p>A sub-process (a) is represented by a single process element containing a reference to another process model (b).</p>	 <p>(a)</p>	 <p>(b)</p>
<p>R8.1 local error handler</p>	<p>A local error handler is specified within a process model. There are handlers of known errors (at the <i>Assemble box</i> process step) and handlers of unknown errors (at the <i>Glue fabric</i> process step).</p>		
<p>R8.2 global error handler</p>	<p>A global error handler is a process model (b), that is referenced from another process model (a). There are handlers of known errors (at the <i>Insert pins in bottom side</i> process step) and handlers of unknown errors (at the <i>Insert pins in upper side</i> process step).</p>	 <p>(a)</p>	 <p>(b)</p>





As we created the graphical concrete syntax for MultiProLan, we had to create different mechanisms to reduce the diagram complexity and make it more readable when multiple modeling elements exist. At the level of language syntax, MultiProLan's complexity of model diagrams is addressed similarly to programming languages with a functional decomposition approach, e.g., by using sub-processes and global error handlers. At the modeling tool level, there are mechanisms to hide some parts of a diagram to simplify the view of the diagram and deal with the scalability problem. Such mechanisms are different modeling layers, +/- buttons within process steps and unordered sets of steps, the zoom feature, and element-type filtering tools.

The product family variability also needs to be addressed in production process modeling, especially in the context of Industry 4.0, but it increases the complexity of model diagrams. Variation gates with several branches between them are used to distinguish different product variations of the same product family. The sub-process modeling concept can address the variability of product families, as the same sub-process can be used in different product variations. Also, filtering tools can be used to choose a product variation to be presented and hide other variations from a diagram.

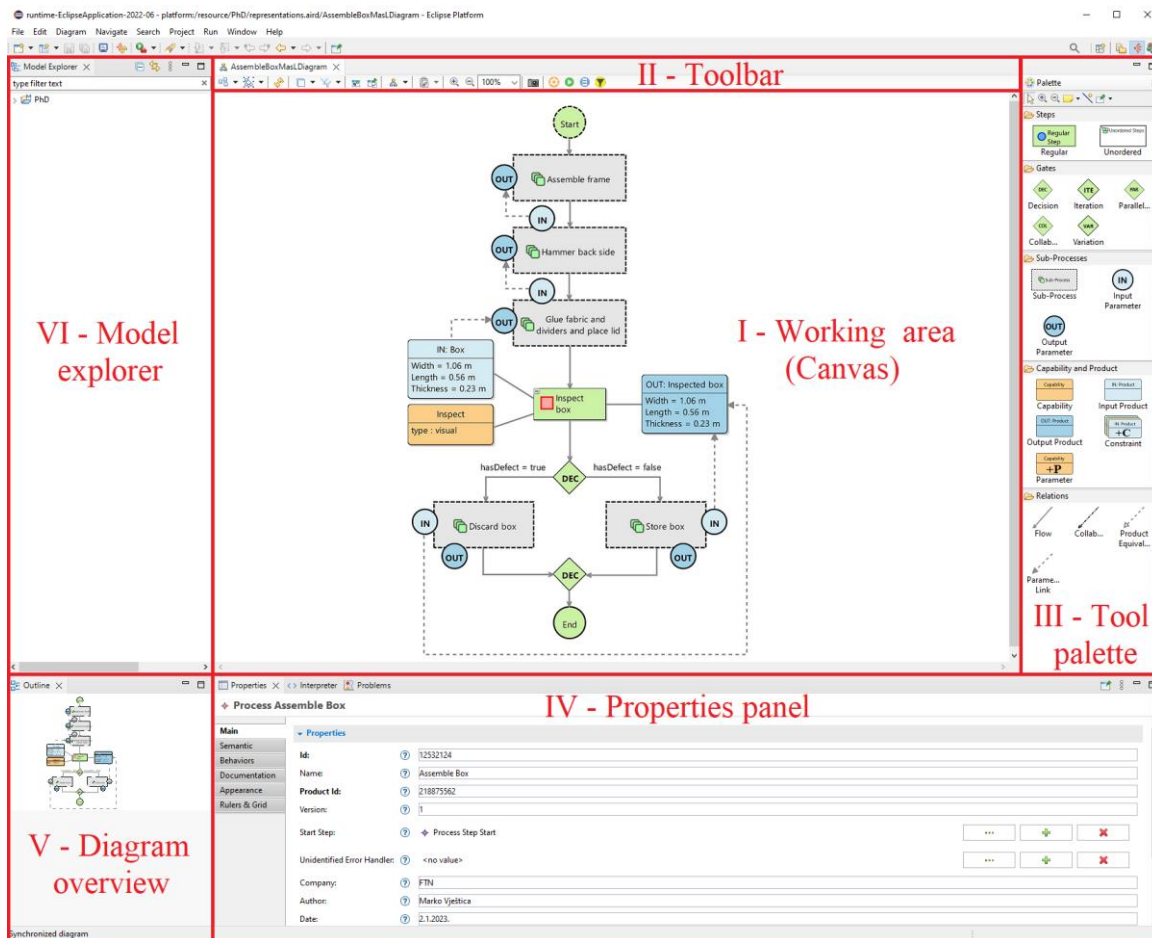
In the following section, we present an overview of Process Modeling Tool, which utilizes the MultiProLan language, and describe the tool's main components.

## 7.4 Process Modeling Tool

Process Modeling Tool, described in this section, is the one presented as a part of the architecture outlined in Figure 5.1. The tool utilizes MultiProLan and helps process designers model production processes. It was created in accordance with the principles of Rapid Software Development (RSD) [241] and by using the Eclipse Sirius framework. RSD aims to minimize the gap between software development and deployment by developing software in short rapid parallel cycles. It is built on agile practices, such as continuous integration and fast value delivery, making rapid and continuous experimentation on the software possible. As the requirements and development of MultiProLan and Process Modeling Tool were frequently changing during our research, due to the new and dynamic area of Industry 4.0, RSD fitted to such a changing and dynamic development process. In the rest of this section, we describe the main parts and features of Process Modeling Tool.

The user interface of Process Modeling Tool is presented in Figure 7.6. The tool is composed of the following:

- i) Working area – a canvas in which process designers create MultiProLan models;
- ii) Toolbar – an array of action buttons leading to various features;
- iii) Tool palette – a palette containing tools for the creation of various elements and relationships;
- iv) Properties panel – a panel with a changeable set of properties depending on the selected diagram, element type, or relationship type;
- v) Diagram overview – an area representing the whole diagram, even if the diagram is too large for Working area; and
- vi) Model explorer – a set of projects containing MultiProLan models.



**Figure 7.6.** The Process Modeling Tool user interface.



The Sirius framework automatically generates Working area; Toolbar with predefined action buttons discussed further in this section; empty Tool palette; Properties panel for each element and relationship type, as well as for the diagram; Diagram overview and Model explorer areas. Toolbar can be extended by adding new action buttons and Properties panel can be customized by changing which properties need to be shown or hidden or by changing their interface. Tool palette is the only tool part that needs to be fully implemented.

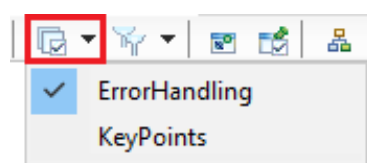
Toolbar has automatically generated action buttons, such as ones to arrange elements on a diagram, refresh a diagram, choose modeling layers, filter diagram elements, zoom in and out, and export a diagram as an image. In Figure 7.7, we present Process Modeling Tool Toolbar, while newly added or changed action buttons are framed. The action button that is modified is the modeling layers button, presented in Figure 7.8, with Error Handling Layer turned on. Execution Layer is the default one and is always visible, while Error Handling Layer and the layer with key points are optionally visible. The key points layer contains only the key point modeling concept and as a rather small layer, we considered it as a part of Execution Layer when mentioned in previous sections. It is created in Process Modeling Tool only to make a model diagram more readable. By turning on or off some modeling layer, a model diagram changes by showing or hiding elements related to the layer. Tool palette changes as well, which is discussed further in this section. There are four action buttons added to Toolbar:

- a) Orchestrate MasL process model – a MasL process model presented in Working area is sent to Orchestrator, after which it is automatically transformed into a DetL model. Both MasL and DetL models are stored in Knowledge Base. Afterward, high-level instructions are generated by Instruction Generator and sent to Digital Twin for execution.
- b) Execute DetL process model – a DetL process model presented in Working area is sent to Instruction Generator to automatically transform it into high-level instructions. These high-level instructions are then sent to Digital Twin for execution.
- c) Generate documentation – a MasL or DetL process model presented in Working area is automatically transformed into manufacturing documentation of various types. After the Generate documentation button is selected, a dialog is opened as presented in Figure 7.9. A user can choose various documentation to be generated, which product or process variation to use for the generation, and a location to save the generated documentation.
- d) Filter variations – a MasL process model presented in Working area is filtered based on the chosen product or process variation. A user can choose a product or process variation and choose whether this variation is to be used as the default one, meaning that each new element created on a diagram is going to belong to the chosen default variation. After a variation is chosen, elements and relationships belonging to other variations are hidden. Therefore, only the chosen variation and its super variations are presented to a user.

Tool palette is presented in Figure 7.10, and it consists of the following sections: (i) process steps; (ii) gates; (iii) sub-processes and parameters; (iv) capabilities and products, alongside parameters and constraints; (v) resources and storage (if a DetL process model is being created, otherwise this section is hidden); and (vi) relationships. Each tool has a tooltip, providing additional information to users on how to use the tool. There are also various shortcuts that help users model



**Figure 7.7.** Process Modeling Tool Toolbar.



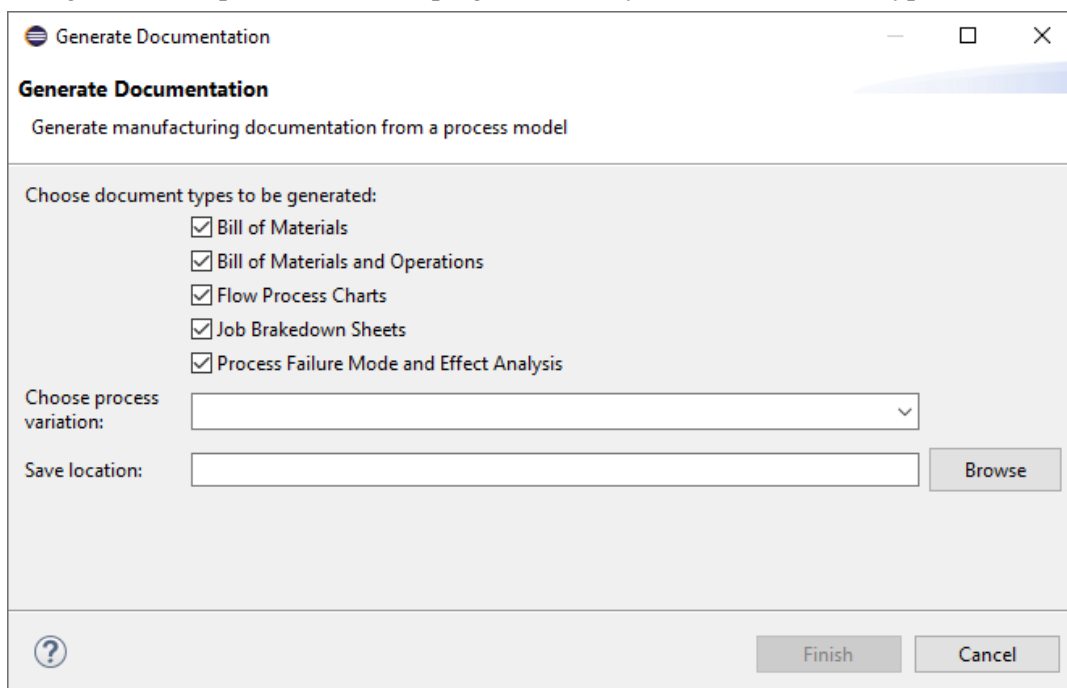
**Figure 7.8.** The Process Modeling Tool layer button.

production processes faster, such as when creating product equivalents, a product equivalent link can be used to connect a process step and a product of some previous process step, thus automatically creating an input product for the chosen step with same properties as the previous one. Another example is when creating capability parameters or product constraints which can be done by double clicking a capability or a product, after the parameter or constraint properties can be inserted by typing their values with a space as a delimiter for each property value.

Tool palette changes whenever a modeling layer is turned on or off. As presented in Figure 7.11, when Error Handling Layer is turned on, Tool palette is extended with the errors and error elements sections, while when the key points layer is turned on, Tool palette is extended with the key points section. The feature of extending or collapsing Tool palette based on the activation status of modeling layers helps process designers when modeling production processes, as tools not used by a particular user group are hidden from them, lowering the level of details in Process Modeling Tool. For example, process engineers only see basic Tool palette with key points optionally, while quality engineers see the error tools as well. Therefore, a particular user group is not burdened by the tools that the users do not use.

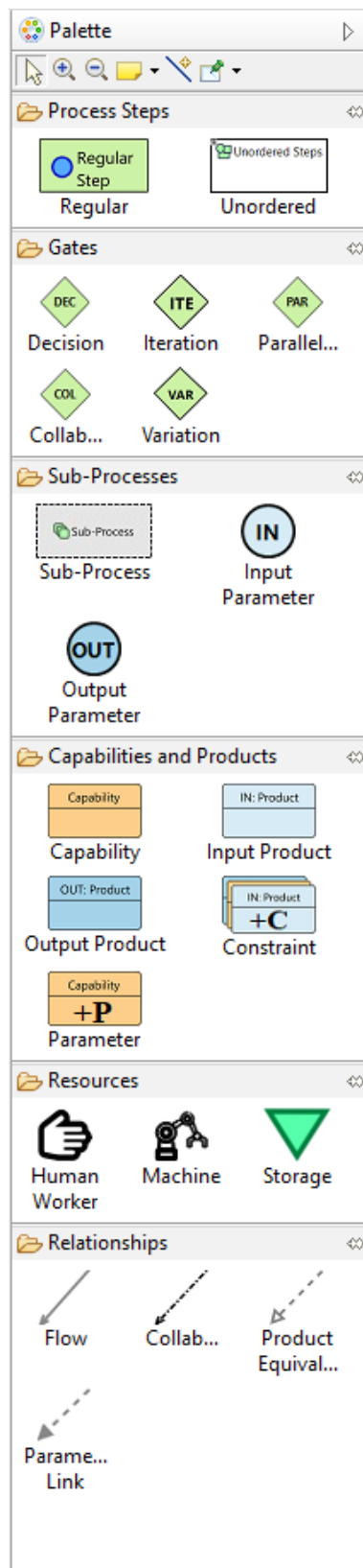
As process designers need to communicate with Knowledge Base to store new process models or retrieve the existing ones, Process Modeling Tool has import and export features as well. By choosing a process model located in the Model explorer area of Process Modeling Tool, a user can start the export feature. Afterward, a user can choose a server and a port of Orchestrator with Knowledge Base to which a process model should be sent. Then, a process model is automatically transformed into a JSON file, which is sent to the location of Orchestrator's Application Programming Interface (API) that waits for the process model in the JSON format. The Orchestrator's inner components automatically transform a JSON process model into a format appropriate for Vaticle TypeDB and programmatically store the process model in Knowledge Base permanently.

Production process models stored in Knowledge Base can be imported into Process Modeling Tool via the import feature. By choosing a modeling project located in the Model explorer area of Process Modeling Tool, a user can start the import feature. A user can choose a server and a port of Orchestrator with Knowledge Base from which a process model should be received. Afterward, a user can choose a process model from Knowledge Base that needs to be imported into Process Modeling Tool. The process model is programmatically read from Vaticle TypeDB Knowledge

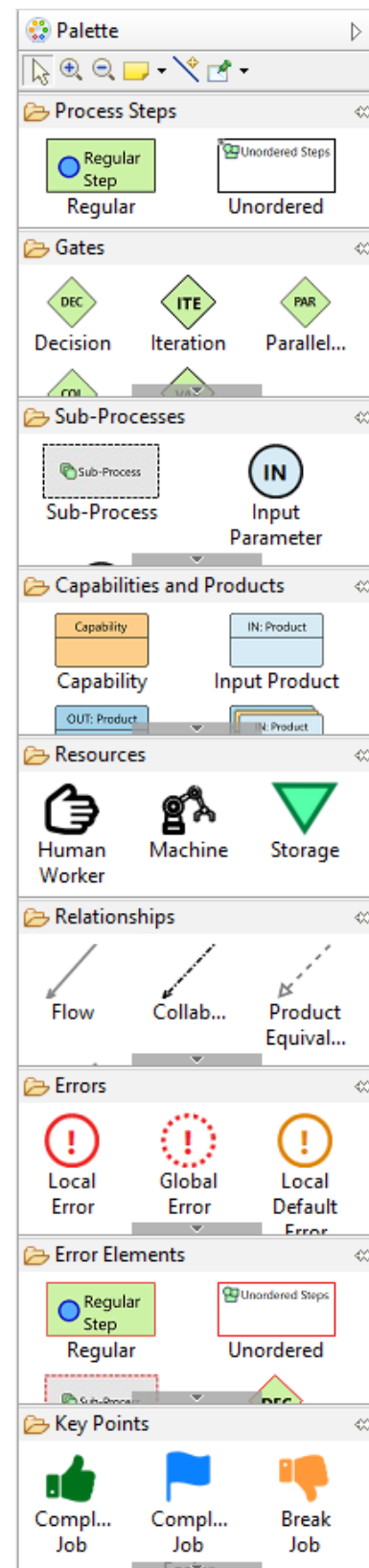


**Figure 7.9.** The generate documentation dialog.

Base by Orchestrator's inner components and automatically transformed into the JSON file. The JSON process model is sent to Process Modeling Tool's API, after which it is automatically transformed into the production process model and stored in the chosen modeling project.



**Figure 7.10.** Tool palette of Process Modeling Tool without additional layers turned on.



**Figure 7.11.** Tool palette of Process Modeling Tool with both additional layers turned on.

## 7.5 Summary

In this section, we presented MultiProLan, a DSML for production process modeling, whose models are suitable for dynamic production orchestration and automatic generation of executable resource instructions and manufacturing documentation. First, we presented the abstract syntax of MultiProLan and various constraints as part of the syntax. The abstract syntax is developed based on the FODA models related to production process modeling, discussed in the previous section. Then, we presented the concrete graphical syntax that is implemented based on the presented abstract syntax. The graphical syntax was chosen over the textual syntax as process designers are already familiar with various process charts. MultiProLan is created in a way to fulfill the requirements defined in Section 4.2.1, thus allowing the creation of resource-agnostic production process models that can be easily orchestrated in a chosen production system and transformed into resource-aware production process models and executable resource instructions. The modeling of production processes by means of MultiProLan is possible through Process Modeling Tool that is discussed at the end of this section. Process Modeling Tool covers various features that provide easier production process modeling and make process model diagrams more readable and understandable.

To test the usability of MultiProLan, Process Modeling Tool, and the whole MD solution in the assembly industry, we created two proof-of-concept use cases. The first use case demonstrates the possibilities of MultiProLan and Process Modeling Tool in creating production process models with different product variations. The second use case demonstrates the usability of the whole MD solution in creating objects from LEGO® bricks. Different robots, including industrial mobile robots, and human workers participated in the second use case in which they received instructions from our MD solution and assembled objects from the bricks. These two use cases are presented in detail in Section 8.

Besides demonstrating the usage of our MD solution in various use cases, it needs to be tested by users from different user groups. MultiProLan and Process Modeling Tool, as the core elements of our MD solution, were evaluated by participants of the experiment, in which they modeled production processes. The evaluation participants tested whether MultiProLan and Process Modeling Tool fulfill certain quality characteristics. The evaluation process and results are described in Section 9.

## 8 Application of the MD Solution and MultiProLan

Our MD solution and MultiProLan are tailored primarily for discrete product manufacturing, especially for the assembly industry, but can be used for production process modeling in general. Therefore, our solution is mainly applied in assembly production, with the potential to be applied in the process industry as well. Many researchers have also been applying their solutions in assembly production, as such production can range from a relatively easy production process with a few simple process steps to a very complex production process with numerous steps. Therefore, assembly production can be a good starting point to make a proof-of-concept of a novel solution, as it has discrete and precise process steps included in more or less complex production processes. To validate our MD solution and MultiProLan, we applied them in two proof-of-concept use cases and presented them in this section.

The first use case is created to demonstrate the possibilities of MultiProLan and Process Modeling Tool. In this use case, a customized wooden box is assembled, whose production process model includes concepts, such as different product and process variations, collaboration between resources, and parallel execution of process steps.

The second use case represents a demonstration environment created to test the solution in assembling objects from LEGO<sup>®</sup> bricks, avoiding failure costs associated with the real production system. In this demonstration environment, different robots are used. Some robots are industrial mobile robots, which are often used in real production, and some are research-grade smart robots used for demonstration purposes. Human workers are also present in this use case in order to assemble LEGO<sup>®</sup> bricks.

Our demonstration environment supports the assembly of objects from LEGO<sup>®</sup> bricks, whose production processes do not require the full potential of MultiProLan, as only a few MultiProLan modeling concepts are used to model these processes. Also, the manufacturing documentation automatically generated from such production process models does not have too much content to present. Therefore, we created the use case of assembling a customized wooden box, which is not a part of the demonstration environment, to present various MultiProLan modeling concepts, and instructions and documentation generated from the created models.

First, we present the use case related to the customized wooden box production in Section 8.1, to introduce the usage of MultiProLan with its full potential. Afterward, the use case related to the demonstration environment is covered in Section 8.2, to present the usage of the whole MD solution in practice. The summary of these two proof-of-concept use cases is discussed in Section 8.3.

## 8.1 Example of MultiProLan Models of Customized Wooden Box Assembly

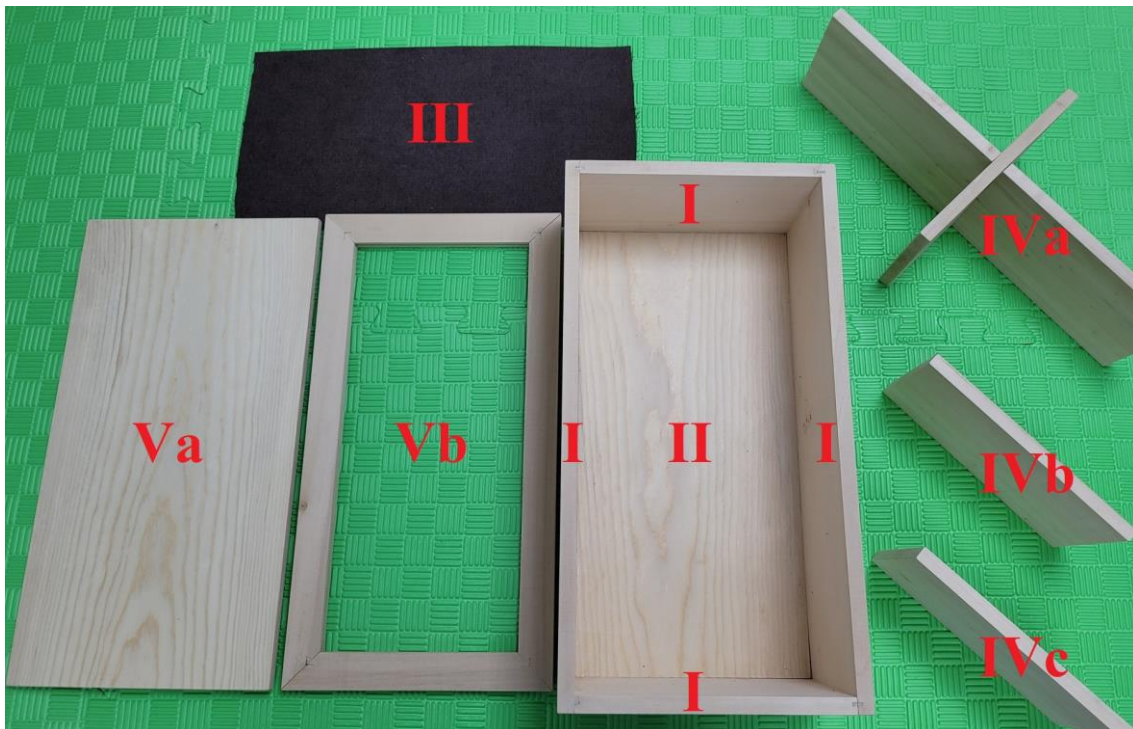
In this section, we present a process of wooden box production modeled using MultiProLan and Process Modeling Tool. The wooden box MultiProLan process models were already discussed in [10,11], and in this thesis, they are extended and discussed in more detail. The box is composed of the following parts, presented in Figure 8.1:

- i) four wooden planks that represent different sides of the box;
- ii) a thin wooden back side;
- iii) a piece of fabric that is to be glued on the back side;
- iv) dividers that separate the space inside the box; and
- v) a lid that is to be placed on the top of the box.

First, the four wooden planks are assembled into a frame using wooden dowel pins, and the wooden back side is hammered into the frame, creating the box. A piece of fabric is then glued at the bottom of the box. Afterward, various dividers can be chosen and glued inside the box, and a lid of the selected type is placed at the top of the box. An example of the produced box is presented in Figure 8.2. Production of wooden boxes is performed in a smart factory, which is composed of:

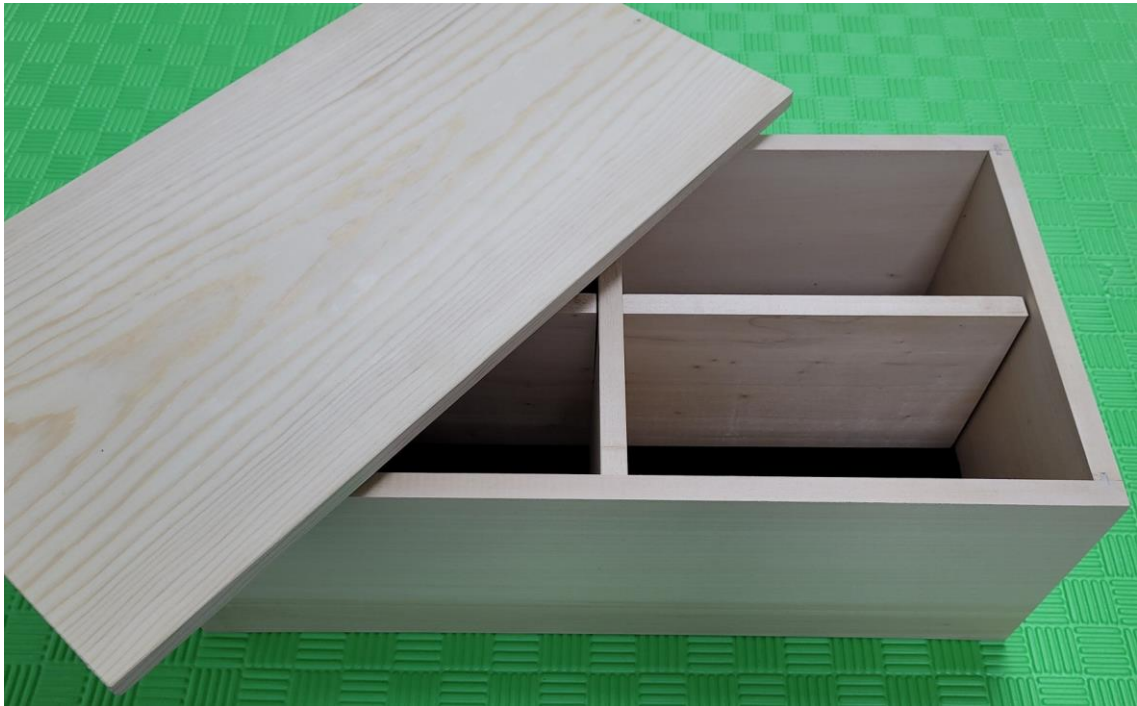
- the smart shelf – storage where wooden planks, dowel pins, fabrics, dividers, and lids are stored;
- the first assembly table – storage used to assemble four wooden sides;
- the second assembly table – storage used to hammer the back side into the frame;
- the third assembly table – storage used to glue fabric and dividers and place lids;
- the recycle bin – storage for impaired parts;
- the finishing area – storage for finished boxes; and
- human workers and mobile robots – smart resources able to perform required activities.

This section is divided into the following subsections. In Section 8.1.1, a MasL process model of wooden box production created by process designers is presented. A DetL process model of



**Figure 8.1.** Parts of the wooden box used for the assembly.





**Figure 8.2.** A variation of the assembled wooden box.

assembling the frame, created by Orchestrator, is presented in Section 8.1.2. Based on the presented DetL process model, the automatically generated high-level and resource-specific instructions are presented in Section 8.1.3, as well as the process monitoring. In Section 8.1.4, the automatically generated manufacturing documentation from created process models is outlined.

### 8.1.1 Master-Level Process Model of Wooden Box Production

In this section, we present a MasL process model of wooden box production, composed of the main process model and several sub-process models. The main MasL model of wooden box production is presented in Figure 8.3. It is composed of the following elements:

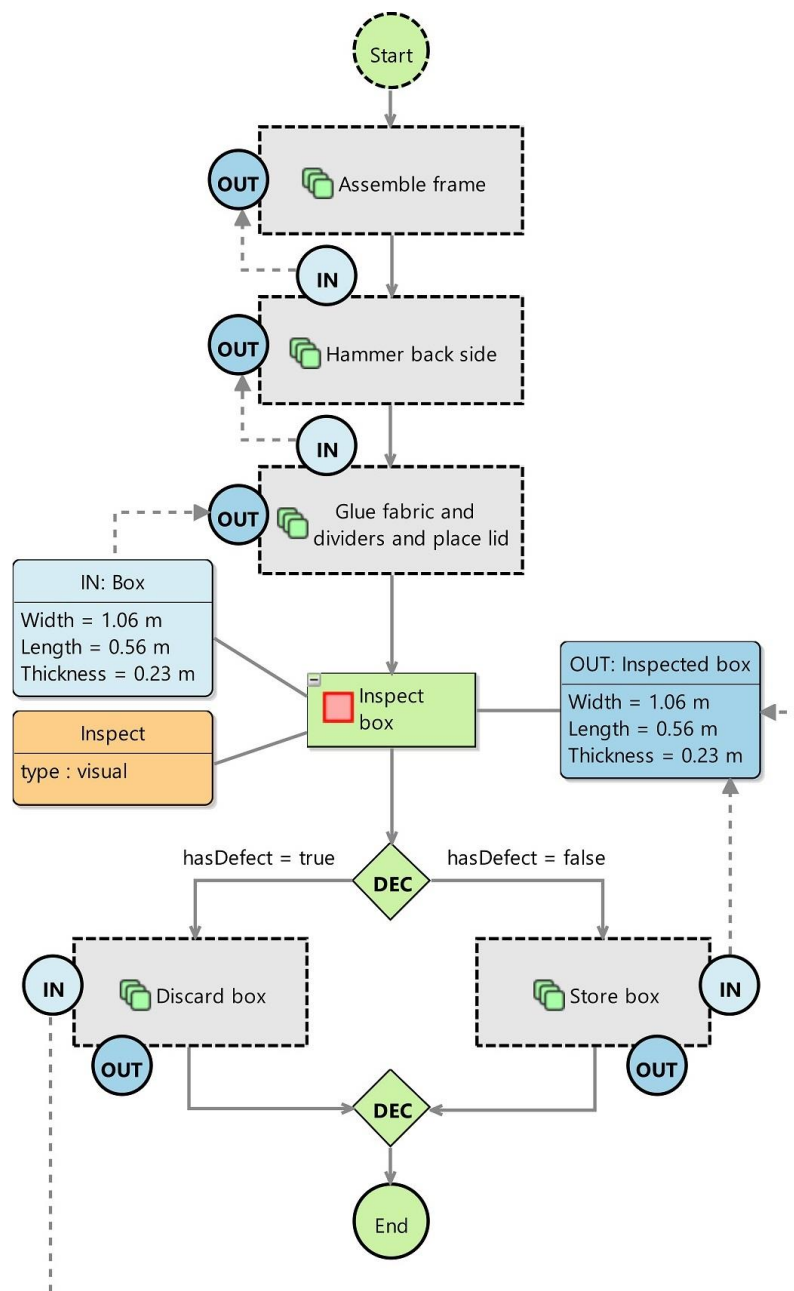
- the *Start* process step;
- the sub-process of assembling a frame of the box;
- the sub-process of hammering a back side into the frame;
- the sub-process of gluing a piece of fabric and dividers into the box and placing a lid on the top of the box;
- the visual inspection of the box;
- based on the inspection result, the created wooden box is discarded if there is any defect identified, or the box is stored if there is no defect identified on the box; and
- the *End* process step.

The output of the *Assemble frame* sub-process represents a frame, which is the input to the *Hammer back side* sub-process, and the output of this sub-process is the frame with the fixed back side, used in the following *Glue fabric and dividers and place lid* sub-process as the input. Outputs and inputs of sub-processes are connected via sub-process parameters and parameter links. After the box is completed, it is used in the *Inspect box* process step to visually detect if there is any defect. In the *Inspect box* process step, an input product is the same as the output of the previous sub-process, and they are connected through the parameter link between the input product and the output parameter. Finally, between the decision gates, a choice is made whether the box is to be discarded or stored based on the inspection results.

In the rest of this section, we describe the first three sub-processes in detail. The first sub-process has parallel process steps of assembling the frame and is used to demonstrate the creation of the DetL process model and resource instructions, as well as to demonstrate process execution and monitoring. The second sub-process has collaborative activities to hammer the back side into the frame. The third sub-process has different variations of dividers and lids; thus, it is used to demonstrate product variations, as well as the automatic generation of the documentation.

The *Assemble frame* sub-process is presented in Figure 8.4. This MasL process model is composed of the following elements:

- the *Start* process step;
- inserting wooden dowel pins into wooden sides and assembling left-bottom and right-upper sides in parallel;
- assembling the frame with the previously assembled left-bottom and right-upper sides; and
- the output process parameter and the *End* process step.



**Figure 8.3.** The main MasL process model of wooden box production.



The process starts with two branches placed between parallelism gates. These two branches have equivalent process steps of inserting wooden dowel pins into one wooden side and assembling two wooden sides into half of the frame. Therefore, only the assembly of the left-bottom sides is discussed in detail, and the assembly of the right-upper sides is done similarly.

The first process step in the left-hand side branch is to insert wooden dowel pins into the bottom side. The process step represents an operation as it is depicted with a circle icon to the left of the process step name. It has two input products, the frame's bottom side, and four wooden dowel pins, all retrieved from storage. The inverted triangle icon to the left of a product name represents that an input product should be retrieved from storage. Two input products have various constraints that are to be considered by Orchestrator when it assigns smart resources able to perform tasks. The bottom side has width, length, and thickness constraints, representing the dimensions of the plank, later used to determine which smart resources are able to pick or place it. Similarly, wooden dowel pins have length and diameter constraints. There are also additional constraints, such as the mass of the planks, but they are not presented in the diagram to make it as simple as possible. The same process step has the *Place* capability with parameters representing four wooden dowel pins with 0.07 m of space between them, while two pins are inserted into the left side of the wooden plank, and other two are inserted into the right side of the plank. The output product of this process step is the bottom side with wooden dowel pins, which will not be stored but used by the following process step. The following step is the assembly of the bottom and left sides. The input of such a process step is the bottom side with pins from the previous process step and the left side that needs to be retrieved

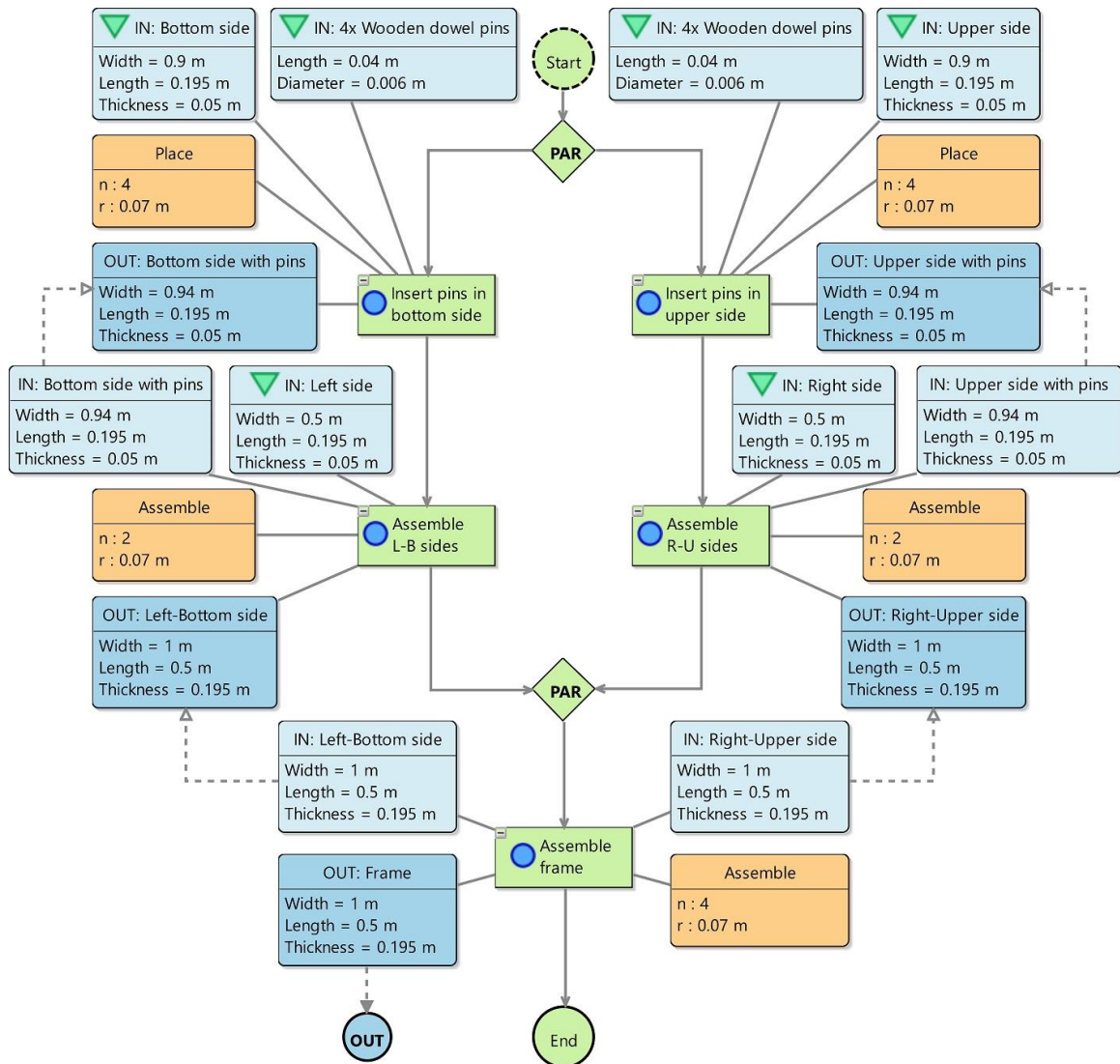


Figure 8.4. The Assemble frame MasL sub-process model.

from storage. The process step has the *Assemble* capability with parameters similar to the *Place* capability, and the output of such a process step is the assembled left-bottom side of the frame.

The assembly of the right-upper sides is represented with two process steps equivalent to the process steps of assembling the left-bottom sides. Inserting wooden dowel pins and assembling each half of the frame are process steps that can be executed in parallel, as they are modeled between two parallelism gates. After each half of the frame is assembled, the following process step (*Assemble frame*) is to assemble the frame, having two input products, the left-bottom and right-upper sides, resulting from the previous two process steps. These input products are not retrieved from storage but are equivalent to the previous process steps' output products, as depicted by the directed dashed lines in the process diagram. This process step has the *Assemble* capability and the frame as the output product. The output product is sent to the output parameter and used in the following sub-process (*Hammer back side*) of the main process (see Figure 8.3).

The *Hammer back side* sub-process is depicted in Figure 8.5. The MasL model presented in the figure is composed of the following elements:

- the input process parameter and the *Start* process step;
- collaborative activities of hammering the back side into the frame produced in the previous sub-process;
- the visual inspection of the frame with the back side;
- based on the inspection result, the created frame is discarded if there is any defect identified, or the sub-process ends and the frame is sent to the following sub-process; and
- the output process parameter and the *End* process step.

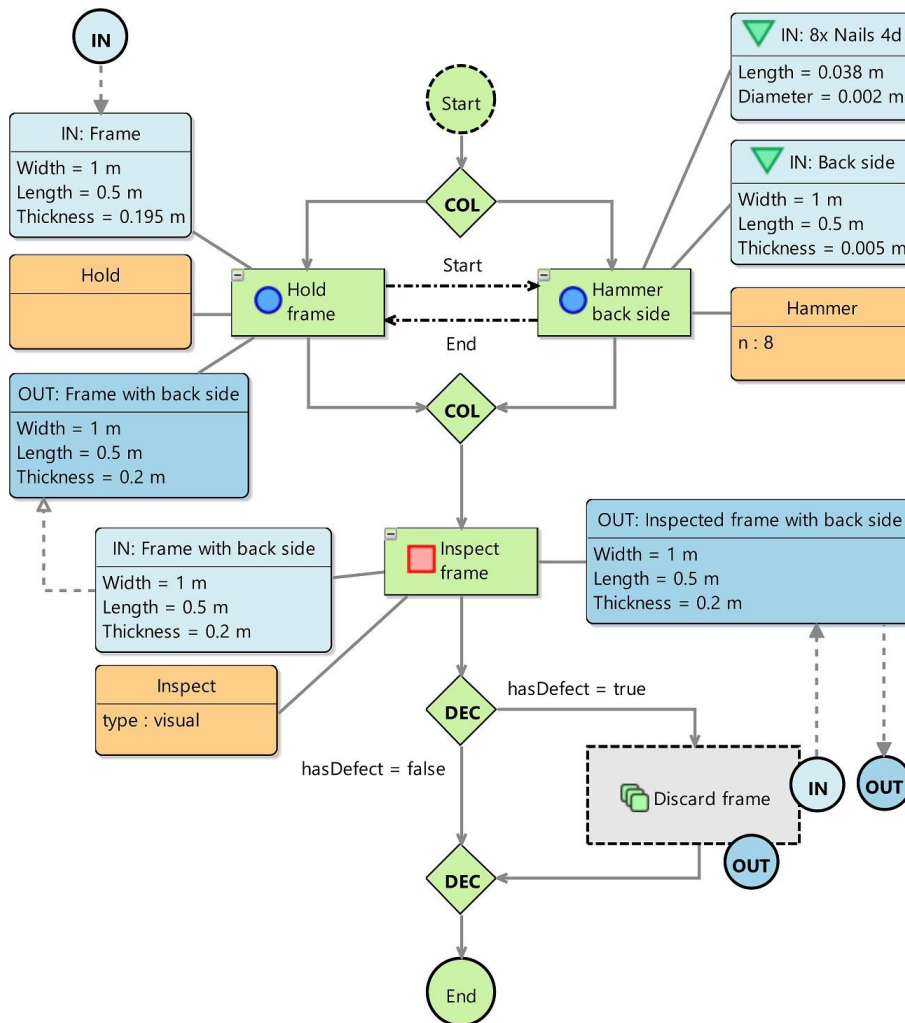


Figure 8.5. The *Hammer back side* MasL sub-process model.

The sub-process starts with collaborative activities of holding the frame and hammering the back side into the frame. These activities are presented as process steps in two branches between collaboration gates. The *Hold frame* process step has the frame as the input product, and this frame is the one assembled in the *Assemble frame* sub-process, imported through the input parameter. This process step has the *Hold* capability and the frame with the back side as the output product. Another collaborative process step is *Hammer back side*, which is the hammering of the back side into the frame that is held. The input products of the *Hammer back side* process step are the back side and eight nails needed for hammering. These input products are retrieved from storage. This process step has the *Hammer* capability, having a predefined number of nails that should be hammered, e.g., eight. It does not have an output product, as the back side is going to be hammered into the frame, and thus, the output product is modeled only in the *Hold frame* process step. The *Hold frame* and *Hammer back side* process steps are collaborative process steps, meaning they are done in a parallel but in a synchronized manner. Hammering the back side should not start before the message arrives that the frame is being held. The frame should be held until the message arrives that the hammering is finished. Such a message exchange is presented in the process diagram with dotted-line relationships between those two process steps. After the collaborative process steps are finished, the frame with the back side is inspected for any deformation. The *Inspect frame* process step requires a visual inspection of the frame, and the inspected frame is sent to the output parameter, used in the following sub-process (*Glue fabric and dividers and place lid*) of the main process (see Figure 8.3). The decision to discard the frame or finish the sub-process depends on whether the frame passes all checks.

The inspected frame that passes visual checks in the *Hammer back side* sub-process is used in the *Glue fabric and dividers and place lid* sub-process, presented in Figure 8.6. The MasL model presented in the figure is composed of the following elements:

- the input process parameter and the *Start* process step;
- gluing a piece of fabric onto the bottom of the frame;
- variations of gluing dividers of different types;
- waiting for the glue to dry;
- variations of placing lids of different types; and
- the output process parameter and the *End* process step.

Before discussing the process model presented in Figure 8.6, the wooden box product variations need to be mentioned. The wooden box can have dividers of different types: (i) a single divider that vertically divides the box into two halves; (ii) two single dividers that vertically divide the box into three thirds; and (iii) a cross divider that divides the box into four quarters. The wooden box can have lids of different types: (i) a fully wooden lid; and (ii) a wooden lid with a glass opening in the middle. The wooden box can have any combination of dividers and lids. Therefore, six product variations of the wooden box are modeled and presented in Figure 8.7, and the images of these six product variations are presented in Figure 8.8.

Each process element in the wooden box production process model has variations assigned in which the elements appear. For example, in the *Glue fabric and dividers and place lid* sub-process model, presented in Figure 8.6, the *Place wooden lid* process step has three variations assigned: *1.1 Box\_SD\_WL*, *1.3 Box\_TD\_WL*, and *1.5 Box\_CD\_WL*, meaning that the wooden lid can be used in any divider type previously used. However, the *Wait for glue to dry* process step has all the variations assigned, as this process step must be executed independently of any variation. The assigned variations to process elements are considered during the process execution and the documentation generation when the desired product variation is chosen. In addition, the process model diagram, presented in Figure 8.6, is burdened with details, even with only a few variations, which can make the diagram hard to read by process designers. By using the Filter variations function, outlined in Section 7.4, the complexity of the process model diagram can be reduced; thus, process designers can be focused only on a single variation. For example, in Figure 8.9, we present a process model diagram after the Filter variations function is executed, choosing the variation in which two single dividers are glued, and the fully wooden lid is placed.

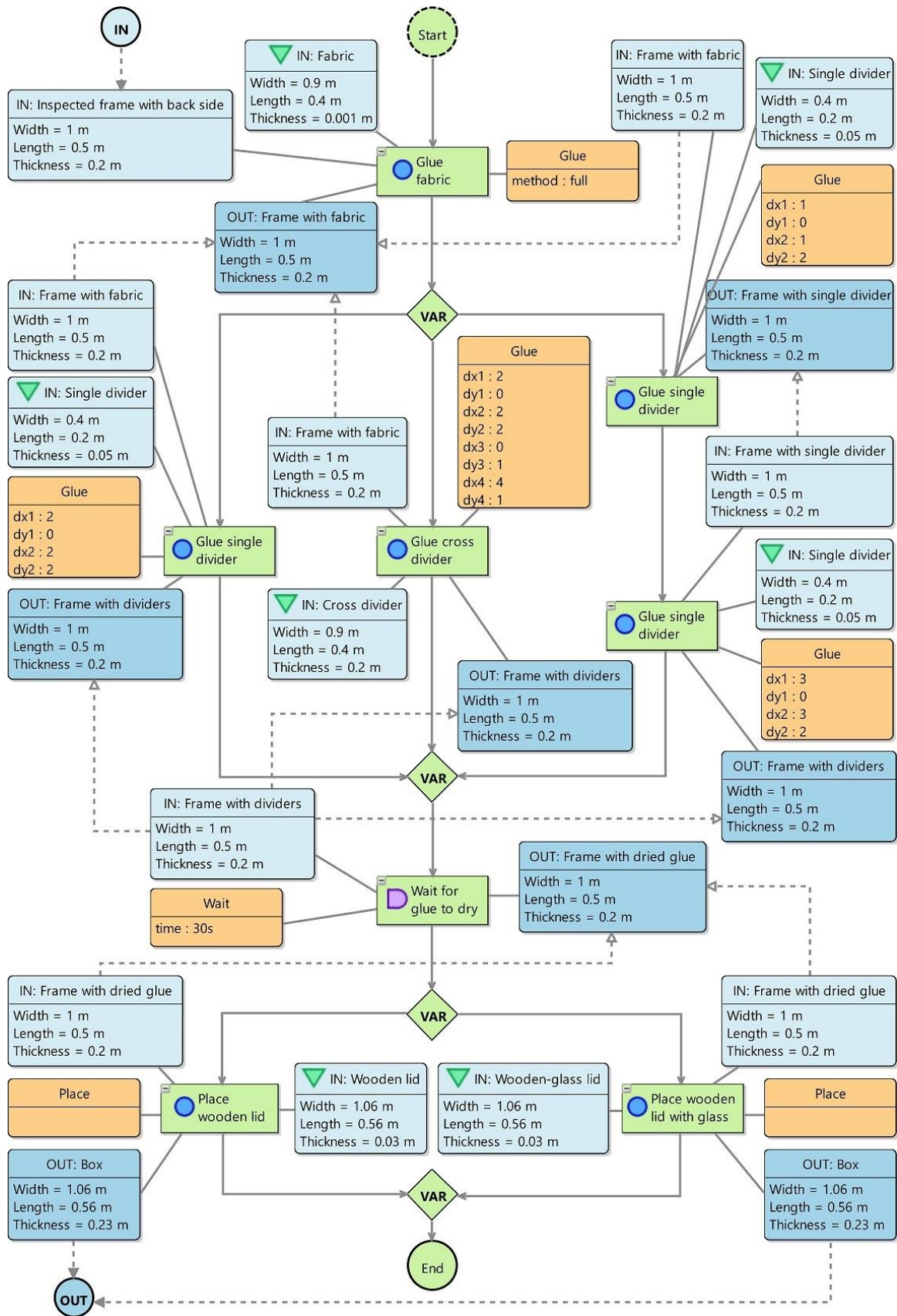
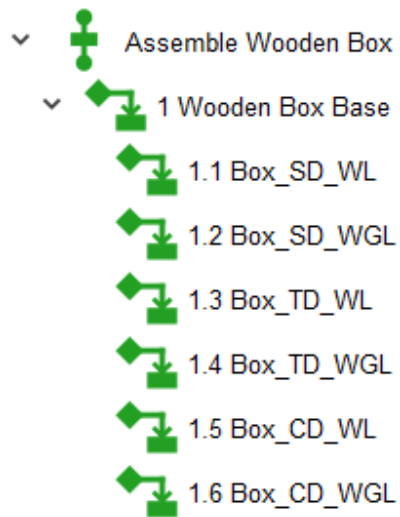
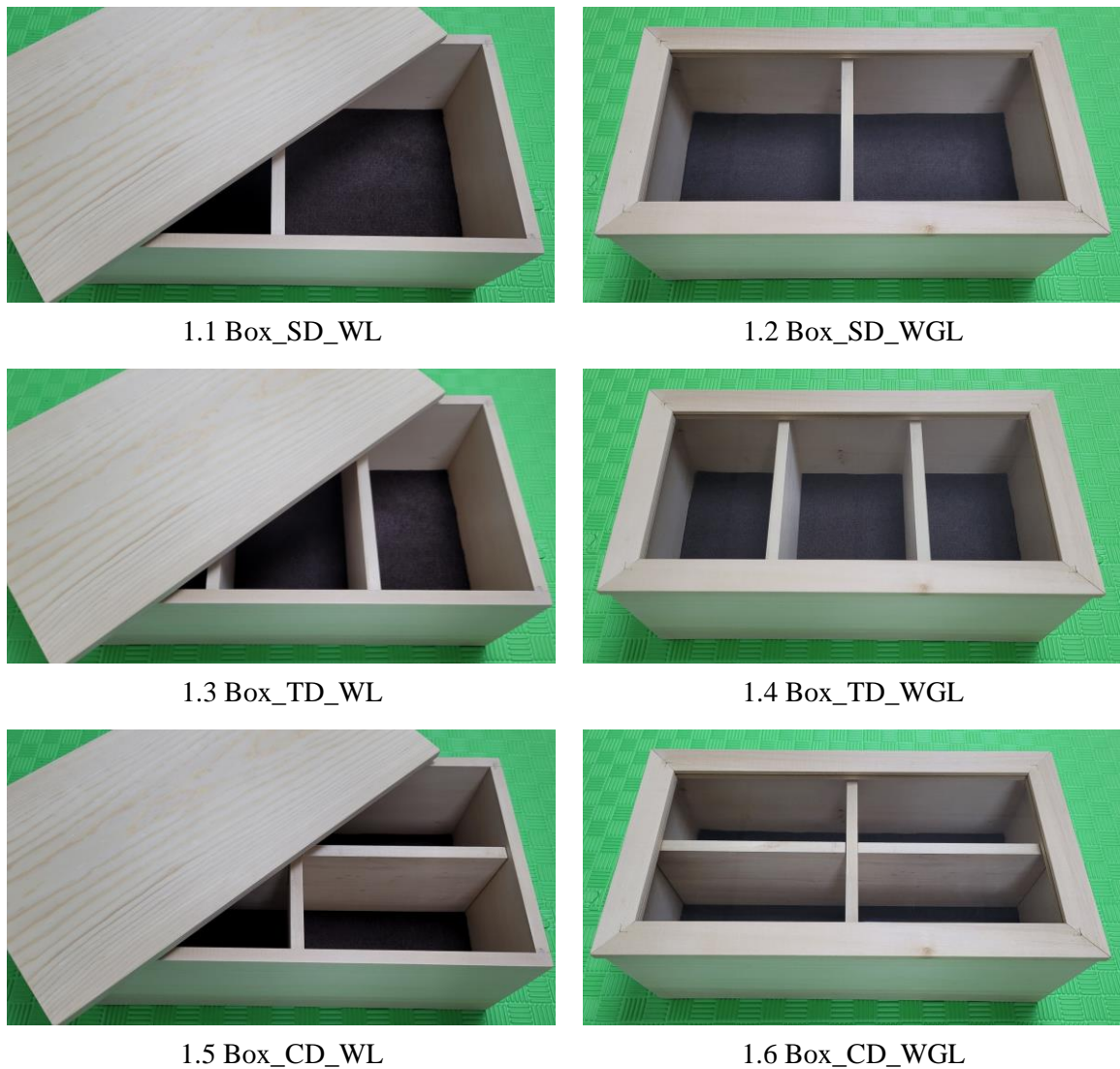


Figure 8.6. The *Glue fabric and dividers and place lid* MasL sub-process model.





**Figure 8.7.** The wooden box product variations model.



**Figure 8.8.** The wooden box variations.

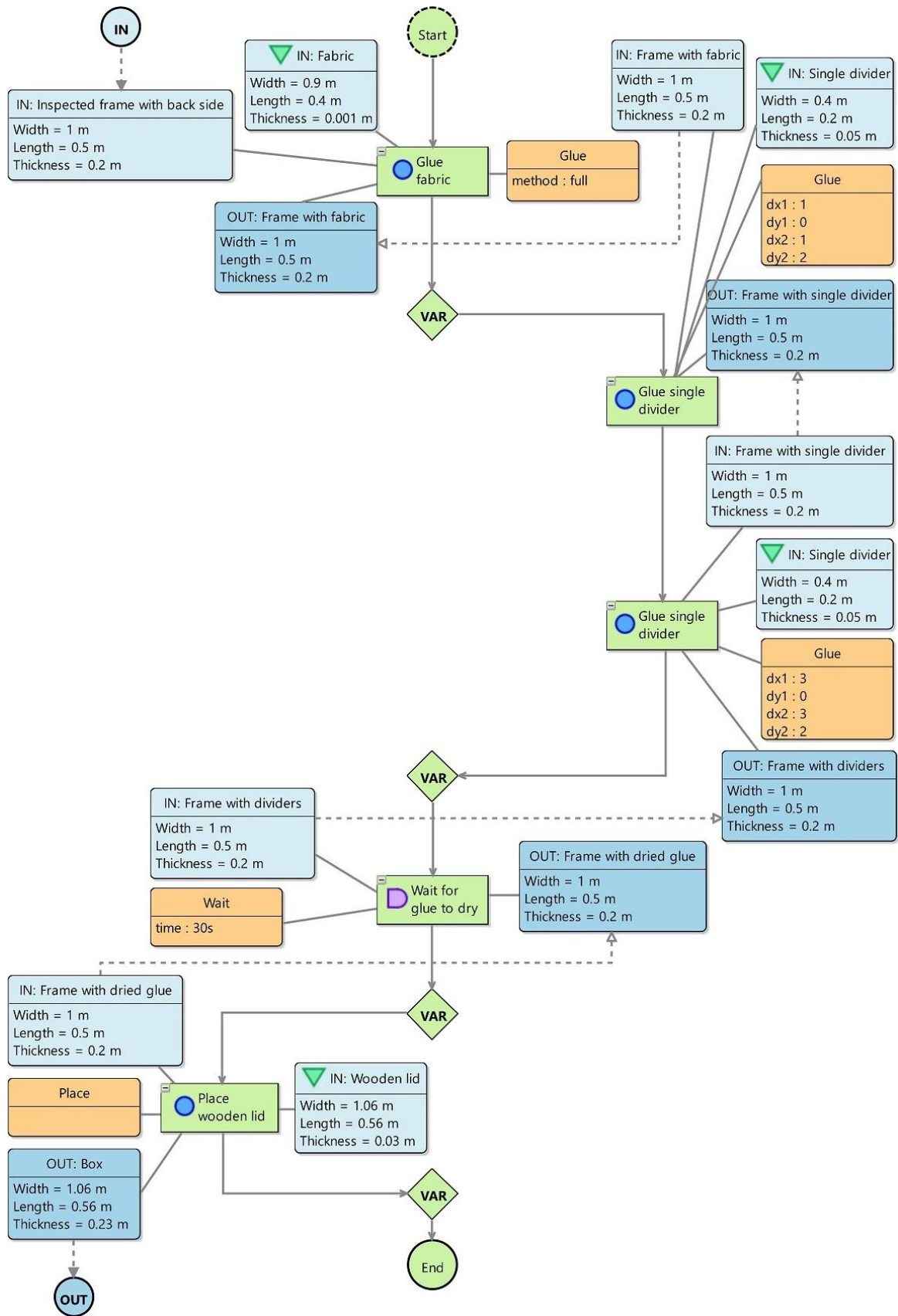


Figure 8.9. The *Glue fabric and dividers and place lid* MasL sub-process model with a single variation (1.3 Box\_TD\_WL).

For the rest of this section, we describe the *Glue fabric and dividers and place lid* sub-process model presented in Figure 8.6. The sub-process starts with gluing a piece of fabric onto the bottom of the frame with the hammered back side. The input products of the *Glue fabric* process step are the inspected frame, which results from the previous sub-process, imported through the input parameter, and a piece of fabric retrieved from storage. The process step has the *Glue* capability, with a parameter that indicates that gluing needs to be applied fully on the piece of fabric and the bottom of the box. The output product of the *Glue fabric* process step is the frame with the piece of fabric glued, which is used in one of the following variations.

After the *Glue fabric* process step, there are three branches between variation gates, each representing a single option of modeled dividers that can be glued in the box. The left-hand side branch contains the *Glue single divider* process step, which divides the box into two halves. The input products are the frame with the fabric, created in the previous process step, as well as the single divider retrieved from storage. The *Glue* capability does not have the method parameter but the coordinates of two points on which the glue is to be applied and placed in the box. The output of this process step is the frame with the glued divider. The middle branch contains the *Glue cross divider* process step, which divides the box into four quarters. The process step has similar input and output products and the *Glue* capability. The difference is that a cross divider is used instead of a single divider, and the glue needs to be applied on the four points and placed in the box. The right-hand side branch contains two *Glue single divider* process steps, each gluing a single divider, but in different places in the box, dividing the box into three thirds. After the fabric and dividers are glued, the process step of waiting for the glue to dry for 30 seconds is performed. This process step represents a delay activity, as depicted with the icon to the left of the process step name. The input product of such a process step is the frame with glued fabric and dividers, which is the output product from one of the previous process steps from different variations. The *Wait for glue to dry* process step has the *Wait* capability, with a parameter representing the waiting time, and the output product of the process step is the frame with the dried glue.

The frame is then used in one of two following branches placed between two variation gates. Both represent the activity of placing a lid at the top of the frame, creating the box as a final product. A fully wooden lid is used in the left-hand side branch, while in the right-hand branch, a wooden lid with a glass opening in the middle is used. As an output product of these two process steps, the box is sent to the output parameter and used in the main process for the inspection activity and the decision of whether to discard or store the box.

Based on the presented MasL process models, DetL process models are automatically generated. The *Assemble frame* DetL process model is discussed in the following section to present the automatic generation of such models and to demonstrate differences between MasL and DetL process models.

### 8.1.2 Detail-Level Process Model of Assembling the Frame

Based on the presented MasL models and production system knowledge from Knowledge Base, Orchestrator automatically generates DetL models of wooden box production for the chosen production system. For example, the *Assemble frame* DetL sub-process model, presented in Figure 8.10, is automatically generated from the MasL sub-process model presented in Figure 8.4. Other sub-processes are created in a similar manner and thus are not discussed further in this thesis. To lower the complexity of the presented model diagram in Figure 8.10, products and capabilities are depicted just for process steps in the left-hand side parallelism branch, while for other process steps, they are modeled but hidden in the diagram using the +/- buttons.

The generated DetL model comprises process steps similar to the related MasL model, but the DetL model is extended with additional details and new process steps, such as production logistic activities and mobile robot configurations. These new process steps are needed to produce the box automatically.

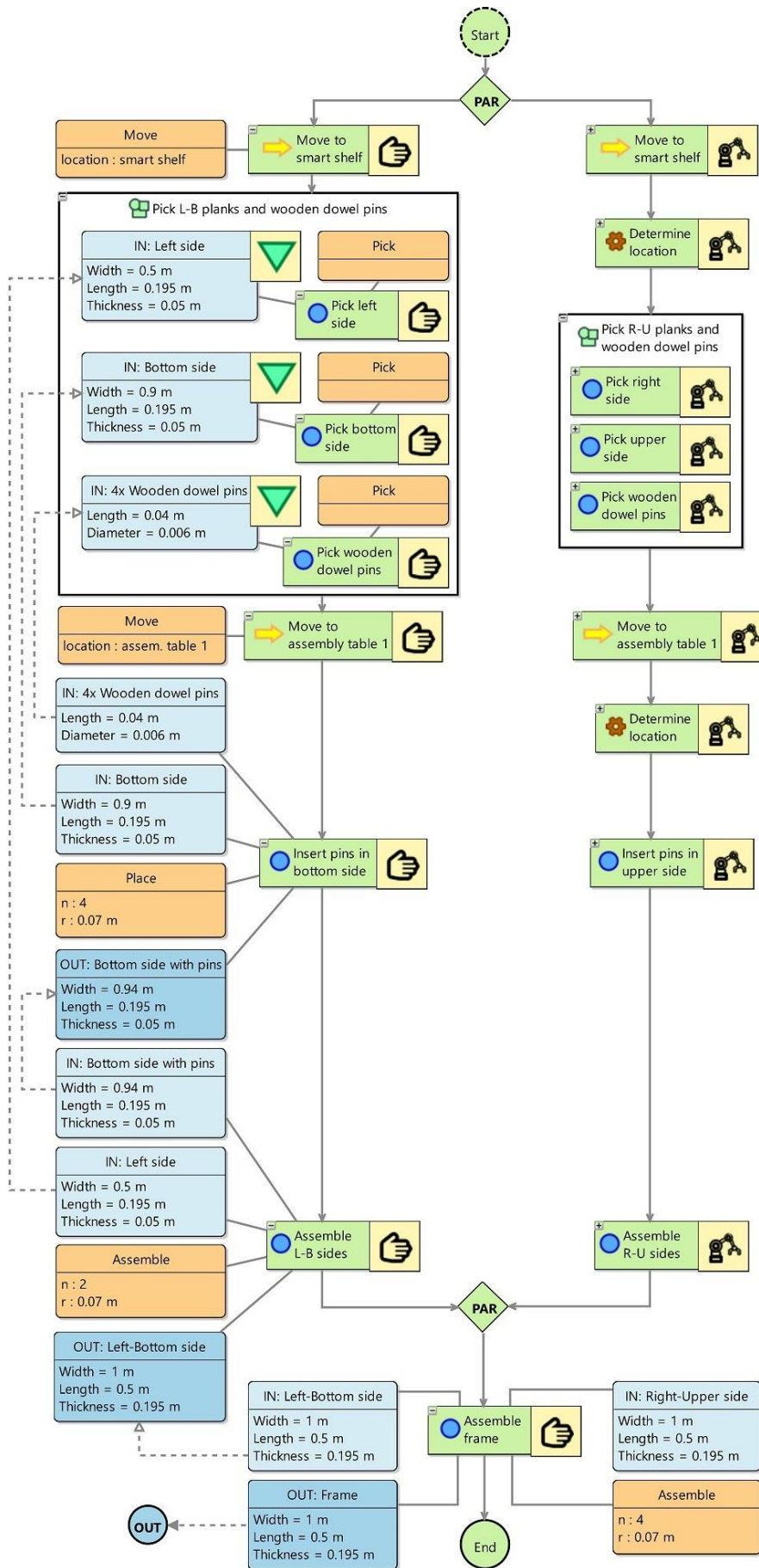


Figure 8.10. The Assemble frame DetL sub-process model.



The *Assemble frame* DetL sub-process model is composed of the following elements:

- the *Start* process step;
- moving to the smart shelf and determining resource position when needed;
- picking up input products from the smart shelf in any order;
- moving to the first assembly table and determining resource position when needed;
- inserting wooden dowel pins into wooden sides and assembling left-bottom and right-upper sides in parallel;
- assembling the frame with the previously assembled left-bottom and right-upper sides; and
- the output process parameter and the *End* process step.

The assembly of left-bottom and right-upper sides are assigned in parallel to a human worker and a mobile robot, respectively. In both parallel branches, transportation process steps are added, which are depicted with the arrow icon to the left of the process step name. To assemble the left-bottom side, the human worker needs to move to the smart shelf, pick the left and bottom sides and wooden dowel pins in any order, move to the first assembly table, insert pins into the bottom side, and assemble the left and bottom sides. Transportation process steps only have the *Move* capability with the location parameter, as this capability requires no input or output products. The *Pick* process steps have a capability and an input product, but an output product does not exist, as it is the same as the input product. Unlike the MasL model, in which input products have general, i.e., abstract storage as an indicator that the products need to be retrieved from it, the DetL model input products have specific storage, such as the specific smart shelf, from which the products need to be retrieved. The inverse triangle objects set on input products depict the specific storage. By selecting storage, it is possible to specify or change the values of the storage attributes. The same can be done with resources set on process steps. As for the *Insert pins in bottom side* process step input products, they are equivalent to the previously picked bottom side and wooden dowel pins. Such equivalent links are denoted with the directed dashed lines between equivalent products. Similarly, the *Assemble left-bottom sides* process step input products are equivalent to the previously picked left side and the produced bottom side with wooden dowel pins. These two process steps' capabilities and output products are the same as in the MasL model.

The second parallel branch represents the assembly of the right-upper sides by the mobile robot. The process steps in this branch are similar to those of the previously described branch, except for the configuration process steps. As the mobile robot assigned to these process steps is not equipped with a machine vision module, it must calibrate itself after each movement to determine its position. The gear icon to the left of the process step name can differentiate configuration process steps from others.

After the left-bottom and right-upper sides are assembled, the same human worker assembles the frame. This activity does not require any transportation process steps as the human worker and the required input products are already at the first assembly table. The assembled frame is used in the following sub-process (*Hammer back side*), as it is sent via the output parameter to the main process model (see Figure 8.3).

Such a DetL process model is ready for the automatic generation of instructions, as it contains all the necessary information related to a process and a production system. In the following section, we present an example of generated instructions as well as process monitoring during the execution of the process.

### 8.1.3 Automatically Generated Instructions and Process Monitoring

The DetL model, presented in Figure 8.10, is suitable for the automatic generation of instructions. Instruction Generator generates high-level instructions from the DetL model and passes the instructions to Digital Twin. By using protocol transformation components, Digital Twin transforms high-level instructions into resource-specific instructions, i.e., human-readable or machine-specific instructions, and sends them to smart resources for execution. In this section, we present automatically generated instructions for a smart mobile robot and a human worker.

As an example, we demonstrate the instruction generation for the *Pick right side* process step, which is similar to the *Pick left side* process step presented in detail in Figure 8.10. It contains the *Pick* capability without any parameters, and the *Right side* input product with constraints and the specific storage, the smart shelf, from which it is to be retrieved. There is also the smart mobile robot assigned to execute such a process step.

From such a DetL process step, Instruction Generator can automatically generate a high-level instruction. An example of such a high-level instruction in a JSON format is presented in Listing 8.1. The instruction starts with the order identifier, which Orchestrator adds, indicating to which customer order the process and process step belongs. Other instruction attributes are automatically generated from the process model except for the scheduled time. After the process identifier is specified, which relates to the executed process, the *Pick right side* process step attributes are presented in Listing 8.1, such as the identifier, name, description, image path, and video path of the process step. Afterward, the *Pick* capability object is specified, alongside its identifier, name, indicator of whether it requires storage, and parameters. The parameters array is empty as the modeled *Pick* capability does not have one specified. Input products and output products arrays are specified next, with the output products array empty as there is no output product for the *Pick right side* process step. Each product object in input and output products arrays has the following attributes specified: identifier, name, quantity, and an indicator of whether the product is stored or is the result of some previous process step. Based on the indicator value, the storage or the equivalent object is specified. As the *Right side* input product is stored on the smart shelf, the storage object is specified, as presented in Listing 8.1, and the equivalent object is empty. The smart shelf storage object has the following attributes specified: identifier, name, and indicators of whether the storage is an actuator as well or is storage only. The resource object is specified next. A smart mobile robot is assigned to execute the instruction, and the following attributes are specified for it: identifier, name, indicators of whether the resource is an actuator and whether it has local storage on it, and the protocol to which the instruction is to be sent. The instruction ends with the scheduled time attribute, automatically added by Orchestrator, indicating the estimation time when the process step is to be executed.

```

1  {
2    "orderId": "720f89365d3a",
3    "processId": "b6624c68-709d-4afa-a59a-8b0178af8f73",
4    "processStepId": "2177e6b8-a653-4ebe-9cf2-fcf55d5a2b65",
5    "name": "Pick right side",
6    "description": "Pick a frame's right-side wooden plank.",
7    "image": "<path>/PickRightSide.jpg",
8    "video": "<path>/PickRightSide.avi",
9    "capability": {
10     "id": "e1a68dce-698c-41fc-af0d-20ff9e1ac093",
11     "name": "Pick",
12     "requiresStorage": true,
13     "parameters": []
14   },
15   "inputProducts": [{
16     "id": "31b93c3f-7ac1-4116-b1f5-b69a730f9b24",
17     "name": "Right side",
18     "quantity": 1,
19     "isStored": true,
20     "storage": {

```

```

21         "id": "88d376a8-ee43-4ca3-a1af-ff5ee7192d2f",
22         "name": "Smart shelf",
23         "isActuator": false,
24         "isStorage": true
25     },
26     "equivalent": {}
27 },
28 "outputProducts": [],
29 "resource": {
30     "id": "91a3f687-fe14-4897-909a-53bd63f0ef37",
31     "name": "Mobile Robot M7",
32     "isActuator": true,
33     "isStorage": false,
34     "protocol": "ROS"
35 },
36 "scheduledTime": "2023-02-14T13:45:30+00:00"
37 }

```

**Listing 8.1.** An example of the *Pick right side* high-level instruction.

The presented high-level instruction is sent to Digital Twin for execution in a simulation. It can also be transformed into a machine-specific instruction automatically, which is sent to a shop floor to be physically executed by a smart resource. As the resource protocol indicates that the high-level instruction is to be sent through the ROS [227,228] protocol, the high-level instruction is transformed into the ROS instruction. The smart mobile robot runs ROS and thus can perform various ROS instructions.

An example of the *Pick* ROS instruction is presented in Listing 8.2, in a structured textual form. The *Pick* instruction is to be sent to the service address composed of the smart resource's name and the command to be executed or, in this case, the *MobileRobotM7/Pick* address. The specific resource, such as Mobile Robot M7, has an action server and waits for a command to arrive, after which the resource can send the feedback to an action client that sent the instruction. The instruction begins with an indicator of whether a product is to be picked from the smart robot itself or storage. The marker identifier refers to the storage location from which a product is to be picked, e.g., the location of the smart shelf from which the right-side plank is to be picked. The instruction ends with two more attributes related to the coordinates from which a product is to be picked. First, the reference point in the coordinate system is specified, and then the related coordinates from the reference point are specified, as well as the orientation of the product. In Listing 8.2, the smart shelf marker position is specified as the reference point. The right-side planks are stored at the top of the smart shelf, and their position is defined in the instruction. In addition, the orientation of the product indicates that the right-side planks are not rotated but placed in line with the smart shelf.

```

1  isPickFromSelf: false
2  markerId: MarkerSS01
3  referencePoint: MarkerSS01
4  referencePose:
5      position:
6          x: 0
7          y: 6
8          z: 2
9      orientation:
10         x: 0
11         y: 0
12         z: 0
13         w: 1

```

**Listing 8.2.** An example of the *Pick right side* machine-specific ROS instruction.

Another example of a high-level instruction is generated from the *Assemble left-bottom sides* process step. It contains the *Assemble* capability with two parameters. The input products are

equivalent to products of previous process steps, and the output product represents the assembled *Left-Bottom side*. A human worker is assigned to execute such a process step.

A high-level instruction is automatically generated from such a process step and is presented in Listing 8.3. This high-level instruction has a similar structure to the one presented in Listing 8.1. However, there are a few differences that need to be discussed further. The *Assemble* capability has two parameters specified in the process model. Therefore, these parameters must be part of the instruction, as presented in lines 13 through 19. Each parameter has its key and value specified in the instruction. Another difference is that input products result from previous process steps and not storage. The *isStored* indicator is *false*, and the storage object is empty. Instead, the equivalent object is specified as presented in lines 27 through 30 and 37 through 40. The output product is presented in this instruction and specified in lines 42 through 49. Finally, compared to the instruction presented in Listing 8.1, the last important difference this instruction introduces is the human worker resource. In this instruction, a human worker is assigned, and the protocol of such a resource is a human interface, referring that the instruction is to be sent to a human worker device.

```

1  {
2    "orderId": "720f89365d3a",
3    "processId": "b6624c68-709d-4afa-a59a-8b0178af8f73",
4    "processStepId": "b6851c5e-2941-4568-8cdf-e1dfa67e300b",
5    "name": "Assemble left-bottom sides",
6    "description": "Assemble frame's left-side and bottom-side wooden planks. Two
7      wooden dowel pins from the bottom-side plank need to be inserted into holes
8      of the left-side plank.",
9    "image": "<path>/AssembleLeftBottomSides.jpg",
10   "video": "<path>/AssembleLeftBottomSides.avi",
11   "capability": {
12     "id": "982dbb73-53ad-4e21-b5f9-81fb803244",
13     "name": "Assemble",
14     "requiresStorage": true,
15     "parameters": [{
16       "key": "n",
17       "value": "2"
18     }, {
19       "key": "r",
20       "value": "0.07 m"
21     }]
22   },
23   "inputProducts": [{
24     "id": "6805e009-fc63-4a38-825a-e41d3326271124",
25     "name": "Left side",
26     "quantity": 1,
27     "isStored": false,
28     "storage": {},
29     "equivalent": {
30       "id": "3c89890d-8dc7-4a92-90bc-abec598978eb",
31       "name": "Left side"
32     }
33   }, {
34     "id": "0985589e-0a19-42f3-9153-04d0sadgq21c68",
35     "name": "Bottom side with pins",
36     "quantity": 1,
37     "isStored": false,
38     "storage": {},
39     "equivalents": {
40       "id": "99dd7833-ad56-45e5-a677-14d7258175f3",
41       "name": "Bottom side with pins"
42     }
43   }],
44   "outputProducts": [{
45     "id": "e126c8c1-c38f-4756-a018-66506b04727e",
46     "name": "Left-Bottom side",
47     "quantity": 1,

```

```

46     "isStored": false,
47     "storage": {},
48     "equivalent": {}
49   }],
50   "resource": {
51     "id": "ff30ac13-4b4e-4690-839e-b6c7c725ea74",
52     "name": "Jovan Jovanović",
53     "isActuator": true,
54     "isStorage": false,
55     "protocol": "human interface"
56   },
57   "scheduledTime": "2023-02-14T13:51:00+00:00"
58 }

```

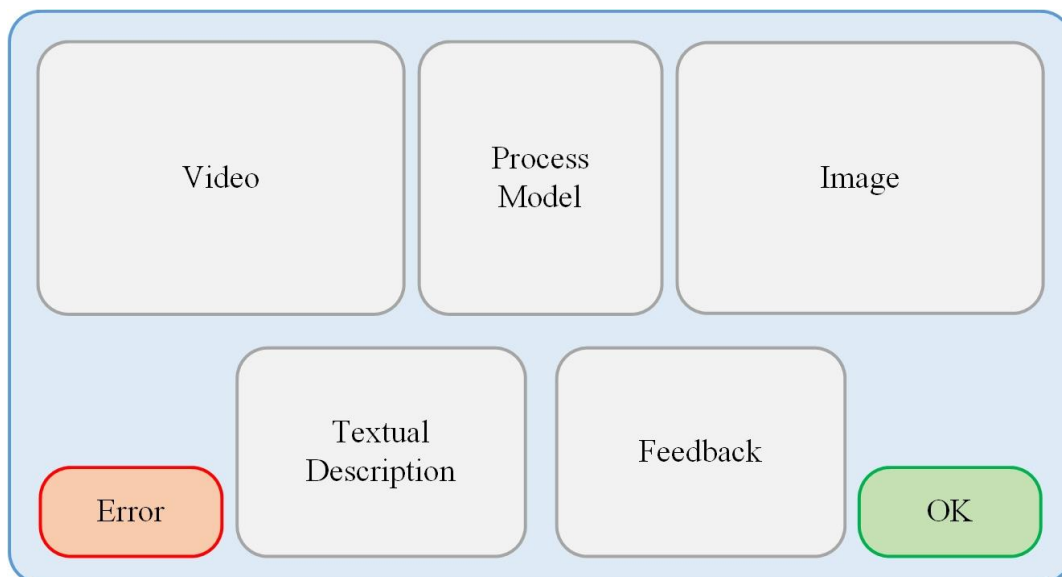
**Listing 8.3.** An example of the *Assemble left-bottom sides* high-level instruction.

The *Assemble left-bottom sides* high-level instruction is sent to Digital Twin for execution in a simulation. It can also be transformed into a human-readable instruction and sent to a human worker to execute it on a shop floor. The high-level instruction needs to be transformed into a format easily readable by human workers. It can be sent to an application running on a tablet, a smart watch, a computer connected to a monitor in front of a worker, or an AR device. The human worker application needs to contain the following elements:

- a textual description of a process step with key points helping a human worker perform a task more easily;
- an image of how to perform a process step;
- a video of performing a process step played by a human worker whenever needed;
- an overview of a process model indicating the process status; and
- buttons to send feedback by a human worker when a process step is performed successfully, or when an error appears.

A user interface mockup of the human worker application is presented in Figure 8.11. The user interface varies depending on the device that is used. For example, if a monitor is used, all the process step content can be presented. However, if a smart watch is used, only a process step name and buttons for a completion status are presented.

After a DetL process model is transformed into instructions and the instructions are sent for execution, the process execution status can be monitored. Monitoring can be done in Process Modeling Tool by gathering the execution feedback from Digital Twin. Based on the gathered



**Figure 8.11.** A user interface mockup of the human worker application.

feedback, the background color of process steps is changing. Process steps can be presented with one of the four background colors, representing their status:

- white – a process step has not been executed yet;
- blue – a process step is currently being executed;
- red – an error has occurred during the execution; and
- green – a process step has been executed successfully.

In addition, when an error occurs, the error background color changes to red, indicating which error occurred.

An example of monitoring the execution of the *Assemble frame* DetL sub-process model is presented in Figure 8.12. To make the process model diagram as simple as possible, all capabilities and products are hidden from it. The process monitoring snapshots are taken at five points in time:

- ( $t_1$ ) The *Assemble frame* process execution is started, and all process steps have a white background, indicating they are not executed yet.
- ( $t_2$ ) Both the human worker and smart mobile robot are executing unordered process steps. The human worker performed more tasks than the robot, as the robot had to be calibrated after moving to the smart shelf.
- ( $t_3$ ) The human worker assembles the left and bottom sides while the mobile robot determines its position in parallel.
- ( $t_4$ ) Parallel process steps are finished, and the human worker assembles the frame.
- ( $t_5$ ) The *Assemble frame* process execution is finished.

The process monitoring feature can help process designers detect and mitigate potential bottlenecks and production process modeling errors. Such a feature is particularly useful when Digital Twin is used in a simulation-only mode. Accordingly, potential bottlenecks and modeling errors can be mitigated before they reach real production.

Besides the automatic instruction generation from process models, the manufacturing documentation can also be automatically generated. The automatic generation and update of the manufacturing documentation can reduce the time spent by process designers to keep the documentation up to date manually. An example of the generated manufacturing documentation is presented in the following section.

#### 8.1.4 Automatically Generated Manufacturing Documentation

After process models are created by using MultiProLan and Process Modeling Tool, they can be used by Documentation Generators to automatically generate and update the manufacturing documentation. The automatic generation of manufacturing documentation can be particularly useful when there are many product variations. Whenever an existing product changes or a new product variation emerges, the documentation needs to be updated, which can happen frequently. Thus, process designers need to spend much time manually changing it.

In this section, we present an example of automatically generated manufacturing documentation based on the previously described process models. The manufacturing documentation is automatically generated for the whole process model of wooden box production, presented in Figure 8.3. However, for demonstration purposes, we present only the documentation generated from the *Glue fabric and dividers and place lid* MasL and *Assemble frame* DetL sub-process models in this section. More precisely, we present the documentation generated for the *1.3 Box\_TD\_WL* product variation presented in Figure 8.9. In such a product variation, a piece of fabric and two single dividers are glued, and after the glue is dried, the fully wooden lid is placed. These are five sequential process steps enriched with all the necessary details required for generating the manufacturing documentation.

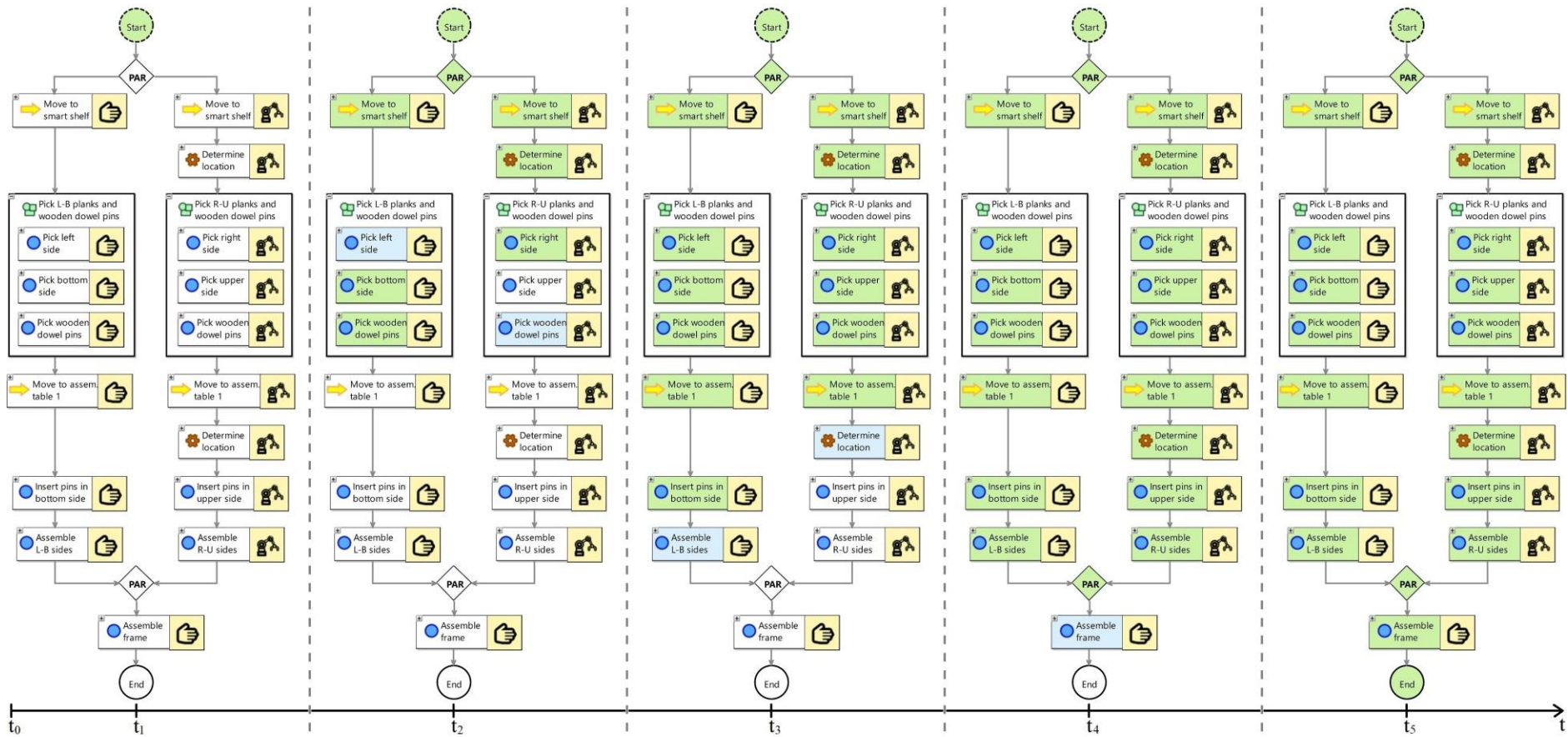


Figure 8.12. An example of monitoring the execution of the *Assemble frame* DetL sub-process model.

From the *Glue fabric and dividers and place lid* MasL sub-process model, BOM, BOMO, ASME FPC, and JBS documentation is generated. We start with the BOM document generated from the model. First, in Figure 8.13, we present a template used for the automatic generation of BOM documents. The BOM template represents a table with labels, presented as black-colored text, and variables, presented as red-colored text. Labels are the same in any BOM document, but variables depend on the values stored in a process model. The BOM template has a header with three variable sections:

- a checkbox for whether a process is proposed or present, depending on the value of the *isProposedProcess* attributed to a process;
- the name of the final product or parameter, based on the result of the *findFinalProductName* function; and
- a company name retrieved from a process model.

The *findFinalProductName* function iterates through all products and process parameters of the chosen product variation and finds the one that has the *isFinal* attribute with the *true* value. The body or the main part of the BOM document is represented by a hierarchy level, a product (i.e., an output product in a process step), a component (i.e., an input product in a process step) that is a part of the product, and quantity of the component in the product. To display all products and components in the correct hierarchy order, an algorithm in Documentation Generator finds the final product in a process model, searches back all related input and output products in the process model, and connects them in an array of product-component objects. Each product-component object has a hierarchy level, calculated by the algorithm, a product, and a related component, with the quantity of that component in the product. Each product-component object is displayed in the BOM document. In the first column, the *calculateDots* function returns dots depending on the hierarchy level and merges them with the hierarchy level of the product-component. A product name, a component name, and the component quantity are displayed in the following columns. The BOM document ends with its footer, containing the authors of the process model and the date of the process model creation, both values retrieved from the process model.

Based on the Documentation Generator algorithms and the presented template, BOM documents can be automatically created from process models. An example of the automatically generated BOM document from the *Glue fabric and dividers and place lid* MasL sub-process model (product variation 1.3) is presented in Figure 8.14. BOMO and ASME FPC documents are automatically generated in a similar manner; thus, we do not present their templates. Examples of BOMO and table formatted FPC of the same process model and product variation are presented in Figure 8.15 and Figure 8.16, respectively.

BOMO and ASME FPC documents have the same header and footer sections as the BOM document, and only the body section is different. Both BOMO and ASME FPC generators begin with the same algorithm that goes from the start of a process model to the end of the process model, detects each process step of the chosen product variation, and adds an appropriate identifier. The first column of BOMO and ASME FPC documents is the process step identifier, starting with the value ten, and each new process step identifier increments by ten. Next, a process step name and work center (i.e., an abstract work center, not the specific one that is to be added by Orchestrator in DetL models) columns are defined. Other columns are different in BOMO and FPC documents. BOMO documents have the following columns: estimated duration of creating a product in minutes, materials (i.e., input products in a process step) and their quantity needed to create the product, and the product (i.e., an output product in a process step) with its quantity. The following columns of ASME FPC documents are related to a process step type, i.e., whether a process step is an operation, an inspection, transportation, a delay, or storage. There is an empty symbol for each process step type, and they are filled whenever a process step is of that type. For example, the *Glue fabric* process step is an operation, but it also has a storage symbol filled, as some of the input products are to be retrieved from storage. Another example is the *Wait for glue to dry* process step, which is of the delay type only.



<b>BOM chart</b>	<input type="checkbox"/> IF «process.isProposedProcess» THEN Proposed Process checked <input checked="" type="checkbox"/> IF «!process.isProposedProcess» THEN Present Process checked		<b>Final Product</b>	<b>«process.company»</b>
			«findFinalProductName()»	
<b>Hierarchy Level</b>	<b>Product</b>	<b>Component</b>	<b>Quantity</b>	
«calculateDots(productComponent.level) + productComponent.level»	«productComponent.product.name»	«productComponent.component.name»	«productComponent.component.quantity»	
<b>Author</b>			<b>Date</b>	
«process.author»			«process.date»	

**Figure 8.13.** The BOM template used by Instruction Generator.

<b>BOM chart</b>	<input type="checkbox"/> Proposed Process <input checked="" type="checkbox"/> Present Process		<b>Final Product</b>	<b>FTN</b>
			Box	
<b>Hierarchy Level</b>	<b>Product</b>	<b>Component</b>	<b>Quantity</b>	
1	Box	Wooden lid	1	
1	Box	Frame with dried glue	1	
.2	Frame with dried glue	Frame with dividers	1	
..3	Frame with dividers	Single divider	1	
..3	Frame with dividers	Frame with single divider	1	
...4	Frame with single divider	Single divider	1	
...4	Frame with single divider	Frame with fabric	1	
....5	Frame with fabric	Fabric	1	
....5	Frame with fabric	Inspected frame with back side	1	
<b>Author</b>			<b>Date</b>	
Marko Vještica			2.1.2023.	

**Figure 8.14.** An automatically generated BOM document from the *Glue fabric and dividers and place lid* MasL sub-process model.

BOMO chart		<input type="checkbox"/> Proposed Process <input checked="" type="checkbox"/> Present Process		Final Product		FTN
Process Step ID	Process Step Name	Work Center	Duration (min/item)	Material	Product	
10	Glue fabric	Assembly table	0.30	1. Inspected frame with back side x1 2. Fabric x1	1. Frame with fabric x1	
20	Glue single divider	Assembly table	0.20	1. Frame with fabric x1 2. Single divider x1	1. Frame with single divider x1	
30	Glue single divider	Assembly table	0.20	1. Frame with single divider x1 2. Single divider x1	1. Frame with dividers x1	
40	Wait for glue to dry	Assembly table	0.50	1. Frame with dividers x1	1. Frame with dried glue x1	
50	Place wooden lid	Assembly table	0.10	1. Frame with dried glue x1 2. Wooden lid x1	1. Box x1	
Author				Date		
Marko Vještica				2.1.2023.		

**Figure 8.15.** An automatically generated BOMO document from the *Glue fabric and dividers and place lid* MasL sub-process model.

FPC chart		<input type="checkbox"/> Proposed Process <input checked="" type="checkbox"/> Present Process		Final Product			FTN
Process Step ID	Process Step Name	Work Center	Operation	Inspection	Transport	Delay	
10	Glue fabric	Assembly table	●	□	⇨	⏸	▼
20	Glue single divider	Assembly table	●	□	⇨	⏸	▼
30	Glue single divider	Assembly table	●	□	⇨	⏸	▼
40	Wait for glue to dry	Assembly table	○	□	⇨	■	▽
50	Place wooden lid	Assembly table	●	□	⇨	⏸	▼
Author				Date			
Marko Vještica				2.1.2023.			

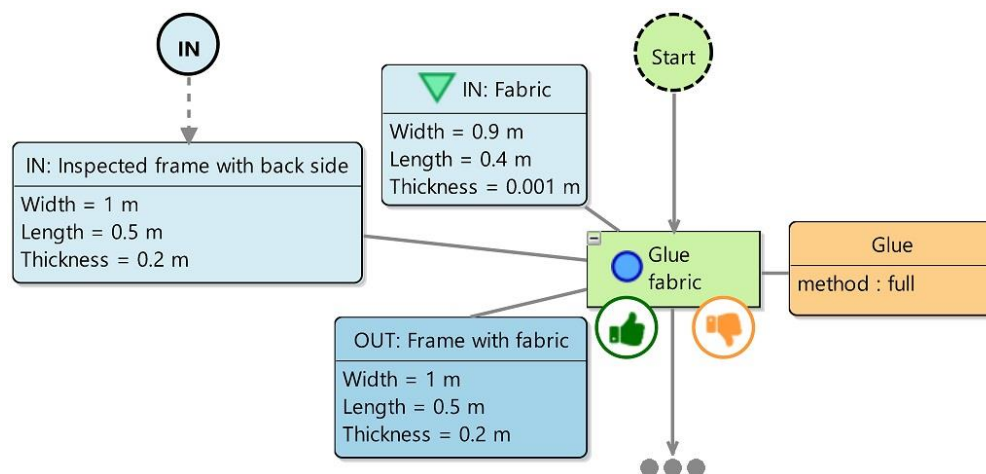
**Figure 8.16.** An automatically generated ASME FPC document from the *Glue fabric and dividers and place lid* MasL sub-process model.

Another automatically generated documentation type is JBS, whose documents can be used as work instructions for human workers. Before the automatic generation of the JBS document from the *Glue fabric and dividers and place lid* MasL sub-process model (product variation 1.3), we added two key points to the *Glue fabric* process step, presented in Figure 8.17 to demonstrate the usage of key points in JBS. Key points can be added to any process step, providing workers with additional information on how to perform a process step, but to make the model simple, we only present key points in a single process step. The first key point, with the icon of the thumb up, provides information that the glue areas must be clean to perform the process step successfully. Another key point, with the icon of thumb down, provides information that there must be no air bubbles after gluing the fabric, indicating what can go wrong. Key points can be added to a process model when the key point modeling layer is turned on.

An example of the automatically generated JBS document is presented in Figure 8.18. It has the same header and footer sections as previously presented document types, with the footer extended by indicators of what each key point type means. The main section has the following columns: an image of a process step, a process step identifier, such as the one presented in BOMO and ASME FPC, with a process step name, a key point icon, and a key point description. If there is no key point for a process step, the key point icon and description are replaced with the message that there are no key points for a process step.








The last document type that we automatically generate from process models is PFMEA. Potential production errors and failures must be modeled to generate a PFMEA document. Production errors can be specified both at the level of MasL and the level of DetL. Any error specified in the MasL process model needs to be propagated to the related DetL process model by Orchestrator. However, some errors need to be specified manually by process designers, as different process steps are added or changed in the DetL model, such as transportation and configuration steps. It is not always possible to automatically enrich process models with production errors, as Orchestrator does not always have the additional domain knowledge required to do it. For example, new production errors that can occur during transportation activities need to be added manually, as process designers possess the required domain knowledge, i.e., in which situations certain errors may occur.





To demonstrate the manual creation of production process errors and the automatic generation of the PFMEA documents, we added two errors into the *Assemble frame* DetL process model. In Figure 8.19, we present only part of the *Assemble frame* DetL model with two errors specified in the *Insert pins in bottom side* process step to keep the model readable while presenting all relevant language concepts. Products and capabilities are hidden from the diagram as well to make it more readable. There are two errors with local error handlers added to the *Insert pins in bottom side* process step, each error having the same corrective process steps. The first error is related to the case when, due to excessive use of force, the bottom-side plank cracks; thus, the plank cannot be



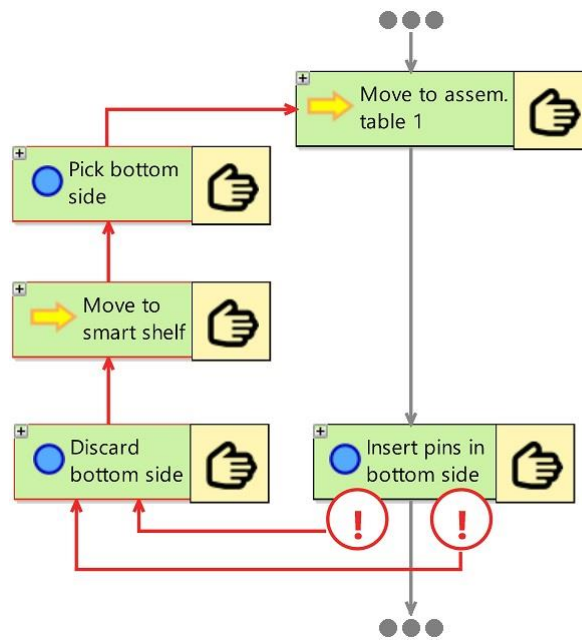
**Figure 8.17.** The *Glue fabric* process step key points.

used further. The second error relates to the case when holes are badly drilled; thus, wooden pins cannot be inserted appropriately. If any of these errors occur, the same sequence of corrective process steps is to be performed. The human worker needs to discard the bottom-side plank to the recycle bin near the assembly table, then move to the smart shelf, and pick a new bottom-side plank. After the human worker performs these error handling process steps, the process execution returns to the regular flow of process steps, meaning that the human worker needs to move to the first assembly table again and insert wooden dowel pins into the new bottom-side plank. The process execution then continues its regular flow. All these errors and error handlers are modeled within Error Handling Layer. The layer can be turned off so that the diagram becomes more readable compared to when the layer is turned on.

JBS chart	<input type="checkbox"/> Proposed Process <input checked="" type="checkbox"/> Present Process		Final Product	FTN
			Box	
Process Step		Key Points	Explanations	
	10. Glue fabric		Glue areas must be clean.	
			There must be no air bubbles after gluing the fabric.	
	20. Glue single divider	Process step has no key points.		
	30. Glue single divider	Process step has no key points.		
	40. Wait for glue to dry	Process step has no key points.		
	50. Place wooden lid	Process step has no key points.		
Author			Date	
Marko Vještica			2.1.2023.	

-  - Complete Job
-  - Complete Job Easier
-  - Break Job
-  - Injure Worker

**Figure 8.18.** An automatically generated JBS document from the *Glue fabric and dividers and place lid* MasL sub-process model.



**Figure 8.19.** The *Insert pins in bottom side* process step errors and error handler process steps.

In Figure 8.20, we present the automatically generated PFMEA document based on the errors modeled in the *Assemble frame* DetL sub-process model. The PFMEA document has the header and the main sections. The header section includes an item or a process name, a core team or process authors, and a key date or a process creation date. These values are automatically generated from the process model. Other attributes in the header section have placeholders to be filled in later. The main or the body section has columns related to potential failures, mostly described in Section 7.2.4. These columns are a process step name, failure (i.e., error) mode, effect, severity, classification, cause, occurrence, prevention, detection description, and detection value, all generated from an error. The last filled column is the Risk Priority Number (RPN), calculated in Documentation Generator by multiplying severity, occurrence, and detection values. RPN indicates whether an error is severe enough to take some recommended actions to reduce its severity or occurrence or become more easily detectable. A company can define a threshold for when to take some actions to reduce the effect of an error. The following columns of the document are empty as they require recommended actions to be applied and new values to be measured. At the time of the PFMEA generation, these recommendations have not been applied yet.

We developed Documentation Generators to generate documents of five different types automatically. Adding new generators to automatically generate documents of new types is possible. To achieve the automatic generation of documents of new types, the MultiProLan meta-model can be extended, new documentation templates created, and new algorithms, i.e., transformation rules, implemented. By extending our solution with new Documentation Generators, various standards and regulations requiring certain documents in a company can be fulfilled, and the number of manual tasks performed by process designers can be lowered.

**Potential  
Failure Mode and Effect Analysis  
Process FMEA**

FMEA number: \_\_\_\_\_

Page: 1 of 1

Item: Assemble frame

Process Responsibility: \_\_\_\_\_

Prepared By: \_\_\_\_\_

Model Year(s)/Programs(s): \_\_\_\_\_

Key Date: 2.1.2023.

FMEA Date: (Orig.) \_\_\_\_\_ (Rev.) \_\_\_\_\_

Core Team: Marko Vještica

Process Function Requirements	Potential Failure Mode	Potential Effect(s) of Failure	Severity	Classification	Potential Cause(s) / Mechanism(s) of Failure	Occurrence	Current Control		Detection	R.P.N.	Recommended Action(s)	Responsibility and Target Completion Date	Action Results				
							Prevention	Detection					Actions Taken	Severity	Occurrence	Detection	R.P.N.
Insert pins in bottom side	Bottom-side plank has been cracked	Bottom and left sides cannot be assembled	10	A	Excessive use of force	2	Add worker's training	-	1	20							
Insert pins in bottom side	Holes has been badly made	Bottom and left sides may not be properly assembled	8	B	Badly drilled holes	3	Add additional tests after bottom sides are made	Add additional check of wooden dowel pins after their insertion	3	72							

**Figure 8.20.** An automatically generated PFMEA document from the *Assemble frame* DetL sub-process model.

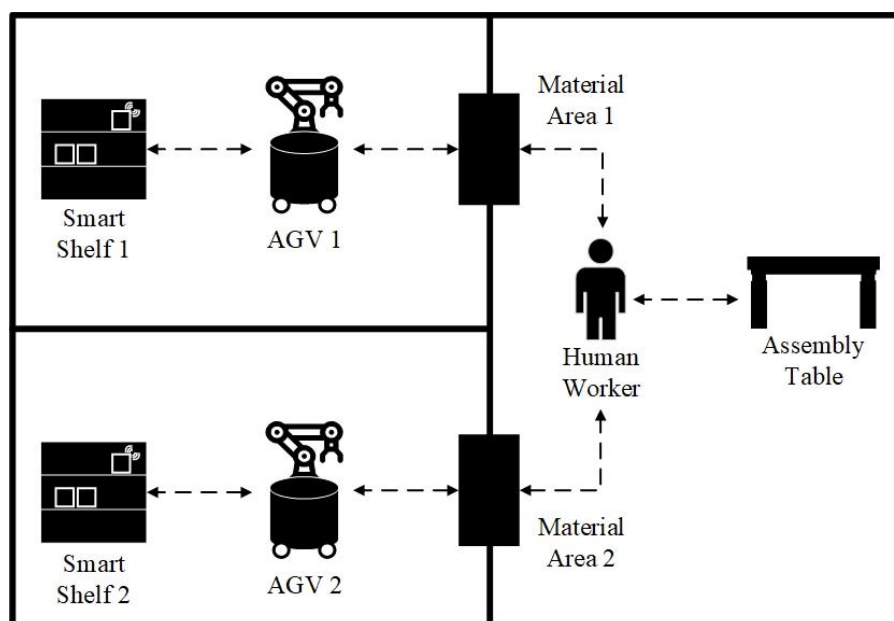
## 8.2 Demonstration Environment with LEGO® Bricks

The demonstration environment, with its topology presented in Figure 8.21, comprises a fenced area with two AGVs, two smart shelves, and two material areas. A human worker and an assembly table are located outside the fenced area. Each smart shelf stores LEGO® bricks of various colors and sizes needed for assembly. With the notion of AGV in Figure 8.21, we denote three types of resources that can be placed in the environment: a research-grade smart robot, a cobot, or an industrial mobile robot. An AGV moves inside the fenced area, picking LEGO® bricks from a smart shelf and placing them on a material area. Therefore, AGVs are used to retrieve bricks from storage and bring them close to the human worker in charge of the assembly, creating a human-machine collaboration through material handling. LEGO® bricks of different types are stored on smart shelves, and depending on the final product, required bricks may be retrieved from both shelves if needed. The human worker picks bricks from material areas and assembles them on the assembly table. The table has a brick pedestal on which LEGO® bricks can be assembled. Both material areas are positioned between AGV and human worker areas, making it possible for AGVs and the human worker to pick and place bricks in the area while adhering to all necessary safety protocols and measures in place. Even if AGVs are cobots, having multiple sensors to detect objects around them, we separated cobots from the worker with the fence, as we want to lower the influence of complex safety mechanisms on the demonstration.

In the following subsections, we discuss two test scenarios performed in the demonstration environment (see Section 8.2.1). Afterward, we present MasL and DetL process models of assembling a flag out of LEGO® bricks in such an environment (see Section 8.2.2). Production process models of assembling a flag out of LEGO® bricks were already presented in [13], and in this thesis, they are discussed in more detail.

### 8.2.1 Test Scenarios Performed in the Demonstration Environment

The presented assembly demonstration environment is created to test the core functionalities of our MD solution and MultiProLan. Process Modeling Tool utilizing MultiProLan is used to create production process models of the brick assembly. From such models, machine-specific instructions are generated and sent to AGVs. Also, human-readable instructions are generated and sent to the



**Figure 8.21.** The scheme of assembly demonstration environment.



human worker when required. The human worker uses a tablet or a smart watch to receive instructions and send feedback. Therefore, in this use case, multiple components of our solution are used:

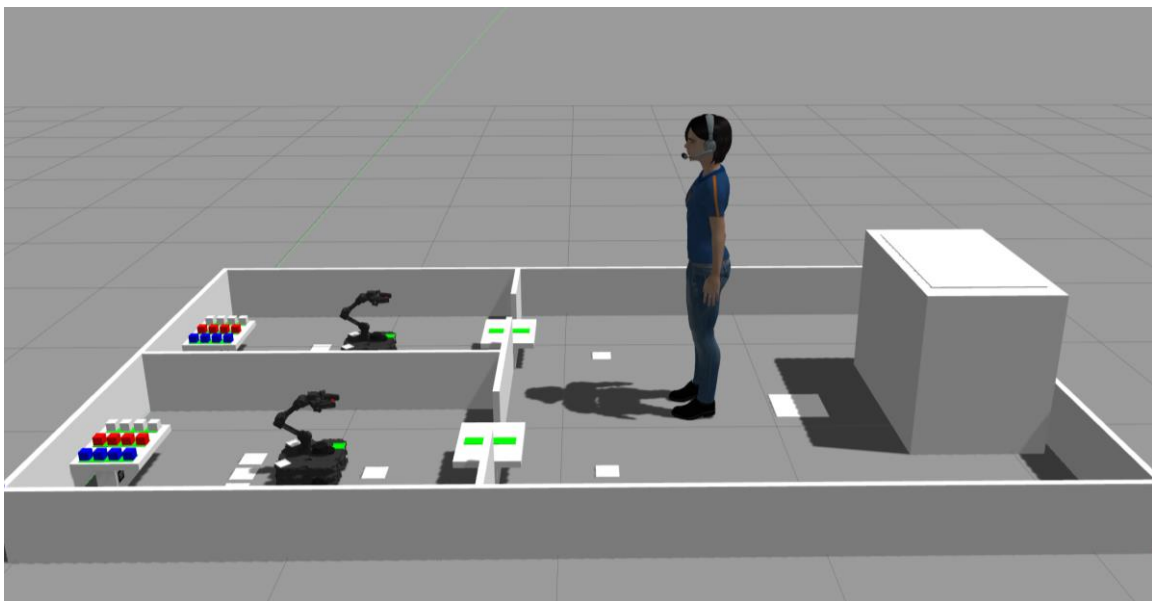
- Process Modeling Tool and MultiProLan are used by process designers to create MasL process models;
- Orchestrator is used to automatically transform MasL into DetL process models;
- Instruction Generator is used to automatically generate high-level instructions from DetL process models;
- Digital Twin with embedded protocol transformation components is used to transform high-level instructions into machine-specific and human-readable instructions; and
- the tablet or smart watch applications are used by the human worker to receive instructions and provide feedback.

It should be pointed out that Orchestrator, Digital Twin with embedded protocol transformation components, and the tablet and smart watch applications are developed as part of related research and development efforts as described in Pisarić et al. [4,5], but not as part of this research. Instead, we consider them as black box components in this thesis.

The Digital Twin visualization, presented in Figure 8.22, is developed by using the Gazebo simulator [229,230]. It reflects the physical, real demonstration environment in which we tested our solution. There are two test scenarios we used in such a demonstration environment.

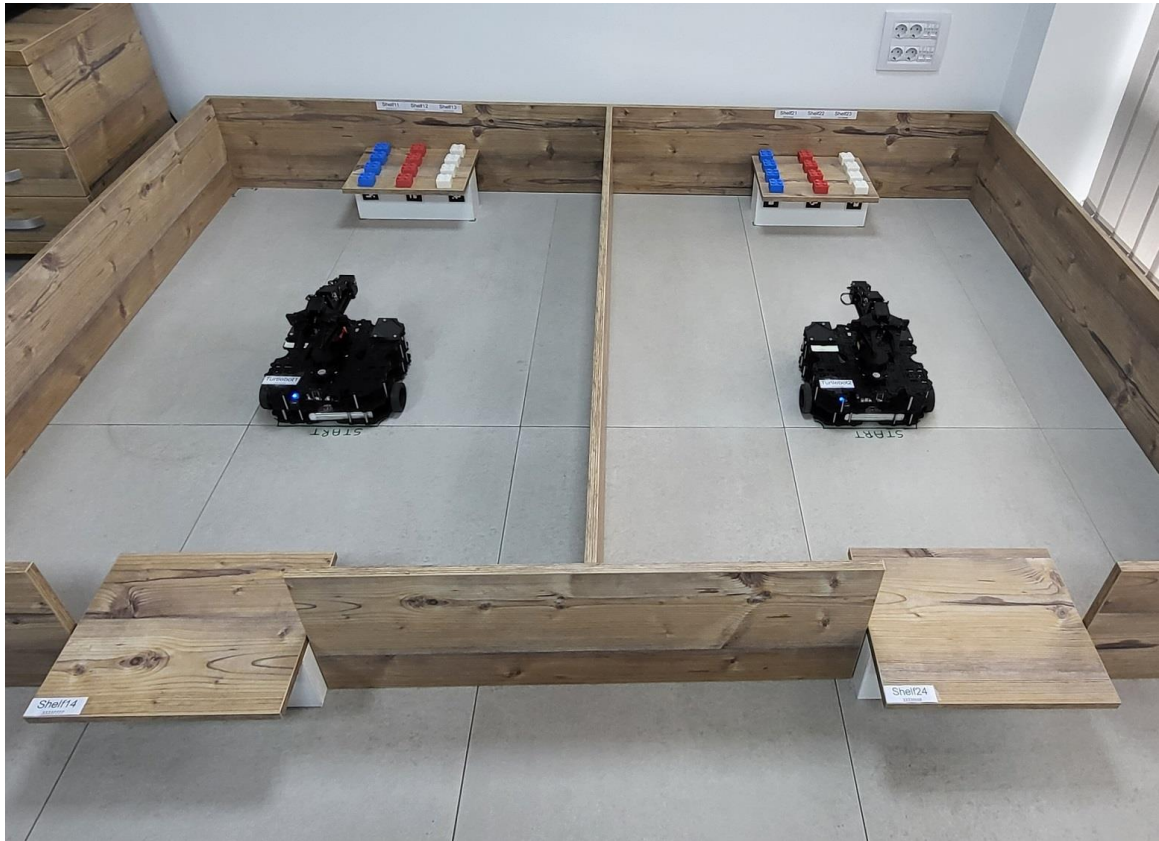
The first test scenario includes research-grade smart robots as AGVs, presented in Figure 8.23. These robots use ROS [227,228] and each high-level instruction sent to Digital Twin is then transformed into ROS-based instructions specific to such robots. Research-grade smart robots are used to pick and place required LEGO® bricks in this test scenario, but we also tested them to assemble objects from LEGO® bricks.

The second test scenario includes industrial mobile robots as AGVs, presented in Figure 8.24. These robots are used in the real production of assembling various products. They can pick, place, and assemble different objects and we tested them by using our solution. The solution generated high-level instructions, which are then transformed into specific ROS instructions understandable by a digital twin of a specific robot. The digital twin then transforms these ROS instructions into machine-specific instructions. The robots do not have a machine vision module installed; thus, they must calibrate themselves each time they move between shelves and material areas to determine their position. Both research-grade smart robots and industrial mobile robots receive instructions



**Figure 8.22.** The digital twin of assembly demonstration environment.

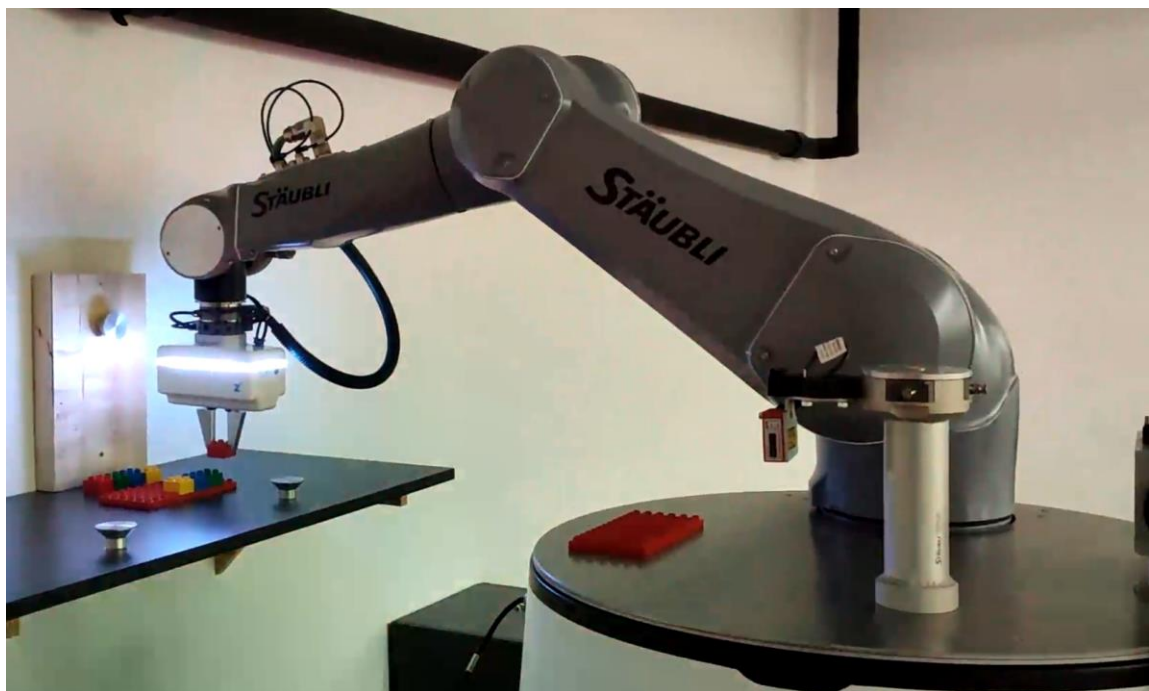




**Figure 8.23.** The assembly demonstration environment with research-grade smart robots.

wirelessly via an appropriate protocol and in a similar manner they send the execution status and feedback to our MD system.

A human worker that is a part of both test scenarios, receives instructions through a tablet or a smart watch, with process steps descriptions, images, audio, and videos. After each instruction



**Figure 8.24.** The assembly demonstration environment with an industrial mobile robot.

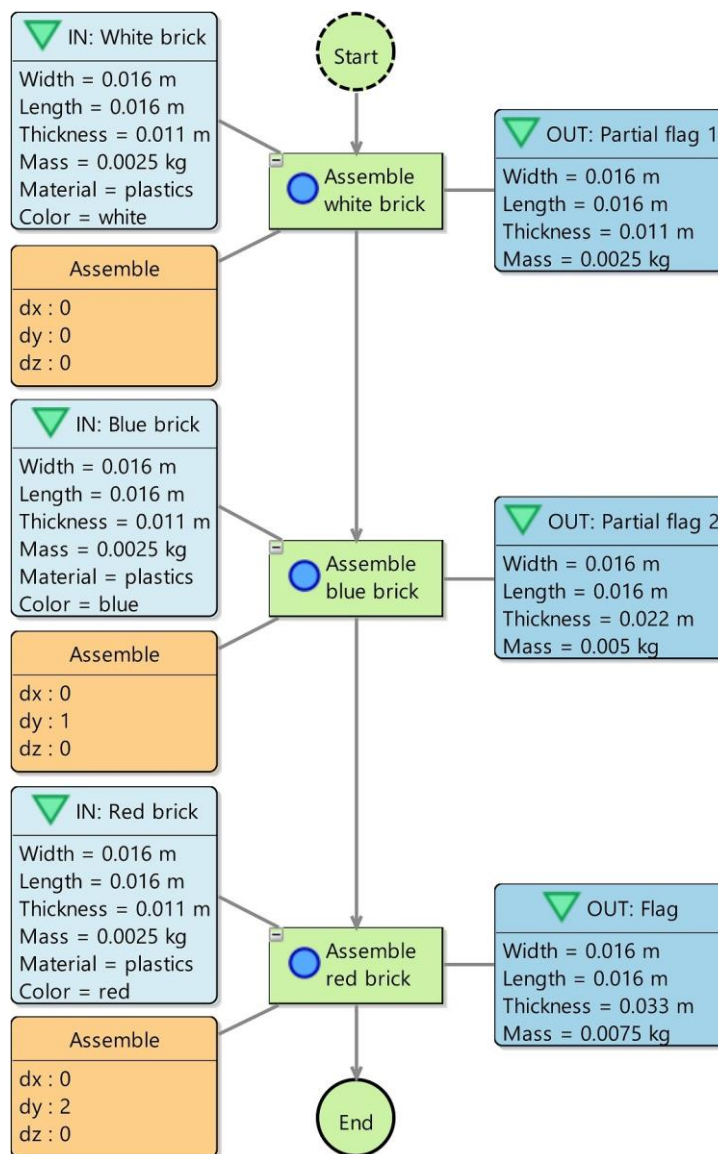
execution, the human worker must confirm the execution status through the same device the instructions are received.

In both test scenarios, the same process models are used to create objects from LEGO® bricks. In the following section, we present MasL and DetL models of assembling a flag from LEGO® bricks.

### 8.2.2 Assembling a Flag from LEGO® Bricks

In this section, we present an example of assembling a flag from LEGO® bricks. The flag is to be made using three bricks of different colors. They are assembled one on top of another, making a sequence of activities – the bottom brick must be first assembled on the brick pedestal, then the middle brick on the top of the bottom brick, and, in the end, the upper brick on the top of the middle brick.

The MasL process model of assembling a flag, created by a process designer, is presented in Figure 8.25. This process model represents a sequence of assembly process steps, starting with the



**Figure 8.25.** The MasL process model of assembling the red-blue-white flag out of LEGO® bricks.

assembly of a white LEGO® brick. The *Assemble white brick* process step has an input product representing a white brick that needs to be retrieved from storage. It has the following constraints: (i) dimensions, i.e., width, length, and thickness; (ii) mass; (iii) material; and (iv) color. These constraints are considered by Orchestrator when it matches appropriate resources with assembly process steps. In both test scenarios, AGVs can pick, place, and assemble LEGO® bricks having the presented constraints. The *Assemble white brick* process step has the *Assemble* capability with parameters representing relative coordinates where to assemble a brick related to the brick pedestal. Therefore, the white brick needs to be assembled on the center of the brick pedestal. The output product of the *Assemble white brick* process step is the partially assembled flag, having only the white LEGO® brick. The storage icon on the output product means that the product is to be placed in storage, or in this case, on the top of the brick pedestal located at Assembly Table. The constraints of such an output product are dimensions and mass equivalent to the ones of the white brick.

The following process step in the assembly sequence is the assembly of a blue LEGO® brick. The *Assemble blue brick* process step has a blue brick as an input product that needs to be retrieved from storage, similar to the previous process step input product, with the value of color constraint as the only difference. The process step has the *Assemble* capability with parameters similar to the capability of the previous process step. The only difference is in the *dy* relative coordinate, meaning that the blue brick is to be assembled on the previously assembled white brick. The output product of the *Assemble blue brick* process step is the partially assembled flag, having blue and white LEGO® bricks, placed on the brick pedestal. The width and the length of this partially assembled flag are the same as it is in the output product of the previous process step. However, as the blue brick is added on top of the previously partially assembled flag, the thickness and the mass of the output product are doubled.

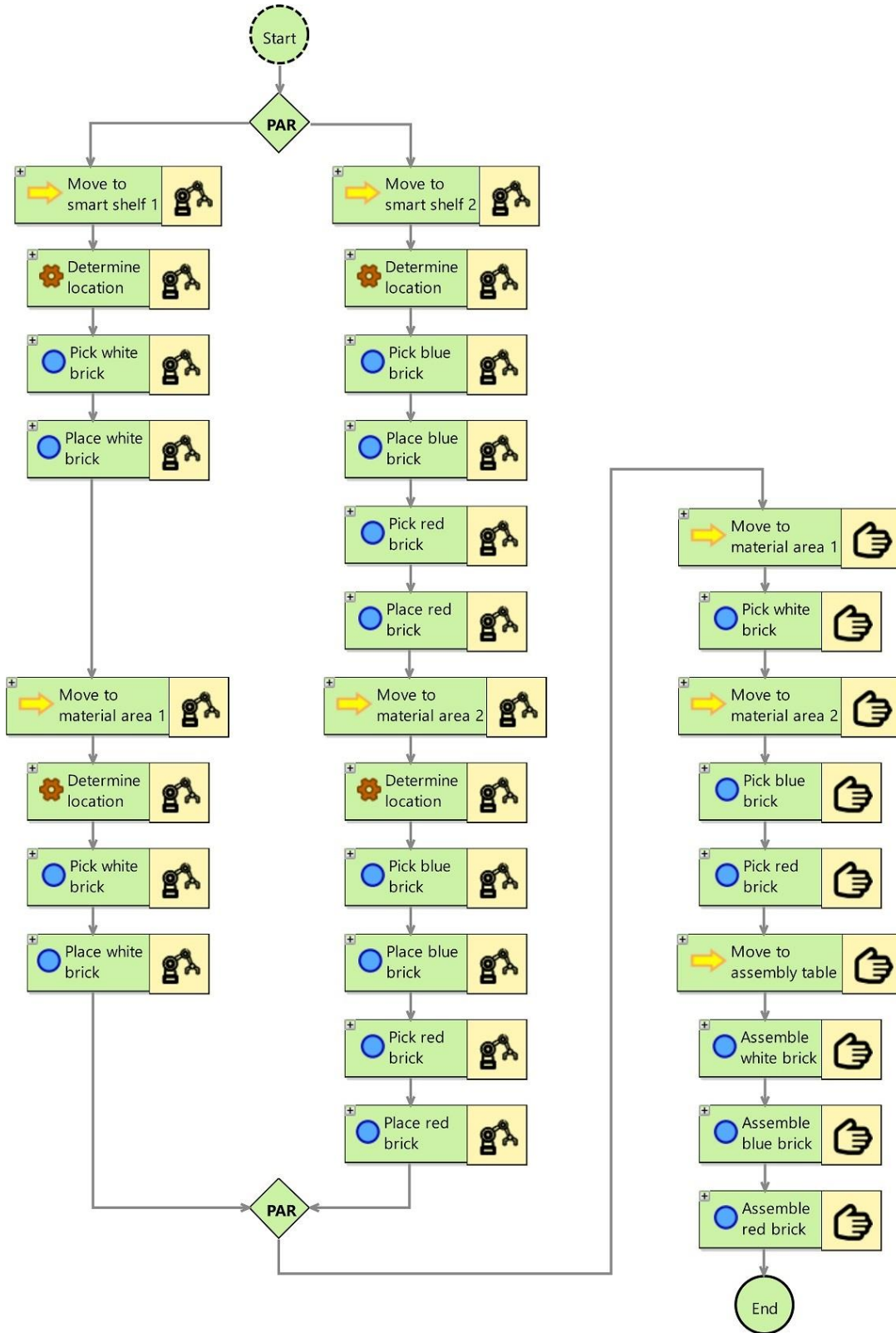
The final process step in the assembly sequence, before the process ends, is the assembly of a red LEGO® brick. The *Assemble red brick* process step has a red brick as an input product that needs to be retrieved from storage, similar to the input products of the previous two process steps. The process step has the *Assemble* capability, with parameters similar to the ones of the previous two process step capabilities, differencing only in the *dy* relative coordinate, meaning that the red brick is to be assembled on top of the previously assembled blue brick. The output product of the *Assemble red brick* process step is the final product – the red-blue-white flag out of LEGO® bricks, placed on the brick pedestal. In comparison to the output product of the previous process step (i.e., the partially assembled blue-white flag), the width and the length of the red-blue-white flag are the same, but the thickness and the mass are higher as the red brick is added on the top of the previously partially assembled flag.

To execute a process automatically, a MasL process model needs to be transformed into a DetL process model related to the specific production system in which the process is to be executed. In our case, the production system is one of the discussed test scenarios in the demonstration environment. The previously modeled MasL process model is sent to Orchestrator to automatically create the DetL process model, presented in Figure 8.26. Products and capabilities are specified in the DetL process model but are hidden from the diagram to make it more readable. In the rest of this section, we discuss the DetL process model and production of the final product.

In our demonstration environment, for this particular case of assembling the red-blue-white flag out of LEGO® bricks, white bricks are stored on Smart Shelf 1, while blue and red bricks are stored on Smart Shelf 2. Therefore, AGV 1 needs to retrieve a white brick from Smart Shelf 1, while AGV 2 needs to retrieve blue and red bricks from Smart Shelf 2. The DetL process model starts with a parallelism gate, representing the retrieving of LEGO® bricks done in parallel by AGVs. In the left-hand side branch between parallelism gates, the retrieving of the white brick is modeled. First, AGV 1 needs to move to Smart Shelf 1 and determine its location. After the configuration activity is finished, AGV 1 needs to pick a white brick and place the brick in its local storage. AGV 1 then moves to Material Area 1, determines its location, picks the white brick from its local storage, and places the brick on Material Area 1. In the right-hand side branch between parallelism gates, AGV 2 needs to pick blue and red bricks from Smart Shelf 2 and place them on Material Area 2. These activities are similar to the ones in the left-hand side branch, with the only

difference being that AGV 2 needs to pick two bricks after moving to Smart Shelf 2 and place both of them after moving to Material Area 2.

After red, blue, and white bricks are placed on the material areas, the human worker needs to move to Material Area 1, pick the white brick, then move to Material Area 2, and pick blue and red



**Figure 8.26.** The DetL process model of assembling the red-blue-white flag out of LEGO® bricks.



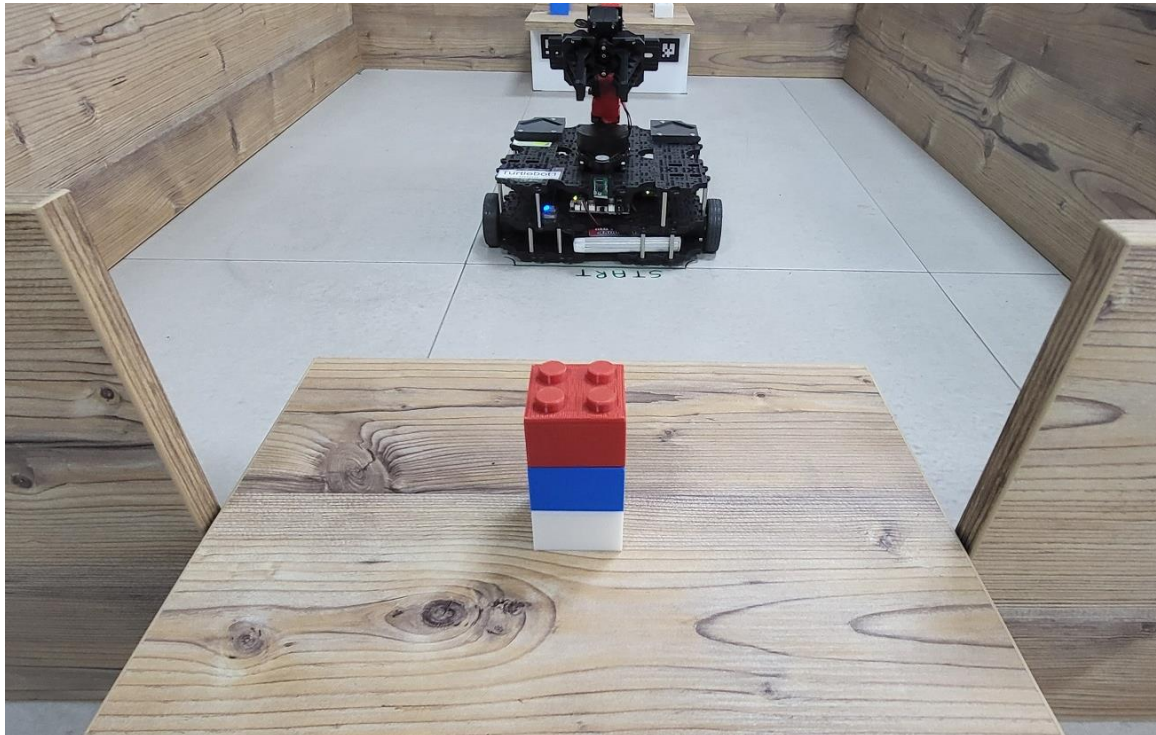
bricks. The human worker then needs to move to Assembly Table and assemble white, blue, and red bricks in that order, producing the red-blue-white flag out of LEGO® bricks.

By using Instruction Generator, high-level instructions can be automatically generated from the DetL process model, after they are transformed into machine-specific instructions for AGVs and human-readable instructions for the human worker. All these high-level and resource-specific instructions are similar to the ones presented in Section 8.1.3, and thus they are not presented in this section. After executing resource-specific instructions by the resources in our demonstration environment, the red-blue-white flag out of LEGO® bricks is produced, as presented in Figure 8.27.

### 8.3 Summary

In this section, we presented the application of our MD solution with a focus on MultiProLan and Process Modeling Tool. Thus, we aimed to make a proof-of-concept of our solution for production process modeling and execution. We modeled various production processes in the assembly industry, and some of these models were presented in this section through the use cases. Based on our experience in production process modeling, one of the biggest challenges when using a modeling language with concrete graphical syntax is the readability of process diagrams. When there is a relatively complex production process model with many process steps, or there are too many errors that need to be specified, or there are numerous product variations present in a process model, the process diagram becomes overwhelmed with details. We managed to deal with the model diagram complexity through different MultiProLan concepts created for such a purpose and through various mechanisms implemented in Process Modeling Tool, as discussed in this section. The whole MD solution is tested in our demonstration environment, but to model and execute more complex production processes, especially in the industry outside the assembly domain, the MD solution, including MultiProLan and Process Modeling Tool, would need to be further developed.

Based on the experience we gathered while modeling various production processes in different assembly use cases, we believe that MultiProLan and Process Modeling Tool can be used in practice as:



**Figure 8.27.** The produced red-blue-white flag out of LEGO® bricks.

- the main concepts needed for production process modeling are covered;
- the language and the tool are easy to use;
- their process models are readable and understandable;
- users are protected from making errors during modeling; and
- users are provided with mechanisms to deal with the model diagram complexity.

However, besides applying MultiProLan and Process Modeling Tool in the assembly use cases, providing a proof-of-concept and presenting our experience, they must be tested by other users. Therefore, MultiProLan and Process Modeling Tool were evaluated by users from various user groups. In the following section, we present the results of the evaluation process, in which users tested whether MultiProLan and Process Modeling Tool can be used in practice by investigating the following quality characteristics: functional suitability, usability, reliability, expressiveness, and productivity. We gathered useful feedback from the evaluation participants, which helped us improve MultiProLan and Process Modeling Tool.

## 9 Evaluation of MultiProLan and Process Modeling Tool

Users of various profiles need to be included in the evaluation process to systematically evaluate a language and a modeling tool and gather broader feedback. According to Salman et al. [242], when applying new technology or a new approach in a software engineering experiment, there is no significant difference whether students or professionals are involved. Thus, we included researchers and students from the academic community, as well as process engineers and software developers from the industry in the evaluation process. Each participant type has different skills related to the production process modeling and tool development. We created an experiment in which participants evaluated MultiProLan and Process Modeling Tool based on their functional suitability, usability, reliability, expressiveness, and productivity characteristics. We also evaluated implementation-related characteristics, such as maintainability, extensibility, reusability, and integrability. Quality characteristics evaluated by participants are analyzed, and conclusions on how to improve MultiProLan and Process Modeling Tool are presented in this section.

In this section, we discuss: (i) the experiment objective and hypothesis (see Section 9.1); (ii) the experiment participants (see Section 9.2); (iii) the experiment preparation and execution (see Section 9.3); (iv) the experiment results and data analysis (see Section 9.4); (v) an overview of other quality characteristics (see Section 9.5); (vi) threats to validity (see Section 9.6); and (vii) a summary of the evaluation process (see Section 9.7). The evaluation results were already presented in [11].

### 9.1 Experiment Objective and Hypothesis

MultiProLan is used in a small-scale assembly use case [13] and is developed iteratively based on the process designers' feedback. As we plan to apply MultiProLan in additional industry use cases, various users, such as students, researchers, software developers, and process engineers, have participated in the evaluation process.

To evaluate MultiProLan and Process Modeling Tool, we applied the Framework for Qualitative Assessment of Domain-specific languages (FQAD) [101] and chose a subset of the proposed characteristics already being applied to a DSML evaluation [243]. The framework has been used to determine the evaluator's perspective on fundamental DSL quality characteristics and to guide the evaluator through the evaluation process. These characteristics are adapted and integrated from the international system and software quality standard ISO/IEC 25010:2011 [244].

Participants of the experiment evaluated their experience in using MultiProLan based on the following characteristics:

- **functional suitability** – a degree to which MultiProLan supports developing solutions to meet the needs of the production process modeling domain;

- **usability** – a degree to which users can use MultiProLan to achieve specified goals;
- **reliability** – a property of MultiProLan that aids in productizing reliable programs, i.e., model checking ability;
- **expressiveness** – a degree to which a problem-solving strategy can be mapped onto a program naturally; and
- **productivity** – a characteristic related to the number of resources that users spend to achieve specified goals.

Other characteristics proposed in FQAD also need to be analyzed; however, these characteristics require implementation knowledge. Thus, they are discussed and evaluated by the authors of MultiProLan. These characteristics are:

- **maintainability** – the degree to which MultiProLan is easy to maintain;
- **extensibility** – the degree to which MultiProLan has general mechanisms for users to add new features;
- **reusability** – the degree to which MultiProLan's constructs can be used in more than one language; and
- **integrability** – the degree to which MultiProLan is amenable to integration with other languages.

This experiment was created to evaluate the user experience with MultiProLan based on the presented characteristics. The experiment's goal is to check whether MultiProLan can be used in practice in the domain of Industry 4.0 assembly production, with positive user experience from different user groups. Thus, we investigated the following hypothesis.

**EH<sub>null</sub>** – *MultiProLan can be used in practice as it has all the following quality characteristics: functional suitability, usability, reliability, expressiveness, and productivity.*

**EH<sub>alt</sub>** – *MultiProLan cannot be used in practice as it does not have one or more of the following quality characteristics: functional suitability, usability, reliability, expressiveness, or productivity.*

We may confirm the EH<sub>null</sub> hypothesis if there is more than 50% of positive feedback from the evaluation participants per each quality characteristic. If there is a single quality characteristic that does not have more than 50% of positive feedback, we may reject the EH<sub>null</sub> hypothesis and confirm the EH<sub>alt</sub> hypothesis. The way that positive feedback is calculated is discussed in Section 9.3.

## 9.2 Experiment Participants

At the time of the experiment execution, all participants had at least a B.Sc. in computer science. There were 25 participants in the experiment, and they can be categorized into four disjunct groups:

- process engineers – participants who are experienced in production process modeling;
- software developers – participants who are experienced in the engineering of various modeling tools;
- researchers – participants with Ph.D. in computer science, most of which are experienced in business process modeling; and
- students – M.Sc. and Ph.D. students with previous experience in process modeling.

In the first group, there were 2 participants who were experts in production process modeling. These two process engineers are not the same as those who participated in language development. In the second group, there were 5 participants who tested and developed modeling tools in the industry. In the third group, there were 6 participants who investigated different areas of computer science and possessed experience in business process modeling. In the fourth group, there were 12 students – 5 M.Sc. students and 7 Ph.D. students who all had experience in process modeling using



BPMN, UML, and PN. All the students attended master's courses covering business process modeling and domain-specific modeling and languages.

### 9.3 Experiment Preparation and Execution

Due to the pandemic of COVID-19, we had to perform the evaluation experiment online via video conferencing tools. We performed the experiment in a controlled environment where each participant did the evaluation separately. The whole experiment lasted about 90 minutes per person. The evaluation process was divided into five phases, and each participant had:

- an overview of the experiment (about 10 minutes);
- an introduction to the production process modeling (about 10 minutes);
- a document that represents the MultiProLan and Process Modeling Tool tutorial (about 15 minutes to read the document);
- a document that includes tasks that participants need to accomplish (about 45 minutes to finish the tasks); and
- a questionnaire to evaluate MultiProLan (about 10 minutes).

The tutorial was composed of two parts: (i) the MultiProLan's modeling concepts; and (ii) the Process Modeling Tool usage. As the participants were unfamiliar with MultiProLan, we had to describe its basic modeling concepts in the form of an online live tutorial. Afterward, the Process Modeling Tool user interface was described, and a brief description of how to use the tool was given to the participants.

The tasks that participants needed to accomplish were given in the form of functional requirements, presented in Appendix A, alongside the solution of the tasks. The requirements were presented as a textual description of a wooden box production process. The box is composed of four wooden planks representing different sides of the box and a thin wooden back side. The four wooden planks need to be assembled into a frame using wooden pins, and the wooden back side needs to be hammered into the frame, creating the box. First, the left-bottom and right-upper sides are assembled in parallel, and then these two intermediate parts are assembled into the frame. Collaborative activities of holding the frame and hammering the back side into the frame need to be executed after the frame is assembled, creating the box. Afterward, the box needs to be inspected for any defects, and if a defect is found, the box needs to be discarded. Otherwise, the box needs to be stored. We chose the wooden box production example as it is composed of nearly all the basic modeling concepts of MultiProLan and is simple enough so that participants who are inexperienced with our tool can finish the tasks in a relatively short time. The tasks were composed of the following:

- creating a MasL process model of the wooden box production;
- adding four different errors in the previously created MasL process model; and
- creating a part of a DetL process model of the wooden box production.

Participants modeled only two parallel branches in the DetL model example, as there are many details in DetL models, and it would be time-consuming to model them all. Also, all the DetL modeling concepts are included in these two parallel branches, i.e., resources, storage, transportation and configuration process steps, and the rest of the DetL model would be modeled in a similar way.

Participants had no time limit to finish modeling the example, as we did not measure the modeling time but the quality of MultiProLan and Process Modeling Tool based on the described characteristics. During the modeling tasks, participants could ask us any question related to MultiProLan and the example; however, we did not interfere with how the participants modeled the given processes.

After participants finished their modeling tasks, they were asked to complete the online questionnaire, whose questions are presented in Appendix A. We guaranteed that the gathered

answers were anonymized and that their data would not be given to third parties. The participants agreed to these terms and, before answering the questions from the questionnaire, were asked to fill in their name and e-mail address, so we could contact them if we needed any additional information. Then, they were asked to choose the participant group they belonged to. The questionnaire was composed of the following sections: (i) participant's experience; (ii) MultiProLan's quality characteristics; and (iii) free comments.

In the first section, participants were asked about their experience in business process modeling, production process modeling, and Computer Aided Software Engineering (CASE) tools in general. We also asked them to populate which languages and CASE tools they have used. As there were different groups of users, this information was useful when we analyzed the experiment results and put the answers in a broader context. For each experience-related question, participants could choose one answer on a five-level Likert scale ranging from 1 – *Inexperienced* to 5 – *Experienced*.

The quality characteristics we evaluated with participants were functional suitability, usability, reliability, expressiveness, and productivity. Each of these characteristics represents one of the groups of questions that are defined in FQAD. Participants also needed to choose one answer to each question on a five-level Likert scale. For the functional suitability characteristic, this scale ranged from 1 – *Very low* to 5 – *Very high*; for the usability, reliability, and expressiveness characteristics, this scale ranged from 1 – *Strongly disagree* to 5 – *Strongly agree*; and for the productivity characteristic, this scale ranged from 1 – *Long* to 5 – *Short*, as it involved the evaluation of specification time.

In the third section, participants could leave free-form comments on everything we did not ask them and make suggestions on how to improve MultiProLan and Process Modeling Tool, which happened to be particularly useful.

As stated at the end of Section 9.1, we may confirm the  $EH_{null}$  hypothesis if there is more than 50% of positive feedback from the evaluation participants per each quality characteristic. To calculate the percentage of positive feedback for each quality characteristic, we calculated the percentage of positive feedback answers on each question related to the characteristic. By the positive feedback answer, we denote each answer to a question having a value of 4 or 5 on the five-level Likert scale.

## 9.4 Experiment Results and Data Analysis

After the participants finished the evaluation process by filling in the questionnaire, we summarized their answers in Table 9.1. In this table, all the questions with answers presented on a five-level Likert scale are shown, alongside the percentage of each answer. The questions in Table 9.1 are divided into the following sections: (i) participants' experience; and (ii) MultiProLan's quality characteristics: functional suitability, usability, reliability, expressiveness, and productivity. Further in this section, we discuss the evaluation results. First, we discuss the answers and comments gathered from the questionnaire. Participants wrote various comments, but we only discuss the ones that constructively criticized MultiProLan and gave us suggestions on what may be improved. We do not discuss comments that praised MultiProLan or criticized the user interface related to the Sirius framework. Then we discuss statistical analysis done on the data gathered from the questionnaire.

**Table 9.1.** The questionnaire statistics.

<b>Experience</b>					
<b>Question</b>	<b>Inexperienced</b>	<b>Relatively inexperienced</b>	<b>Medium experienced</b>	<b>Relatively experienced</b>	<b>Experienced</b>
How would you describe your previous experience in designing business processes?	20%	12%	40%	20%	8%
How would you describe your previous experience in designing production processes?	56%	20%	16%	4%	4%
How would you describe your previous experience with Computer Aided Software Engineering (CASE) tools for modeling?	8%	20%	28%	28%	16%
<b>Functional suitability</b>					
<b>Question</b>	<b>Very low</b>	<b>Low</b>	<b>Medium</b>	<b>High</b>	<b>Very high</b>
How would you describe the scope of production process domain concepts and scenarios that can be expressed in MultiProLan?	0%	0%	0%	64%	36%
How would you describe MultiProLan's level of suitability for the production process specification?	0%	0%	4%	64%	32%
<b>Usability</b>					
<b>Question</b>	<b>Strongly disagree</b>	<b>Disagree</b>	<b>No opinion</b>	<b>Agree</b>	<b>Strongly agree</b>
MultiProLan language elements are understandable.	0%	4%	0%	36%	60%
The concepts and symbols of MultiProLan are learnable and rememberable.	0%	0%	0%	20%	80%
MultiProLan has capability to help users achieve their tasks in an acceptable number of steps.	0%	0%	8%	52%	40%
MultiProLan is appropriate for your needs.	0%	0%	48%	16%	36%
MultiProLan Eclipse environment has elements that facilitate to operate and control the language.	0%	0%	20%	52%	28%
MultiProLan has graphical symbols that are good looking/attractive.	0%	8%	0%	44%	48%
By separating MasL and DetL models, users can model production processes easier.	0%	0%	4%	32%	64%
By creating different modeling layers, models become more readable.	0%	0%	0%	12%	88%
<b>Reliability</b>					
<b>Question</b>	<b>Strongly disagree</b>	<b>Disagree</b>	<b>No opinion</b>	<b>Agree</b>	<b>Strongly agree</b>
MultiProLan protects users against making errors.	0%	0%	16%	60%	24%
MultiProLan has a functional model validator.	0%	4%	28%	32%	36%
<b>Expressiveness</b>					
<b>Question</b>	<b>Strongly disagree</b>	<b>Disagree</b>	<b>No opinion</b>	<b>Agree</b>	<b>Strongly agree</b>
A problem-solving strategy can be mapped into a specification easily.	0%	0%	0%	64%	36%
MultiProLan is at the right abstraction level such that it is not more complex or detailed than necessary.	0%	0%	0%	52%	48%
MultiProLan provides one and only one good way to express every concept of interest.	0%	12%	20%	48%	20%
<b>Productivity</b>					
<b>Question</b>	<b>Long</b>	<b>Relatively long</b>	<b>Medium</b>	<b>Relatively short</b>	<b>Short</b>
How would you describe the specification time of a production process model with MultiProLan?	0%	4%	28%	52%	16%

### 9.4.1 Questionnaire Results

**Experience.** The participants were mostly medium experienced in business process modeling, inexperienced in production process modeling, and relatively experienced with CASE tools. For business process modeling languages, most of the participants stated that they were using BPMN [30] (17 participants), UML [43] (9 participants), flowcharts [245] (5 participants), PN [46,47] (4 participants), and EPC [53] (1 participant), 3 of them did not answer, and 2 of them answered with "none". As participants were not experienced in production process modeling, 12 of them did not answer whether they had used any production process modeling language, and 5 of them answered with "none". Others stated that they were using BPMN (2 participants), flowcharts, PN, process sheets [209], FPC [25], BOMO [24], UML, Piping and Instrumentation Diagram (P&ID) [246], and informal techniques (there was only 1 participant for each of these answers). For the CASE tools, most participants answered that they were using System Analysis Program Development (German: Systemanalyse Programmentwicklung, SAP) PowerDesigner [247] (15 participants), and some of them were using Oracle SQL Developer Data Modeler [248] (6 participants), Camunda BPM [249] (3 participants), MagicDraw [250] (3 participants), Oracle Designer [251], IIS\*Studio Development Environment [167], Microsoft Visio [252], Yaoqiang BPMN Editor [213], AutoCAD [253], Enterprise Architect [254] (2 participants for each of these answers) and so on. Only 1 participant answered the question with "none", and only 1 participant did not answer the question.

**Functional suitability.** Most participants evaluated MultiProLan's functional suitability as high (64% for both questions) or very high (36% and 32% for the first and the second question, respectively). They have evaluated that the production process domain can be specified by MultiProLan and that MultiProLan is suitable for production process modeling. However, in the free comment section, one of the process engineers stated that MultiProLan should support a few additional features if it is to be applied in the process industry. The same process engineer stated that production process modeling should also cover safety aspects, especially if there are human-machine collaborations. Another comment was that capabilities should not be arbitrarily written but read from a dictionary, as users may write a capability using another natural language or even slang. Thus, none of the resources could be matched with a process step that the capability belongs to. Later, as described in this thesis, we created a capability repository to solve such an issue.

**Usability.** Different questions were specified for the usability characteristic. Participants mostly agreed or strongly agreed that MultiProLan's elements are understandable (36% agreed, 60% strongly agreed) and that MultiProLan's concepts and symbols are easily learned and easily remembered (20% agreed, 80% strongly agreed). In the free comment section, a few participants left a negative comment on collaboration modeling. It was confusing where to put an output product of a collaboration between process steps. We plan to improve the modeling concept of collaboration in the future. One of the participants suggested that we could create a textual syntax for MultiProLan and test this syntax, as some of the users may find it easier to use. We will consider the creation of MultiProLan's textual syntax in the future.

As Process Modeling Tool is created with multiple shortcuts that can be used, participants also agreed that the tool helps users to achieve tasks in an acceptable number of steps (52% agreed, 40% strongly agreed). However, two participants left a comment that adding constraints and parameters were the most time-consuming operations. One of them suggested that constraints and parameters should be added on double-click and that naming patterns could be used to fill in all the attributes needed. We fully agree with this statement, and we implemented the suggested features in the tool after the evaluation process was finished.

Nearly half of the participants had no opinion (48%) on whether MultiProLan is appropriate for their needs, which is expected as most participants did not have much experience in production process modeling, discussed later in this section.

Participants also had a positive opinion on the Eclipse environment (52% agreed, 28% strongly agreed) and the attractiveness of graphical symbols (44% agreed, 48% strongly agreed). In addition,

participants strongly agreed that separating MasL models from production system details helps users to model production processes more easily (32% agreed, 64% strongly agreed) and that by creating different modeling layers, models have become more readable (12% agreed, 88% strongly agreed). Such participants' approval is important as different detail levels and modeling layers are some of the most significant features of MultiProLan and Process Modeling Tool. In the free comment section, a few participants stated that utilizing different modeling layers is very useful in production process modeling. Two of them suggested that additional modeling layers should be created for products and capabilities, enabling even more flexibility on what users could show on diagrams. In the past, we discussed this and decided to create a +/- button on a process step to show/hide capabilities and products. However, with additional modeling layers, this may be done more efficiently, and we plan to implement and test additional layers in the future.

**Reliability.** For the reliability characteristic, participants mostly agreed that MultiProLan protects users against making errors (60% agreed, 24% strongly agreed). They also mostly agreed that MultiProLan has a functional model validator (32% agreed, 36% strongly agreed); however, there were participants who did not fully agree with that statement (4% disagreed, 28% had no opinion). One participant wrote a comment that error messages were not clear enough to solve problems quickly in a model. In the future, we plan to improve the model validator.

**Expressiveness.** Participants also agreed or strongly agreed about the first two questions of the expressiveness characteristic – that a problem-solving strategy can be mapped into a specification easily (64% agreed, 36% strongly agreed) and that MultiProLan is at the right abstraction level (52% agreed, 48% strongly agreed). However, the third question about MultiProLan providing one and only one good way to express every concept of interest had mixed answers (12% disagreed, 20% had no opinion), but still mostly positive (48% agreed, 20% strongly agreed). Such a response from participants can be partly explained by the difficulty of claiming that MultiProLan provides only one good way to model production processes for participants with little experience in the domain.

**Productivity.** More than half of the participants (52%) stated that the specification time of a production process model is relatively short by using MultiProLan. Such a statement can be a consequence of having nearly all the participants experienced with different CASE tools and many participants experienced with the BPMN language, which has some similar modeling concepts as MultiProLan. Also, Process Modeling Tool has different shortcuts to shorten the time needed for the production process modeling.

For each quality characteristic, we summarized the percentages of each question having positive feedback from participants and calculated the mean values of positive feedback for quality characteristics. The evaluation participants mostly agreed that MultiProLan has all the characteristics tested (the calculated mean values of positive feedback per quality characteristics are given in parenthesis): functional suitability (98.00%), usability (88.50%), reliability (76.00%), expressiveness (89.33%), and productivity (68.00%).

## 9.4.2 Statistical Analysis of the Questionnaire Answers

By using the SciPy library [255] for the Python programming language, we calculated the Spearman's rank correlation coefficient [256] between all the questions in the questionnaire. We did not consider correlations with the low coefficient and the p-value higher than 0.05. We also did not consider trivial correlations with the high coefficient within the same group of questions. For example, there were high correlations between participants' experience with business or production process modeling and their experience with CASE tools.

As we found many correlations with the p-value lower or equal to 0.05, we present only some of the strong correlations with the p-value lower or equal to 0.01. We may state that these selected correlations are significant, and they are presented in Table 9.2. If we look at the first three rows of Table 9.2, we may state that participants who found MultiProLan is at the right abstraction level,

who think that the modeling became easier by separating MasL models from production system details, and who found MultiProLan symbols good looking, they also think that MultiProLan is suitable for production process modeling. The last row in Table 9.2 shows that participants with higher experience in the production process modeling think MultiProLan is appropriate for their needs. More than 75% of the participants are (relatively) inexperienced in production process modeling. Therefore, it is not surprising that almost 50% of participants answered with "no opinion" when asked whether MultiProLan is appropriate for their needs.

**Table 9.2.** Correlation coefficients and p-values for related questions.

Question 1	Question 2	Correlation	P-value
Functional suitability [How would you describe MultiProLan's level of suitability for the production process specification?]	Expressiveness [MultiProLan is at the right abstraction level such that it is not more complex or detailed than necessary.]	0.601	0.001
Functional suitability [How would you describe MultiProLan's level of suitability for the production process specification?]	Usability [By separating MasL and DetL models, users can model production processes easier.]	0.582	0.002
Functional suitability [How would you describe MultiProLan's level of suitability for the production process specification?]	Usability [MultiProLan has graphical symbols that are good looking/attractive.]	0.539	0.005
Experience [How would you describe your previous experience in designing production processes?]	Usability [MultiProLan is appropriate for your needs.]	0.504	0.010

Based on the analysis made in this section, the evaluation results presented in Section 9.4.1, and as there is more than 50% of positive feedback from the evaluation participants per each quality characteristic, **we may confirm the  $EH_{null}$  hypothesis** found in Section 9.1. Thus, we may state that MultiProLan can be used in practice as it has all the following quality characteristics: functional suitability, usability, reliability, expressiveness, and productivity. However, based on the participants' comments discussed throughout Section 9, MultiProLan can be further improved. The free comment section proved to be valuable to us during the evaluation process, and we are fully aware of all the comments that participants have written. Therefore, we applied most suggestions made by participants and left some for future development.

## 9.5 Overview of Other Quality Characteristics

There are quality characteristics that require knowledge of MultiProLan's implementation details to evaluate them. These quality characteristics are maintainability, extensibility, reusability, and integrability. As the experiment's participants were unfamiliar with MultiProLan's implementation details, the authors of MultiProLan evaluated these quality characteristics.

MultiProLan was implemented in EMF and Eclipse Sirius frameworks. These frameworks provide many useful features that help create the tool prototype fast. However, the tool depends on these frameworks and thus not all its features can be adapted.

**Maintainability** represents the degree to which a language is easy to maintain, and it comprises two characteristics:

- modifiability – new functionalities can be added to the language without degrading existing functionalities; and
- low coupling – a change to one component has minimal impact on other components.

Adding modeling concepts to the MultiProLan's meta-model does not require manual changes to the existing modeling concepts, and these novel modeling concepts will be automatically added

to Process Modeling Tool. Thus, existing functionalities of the language will not be degraded. However, changing the existing modeling concepts often requires different manual changes, as the meta-model, graphical syntax, and modeling tool are tightly coupled.

**Extensibility** represents the degree to which a language has a mechanism for users to add new features. Users cannot add new features to Process Modeling Tool. Even if that is possible, users would need to know the implementation details of the whole system as process models are used for the execution purpose, and thus users would need to extend the execution mechanism to support newly added features.

**Reusability** represents the degree to which language constructs can be used in more than one language. MultiProLan's modeling concepts have been reused in creating languages for production system modeling [5], human resource modeling [16,17], and collaborative production process modeling [19]. For example, as mentioned in Section 7.1, modeling concepts such as capabilities, resources, storage, and material flows have been reused for the production system modeling language.

**Integrability** represents the degree to which a language is amenable to integration with other languages. MultiProLan is being integrated with languages for production system, human resource, and collaborative production process modeling through the modeling concepts that are shared between them. The languages for production system and human resource modeling are used to specify all the necessary details of a production system, and these details are used during the creation of DetL models. MultiProLan production process models are used as part of collaborative production process models as well, allowing production process models to be shared between different collaborative parties.

## 9.6 Threats to Validity

During the preparation and performance of the experiment, we aimed to minimize threats to the experiment's validity. However, there are some possible internal, external, and conclusion threats to validity that need to be discussed.

**Internal threats to validity.** Participants of the experiment had different experiences in process modeling. If all the participants were highly experienced or highly inexperienced, the questionnaire results might be different. For example, most participants stated that the specification time of production process models with MultiProLan is short (16%) or relatively short (52%). It is possible that participants would evaluate the specification time as relatively long if they were more inexperienced with CASE tools and the BPMN language. The participants also belong to various groups, and it is possible that if some groups had more participants, the results might be different. As for the implementation-oriented quality characteristics discussed in the previous section (maintainability, extensibility, reusability, and integrability), we evaluated them as we are the only ones having insights into how the tool has been developed. Although we tried to stay as objective as possible, this represents one of the potential threats to the validity of the evaluation process. We have done the evaluation of these quality characteristics as it was hard to find a participant experienced both in production process modeling and the Sirius framework.

**External threats to validity.** We had to do the experiment online due to COVID-19, and it is possible that the experiment results may be different if we had been able to do the experiment on-site. Also, participants with different experiences in process modeling had done the same experiment tasks, which were not tailored for each participant based on their previous experience. Thus, participants with different experiences may get a different impression of MultiProLan. The tasks were made to cover most of the MultiProLan's modeling concepts, and some parts of the tasks were simplified so they could be modeled in a reasonable amount of time. The task with DetL modeling was simplified the most, as it contained only a part of the whole process since modeling the whole DetL process model would require a lot of time. If the tasks were of different difficulty, the experiment results might be different. Finally, as participants were allowed to ask any question

related to MultiProLan and the tasks, it is possible that some participants got more information about the language and the tasks than others.

**Conclusion threats to validity.** There were 25 participants who evaluated MultiProLan and Process Modeling Tool. It was hard to gather participants with at least basic experience in process modeling and CASE tools. If there were more participants, the experiment results might be different. Also, there were only 2 process engineers that evaluated the language and the tool. It would be better if more of them were participating, and we plan to extend the evaluation with more process and quality engineers in the future.

## 9.7 Summary

In this section, we presented the evaluation process and results of testing whether MultiProLan and Process Modeling Tool can be used in practice. Based on the overall positive feedback gathered from the evaluation participants, we may state that MultiProLan and Process Modeling Tool can be used in practice as they have the following quality characteristics: functional suitability, usability, reliability, expressiveness, and productivity. The participants mostly praised the MultiProLan domain coverage, the graphical representation of MultiProLan modeling concepts, the easiness of using MultiProLan through Process Modeling Tool, and the usage of different levels of detail and modeling layers. They also left various suggestions in the questionnaire's free comment section on how to improve MultiProLan and Process Modeling Tool, which helped us to make the language and the tool better. Some of these suggestions are already implemented and presented as part of this thesis.

Based on our experience from various use cases in the assembly industry and the evaluation results presented in this section, by using MultiProLan and Process Modeling Tool, process designers can specify production processes faster, more easily and efficiently, and with fewer faults, in comparison to the ways they are specifying production processes now. Many process designers still specify production processes by using textual documents, flowcharts, or spreadsheets. Without MultiProLan and Process Modeling Tool, process designers would be slowed down, and more faults may occur when specifying production processes, especially if designers are using some informal techniques. To prove the MultiProLan and Process Modeling Tool advantages over contemporary ways of specifying production processes, we plan to conduct a new evaluation process in which various participants will test the performance of MultiProLan and Process Modeling Tool and compare them to the production process specification techniques participants are mostly using.



## 10 Conclusions and Future Work

In this thesis, we presented a novel MD solution and a DSML for production process modeling and automatic generation of executable resource instructions and manufacturing documentation. The presented MD solution and a DSML named MultiProLan are developed to cope with the challenges (Ch1–Ch5) introduced by Industry 4.0.

This section is structured as follows. In Section 10.1, we discuss the outcome of testing the hypotheses defined in this thesis. The main contributions of the research presented in this thesis are outlined in Section 10.2, while future research is discussed in Section 10.3.

### 10.1 Outcome of Hypotheses Testing

Enabling flexible production, caused by various customer needs, is one of the main challenges that Industry 4.0 introduces. Customers require individualized and customized products, creating a disorder in the rigid mass production of a factory that is to produce such products. Accordingly, contemporary factories need to move from mass production to mass customization, by making production more flexible and endurable to product changes. A prerequisite for production flexibility is to have a smart factory with flexible production lines and dynamic resources that can move through the factory and execute high-level instructions. One way to manage such flexible production is to have production process models that are machine-readable and thus can be used to lead the execution of production processes, by executing models or generating executable resource instructions from them. Furthermore, handling errors during production is highly important in such a dynamic environment to run production smoothly and prevent losses. Accordingly, exact procedures to handle errors need to be defined.

Besides the execution of production processes, their models and manufacturing documentation need to be managed and stored. Individualized products may belong to the same product family already produced in the factory but vary in a few parts, colors, or dimensions. Therefore, factories of the future must maintain product families with numerous product and process variations and rapidly create new variations whenever a customer requires specific products. By having many processes and their variations, manual process specification becomes a difficult task for process designers that may lead to numerous specification errors. Another burdensome task for process designers is maintaining manufacturing documentation of different types in such a flexible environment with many product and process variations. Whenever there is a change in a production process, documents of different types must be updated and synchronized. These are time-consuming manual tasks, and they need to be done automatically to reduce the time required by process designers and to mitigate specification errors done by process designers due to a lot of manual work.

To make production more flexible and help factories enter the digital transformation era, we initiated the research presented in this thesis. To test the main hypothesis  $H_0$  of our research,

formulated in Section 3.2, we created a novel MD solution that comprises a novel methodological approach and a system, used for production process modeling and automatic transformation of process models into executable resource instructions and manufacturing documentation. Such a solution enables a formal specification of both resource-agnostic and resource-aware production process models that are machine-readable and thus transformable into executable resource instructions and manufacturing documentation. To confirm or reject the main hypothesis  $H_0$ , we discuss the derived hypotheses first.

Within our novel MD solution, we developed a novel DSML – MultiProLan, used for production process modeling. Such a language is created in a formal and systematic way, fulfilling the requirements presented in Section 4.2.1. The requirements need to be fulfilled by a process modeling language for its models to be suitable for dynamic production orchestration and automatic execution. Thus, MultiProLan is used to model production processes with details required for the automatic generation of executable resource instructions. Instruction Generator and protocol transformation components are implemented to automatically transform production process models into human-readable or machine-specific instructions, sent to resources on the shop floor for execution. Thus, **we may confirm the hypothesis  $H_1$** .

However, such instruction-generation-ready production process models are dependent on a specific production system, and it is difficult for process designers to specify such models, as they need to know all the technological and production system details. Production process models created by process designers need to be reusable in many production systems and need to be easy to read and understand. We developed MultiProLan in a way to include necessary modeling concepts used by process designers to specify resource-agnostic production process models, i.e., to be independent of any production system. Therefore, such process models can be reused in multiple production systems. Also, as they do not include production system details, such models are easy to read and understand. We may say that the readability and understandability of process models were also confirmed during the evaluation of MultiProLan and Process Modeling Tool, presented in Section 9. The evaluation participants provided us with positive feedback that MultiProLan elements are understandable (96%), learnable and rememberable (100%), good looking and attractive (92%), that process modeling becomes easier by separating resource-agnostic and resource-aware models (96%), and that models become more readable by creating different modeling layers (100%). In addition, we managed to represent different aspects of production process modeling in a uniform way, by having a single production process model that includes an execution flow, error handling, and different information related to the manufacturing documentation of different types. Accordingly, by having all the information stored in a single model, different users, such as process and quality engineers, can work together on the same process model, specifying details related to their field of expertise. However, as such a process model may become overwhelmed with information and details, we have implemented different modeling layers in Process Modeling Tool to show modeling concepts related to the specific user group and hide concepts not related to the user group. Therefore, **we may confirm the hypothesis  $H_2$** .

As resource-agnostic production process models are independent of any production system, they can be neither executed nor executable resource instructions can be generated from them. However, for process models to lead production process execution, they need to include technological and production system details. Such resource-aware production process models are overloaded with details about resources, storage, production logistics, and machine configuration activities, burdening process designers additionally. Thus, it is difficult to specify resource-aware production process models manually, becoming a burdensome task for process designers. As MultiProLan is a capability-based process modeling language, providing each process step in a model with a capability required for a process step to be executed, it is also possible to automatically transform resource-agnostic process models into resource-aware process models by using an intelligent system such as Orchestrator. Regardless of whether resource-aware production process models are created manually or automatically, MultiProLan contains modeling concepts that allow such an enrichment of resource-agnostic process models. Process designers can choose a production system in which a production process model is to be executed and automatically create a

production-system-specific process model with the help of Orchestrator. They can also manually create such models or change or optimize existing resource-aware production process models created by Orchestrator. Thus, **we may also confirm the hypothesis H<sub>3</sub>**.

Production process models need to be usable for the automatic transformation into executable resource instructions and manufacturing documentation. As MultiProLan's models are machine-readable, resource-aware production process models can be automatically transformed into executable resource instructions, which may be human-readable or machine-specific instructions. Also, depending on the information needed, both resource-agnostic and resource-aware production process models can be automatically transformed into manufacturing documentation of different types. By using Instruction Generator, Documentation Generators, and a set of transformation rules, these kinds of M2T transformations can be performed. The automatic generation of manufacturing documentation helps process designers avoid keeping the manufacturing documentation up to date manually, which is useful especially in the era of Industry 4.0, as many product and process variations exist. Accordingly, **we may confirm the hypothesis H<sub>4</sub>**.

**By confirming the derived hypotheses H<sub>1</sub>, H<sub>2</sub>, H<sub>3</sub>, and H<sub>4</sub>, we may state that the main hypothesis H<sub>0</sub> is also confirmed and that the main goal of this research is met.**

Furthermore, we may state that the results of our research have theoretical, development, application, and socio-economic contributions, discussed in the following section.

## 10.2 Research Contributions

In this section, we discuss the achieved contributions of the research presented in this thesis. We divided the contributions into the following four groups: theoretical, development, application, and socio-economic contributions. Each contribution group is outlined in the following subsections.

### 10.2.1 Theoretical Contributions

Before the creation of MultiProLan, we investigated the state-of-the-art related to the application of the MD paradigm and DSLs in ISs and Industry 4.0, and especially production process modeling, as presented in Section 4. Based on the preliminary research, we formulated requirements for a production process modeling language whose models would be suitable for dynamic production orchestration and automatic execution of production processes. Then, we conducted a survey of existing modeling languages and for each one of them we checked the fulfillment of the requirements. We also made an analysis of what is covered by the languages and what is missing, making a basis for future research.

After the investigation of the state-of-the-art, we analyzed the production process modeling domain and identified the main concepts that a language for production process modeling needs in the Industry 4.0 context. The domain analysis and concepts are presented in Section 6, by means of the FODA model, creating a foundation for the creation of a novel DSML for production process modeling.

During our research, we did not encounter such a formulation of production process modeling requirements in the context of Industry 4.0, an extensive survey and an analysis of existing production process modeling languages, and an analysis of the production process modeling domain. Therefore, these are theoretical contributions that may be used as a basis for future research endeavors.

The result of our research is a novel MD solution that comprises both the approach and the system for dynamic production orchestration and automatic generation of executable resource instructions and manufacturing documentation, based on models created by a novel DSML. Such an MD solution is presented in Section 5, while a novel, capability-based modeling language –

MultiProLan, is presented in Section 7. By using Orchestrator and MultiProLan's models created at two detail levels, we created a novel methodology to automatically transform resource-agnostic into resource-aware production process models. Thus, a production process model is independent of any production system, making production process modeling easier for process designers. Such a production process model can be automatically transformed into process models specific to different production systems in which they are to be used for automatic execution.

Finally, a theoretical contribution is achieved through applying MD principles in the production domain, pointing out that it is possible to use the MD paradigm and contribute to production, making it more flexible.

### 10.2.2 Development Contributions

The development and implementation of the novel MD solution, language and its modeling tool for production process modeling, and code generators for automatic generation of executable resource instructions and manufacturing documentation are the main development contributions. We summarize their contributions in this section.

The MD solution is developed using the Java programming language and it helps process designers create production process models independent of any specific production system, making resource-agnostic process models. With the help of Orchestrator, resource-agnostic process models can be automatically transformed into resource-aware process models for the specific production system, which are used by Instruction Generator to automatically generate resource instructions to execute process operations. By utilizing such an MD solution, process designers only model production processes without production system details, such as resources that are to execute process steps, production logistics, and machine configurations, making process modeling easier.

MultiProLan is developed using EMF and Sirius frameworks to support modeling concepts related to the Industry 4.0 context, such as product and process variations, resource collaboration, and error handling, among others. The language is built to fulfill the requirements stated in Section 4.2.1. It is a capability-based modeling language, so its models can be independent of any specific production system. The goal of creating MultiProLan is to support the modeling of all production details required for the automatic execution but not to be too complex for a human to comprehend. To achieve this goal, two levels of detail are implemented so that production processes can be modeled in a generic way. By creating two levels of detail, production process models become independent of the production system details, and thus efforts needed during the production process modeling are reduced. MultiProLan is used to:

- speed up and increase the precision with which production processes are designed;
- make fewer faults during process design; and
- enable faster changes in production process models.

Such a language is implemented formally, increasing consistency during modeling, and decreasing the amount of time needed for the modeling.

Process Modeling Tool is implemented by using the Sirius framework to allow process designers to model production processes by utilizing MultiProLan. The tool supports various mechanisms, such as modeling layers and filtering tools, to deal with the scalability problem of process diagrams. By using modeling layers, users of different profiles can work together on the same process model and hide modeling concepts that are not relevant to them. Filtering tools can be used to show only product and process variations that a process designer currently works with and hide others to keep a process diagram clear. Therefore, by using different mechanisms of Process Modeling Tool, a production process model can store various information, but also maintain the process diagram's readability. Furthermore, as the tool is integrated with the MD solution, it is possible to start production from the tool and monitor the execution of the invoked process. Therefore, bottlenecks and production process modeling errors can be detected and mitigated.

Instruction Generator is developed from scratch using the Java programming language to automatically transform resource-aware production process models into high-level instructions, that are transformed into human-readable or machine-specific instructions by protocol transformation components in Digital Twin. Such a generator, as a part of the MD solution, contributes to flexible production, as process models are used to lead production process execution by generating instructions from them. Also, as a part of the MD solution, Instruction Generator makes guided production possible, sending instructions to human workers, alongside textual descriptions, images, audio, and videos. Guided production helps novice workers create products during the training phase and reduces the time experienced workers spend when helping them out. In addition, guided production may help human workers when they frequently change a product or a product variation they produce, or when they create complex products that could take hours to produce.

Documentation Generators are implemented by using the Xtend language to automatically generate, update, and maintain manufacturing documentation of different types from the production process models. Thus, process designers' efforts to manually perform such tasks are reduced.

### 10.2.3 Application Contributions

The MD solution and MultiProLan are applied in the production domain, particularly in the assembly industry. They are applied in a demonstration environment in which human workers, industrial mobile robots, and research-grade smart robots are used to assemble objects from LEGO® bricks, as presented in Section 8.2. We also modeled various production processes from the assembly industry to showcase the usage of MultiProLan, such as the one presented in Section 8.1.

MultiProLan and Process Modeling Tool were evaluated by different groups of users, including process engineers, software developers, researchers, and students. The evaluation results, as presented in Section 9, provided us with valuable feedback from users, and the evaluation hypothesis  $EH_{\text{null}}$  was confirmed – that MultiProLan can be used in practice as it has all the following quality characteristics: functional suitability, usability, reliability, expressiveness, and productivity.

We also presented a new practical experience from applying a novel methodological approach, a software tool, and a DSML in this thesis.

### 10.2.4 Socio-Economic Contributions

The MD solution and MultiProLan are put into public use as a general model of production process management. They are applicable in a wide range of organizations and could enable significant production process improvements. Also, they raise general accumulated knowledge on how to contribute to such process improvements.

By applying the newly created solution and modeling language, manufacturing companies can increase the level of flexibility and automation in their production and move to the digital transformation era, thus fully applying the Industry 4.0 philosophy. Therefore, the research results presented in this thesis, which originates from the collaboration between organizations in Serbia and Austria, can improve production in the context of Industry 4.0, which is one of the key economic factors of the European Union.

## 10.3 Future Work

In this section, we present the future work related to the research presented in this thesis. We divide the future work into three parts described in the following subsections:

- future research in the domain of production process modeling;

- further development of MultiProLan and Process Modeling Tool; and
- new application domains of MultiProLan.

Parts of the described future work have been already discussed in [11] and [14].

### 10.3.1 Future Research in the Domain of Production Process Modeling

The domain of production process modeling covers a wide range of different aspects that need to be considered. In this thesis, we cover the execution and error handling aspects of production process modeling. However, there are various aspects and extensions that need to be covered in the future.

In our MD solution, Orchestrator is used to automatically transform resource-agnostic production process models into resource-aware production process models. Such a transformation can be achieved by using different matching and scheduling algorithms, based on the knowledge about a production system stored in Knowledge Base. This knowledge is created by using Resource Modeling Tool – a prototype modeling tool that uses a custom-built DSML to specify production systems. The DSML is in the early implementation stage, covering only basic modeling concepts related to production systems that are required in our solution (i.e., production resources with capabilities and constraints, as well as interactions between the resources). Model examples created by such a language are already presented in [5]. However, this language and its modeling tool need to be further developed and extended with modeling concepts, such as different types of resources, their interfaces, capabilities, constraints, and storage. Therefore, the extended production system models can be used by Orchestrator for better allocation of resources in a factory.

In addition, as human characteristics are important in the context of Industry 4.0, it is necessary to specify human roles, capabilities, and competencies [257]. Human workers need to be modeled within production system models with more details, as there are different additional constraints when they perform some tasks. For example, unqualified workers cannot perform some specialized tasks; due to legal constraints workers cannot lift a heavy object more than a certain number of times; or color-blind persons cannot perform some tasks. All these human worker characteristics and constraints need to be carefully modeled, and Orchestrator or a resource manager can use such knowledge to better allocate resources. Therefore, human skills such as technical, knowledge, personal, cognitive, and social are considered in Industry 4.0, as well as their health, physical, and mental capabilities [257]. The foundation for human worker modeling in the context of Industry 4.0, with the aim to achieve better integration of human workers and machines, is already laid out in [15]. The language prototype of production and organizational perspectives of the human worker modeling is already presented in [16] and [17], respectively. Human worker models will be extended with concepts such as roles, skills, competencies, capabilities, and limitations, providing more information to Orchestrator when matching and scheduling resources in a factory.

Safety, human physical risk factors, and energy consumption aspects are particularly considered in Industry 4.0, but rarely modeled. These are the modeling concepts that MultiProLan still lacks and need to be considered in the future. The lack of safety aspects modeling with MultiProLan is also noted by one of the evaluation participants. The risk factors need to be modeled for each process step and the decision of which smart resources are to be assigned to process steps should also depend on these factors, which could be decided by Orchestrator. If a risk is high for a human worker, a robot needs to be assigned to execute a process step. Energy consumption also needs to be modeled to estimate the costs of different resources for each process step or to estimate the costs of the whole process executed in different production systems. Therefore, Orchestrator could propose or choose the production system in which the production costs will be the lowest. Both HSE and energy consumption modeling concepts need to be presented as new modeling layers to lower the complexity of process models.

MultiProLan is designed to model processes to be executed in a single facility and it is not possible to model production processes that are shared among multiple facilities or smart factories.

A collaboration between participants is needed to enable execution in multiple factories, creating various intermediate products and the final product in different facilities and factories. This collaboration requires a specification of production processes at the BSM level and an implementation of horizontal integration between the participants. As a product is created in different factories, the quality of each part must be guaranteed as well as contract fulfillment. Therefore, Distributed Ledger Technology (DLT) systems and smart contracts can be used to guarantee the quality of products created by various collaborative parties, to guarantee contract fulfillment in a trustworthy way, and to supervise the state of production [18]. To model collaborative production processes, CE-MultiProLan was created [19] based on MultiProLan and will be further improved to support the modeling of trustworthy collaborative production processes, the automatic generation of smart contracts and DLT configuration artifacts, and the execution of such production process models. Smart contracts can be generated from CE-MultiProLan models and stored in DLT networks, thus distributing production data between collaborative parties to monitor production in near real-time. The automatic generation of DLT artifacts was already presented in [258]. The whole MD system that supports CE-MultiProLan and the automatic generation of smart contracts will be developed and discussed in the future.

Another research field can cover the automatic extraction of production process models from product specifications, such as CAD models. There have been a lot of research activities in this field in past years [259–261], but many research questions are still open. It would be beneficial if MultiProLan process models could be automatically extracted from product specifications. Therefore, product designers would need to create a product model, after which process designers would need to check the extracted process model and optimize it manually whenever necessary, by using MultiProLan and Process Modeling Tool.

### 10.3.2 Further Development of MultiProLan and Process Modeling Tool

In this section, we discuss further development of MultiProLan and Process Modeling Tool. These development improvements are partially gathered as feedback from the evaluation process presented in Section 9.

Our MD solution allows a production system to be chosen for which a resource-aware process model is to be created based on a resource-agnostic process model. Currently, there is no recommendation system that would help process designers choose a production system. The recommendation should be based on various criteria set by process designers, such as duration of process execution or energy consumption. The estimation of process step execution duration is already stored in MultiProLan production process models. Such an estimation, along with the energy consumption estimation, discussed as a future work in Section 10.3.1, can be used to estimate the time and costs of a whole process. As a single process can be executed in various production systems, the recommendation system needs to recommend a production system in which the process execution will be done with the best performance based on the chosen criteria.

As capabilities and parameters are not yet standardized, we created a repository with a taxonomy of capabilities and parameters used in our MD solution. Once capabilities and parameters are standardized, we will replace our capability taxonomy with the standardized one. Therefore, our system will be compatible with any smart resource that supports the standard, extending possibilities and application domains of our solution.

Production quality is not discussed in detail in this thesis, as it is out of the thesis's scope but is quite important. Our MD solution supports the production quality only through completion and acceptance criteria specified for each process step in MultiProLan models. The production quality needs to be further discussed, researched, and integrated into MultiProLan and the MD solution, thus enabling various quality aspects to be specified and consequently increasing the final product quality.

A collaboration between resources, especially a human-machine collaboration, is highly important in Industry 4.0, as many resources need to collaborate to create different products and their variations. Our MD solution utilizes collaboration through message exchange between resources, synchronizing their activities. However, MultiProLan currently allows only the basic modeling of messages between process steps at a high level, such as messages to start or finish performed activities. As part of future work, more complex communication between resources needs to be available for modeling using MultiProLan and supported by our MD solution. Additionally, as mentioned in Section 10.3.1, safety aspects should also be modeled, especially in cases of human-machine collaboration. This was also stated by one of the process engineers during the MultiProLan evaluation process.

MultiProLan has graphical syntax created as process designers are mostly used to process charts. However, it is possible that some process designers could find a textual concrete syntax more suitable. We plan to create the MultiProLan textual syntax as well, and evaluate, analyze, and compare the usage of both graphical and textual syntaxes with process designers. The creation of MultiProLan textual syntax was also proposed by one of the MultiProLan evaluation participants.

During the evaluation process of MultiProLan, we managed to have only two process engineers join the experiment. Their feedback was of utmost importance, and we plan to extend the evaluation in the future by gathering more participants, especially process engineers. We also plan to continue with the evaluation with process engineers from different industry domains. However, an extension of the evaluation will require additional time to gather process engineers from different domains. The feedback from process engineers may be useful in the context of domain coverage and language usage. Additionally, we plan to conduct a new evaluation process in which various participants will test the performance of MultiProLan and Process Modeling Tool and compare them to the production process specification techniques they are mostly using.

There were also a few minor comments related to MultiProLan and Process Modeling Tool left by participants during the evaluation process, such as:

- it was confusing where to put an output product of a collaboration between process steps;
- error messages provided by the model validator were not clear enough to solve problems quickly in a model; and
- an additional modeling layer may be created for capabilities and products so they can be shown or hidden at once.

We will consider all the comments gathered from the evaluation process and will improve both MultiProLan and Process Modeling Tool in the future.

### 10.3.3 New Application Domains of MultiProLan

MultiProLan aims to be used to model production processes of any kind. Currently, it is used to model discrete product manufacturing processes, especially the assembly of goods. There are various new application domains for MultiProLan to be used, and in this section, we discuss these additional MultiProLan applications.

For MultiProLan to be used in different manufacturing domains, such as process manufacturing, additional features, like timers or time estimations, are needed. While discussing with domain experts about the applicability of MultiProLan and Process Modeling Tool outside of the discrete production of hardware elements domain, their feedback was that most of the MultiProLan concepts could be applied to process manufacturing as well. Therefore, one of the most important future research and development steps is to extend the language to support the modeling of process production, whether it is performed in a fully continuous (e.g., water plants) or a batch (e.g., breweries, sugar factories, and pharma factories) manner. We also plan further to evaluate the language in additional industrial use cases to improve the domain concept coverage and the stability of developed tools.



We plan to integrate our solution with a detection system used to detect the execution of process steps in the manufacturing assembly industry, such as the one that utilizes depth cameras [262]. Currently, human workers must confirm via their tablet, smart watch, or monitor whether a process step is executed successfully, or an error occurred. By using a detection system, a process step execution status can be automatically detected and sent as feedback to our system. Therefore, our system could send instructions one by one automatically, without waiting for a human worker to confirm the execution status. The detection system also aims to prevent workers from missing some process steps and thus save the time and costs required to fix the issue. If some process steps are not performed, it could take hours to reassemble the product as not all steps are easily reversible, or some parts of a product cannot be easily reached, and in some cases, the whole product needs to be reassembled [262]. Using our MD solution with the detection system would improve guided production through the automatic sending of process step execution status, helping workers assemble products more efficiently. The detection system could send feedback to Process Modeling Tool as well, updating a process model while monitoring process execution.

In addition, devices supporting Augmented Reality (AR) could be used by human workers when assembling products, and our MD solution could send them instructions on how to execute process steps. The AR technology could be a part of improved guided production and help human workers assemble products. Both AR and Mixed Reality (MR) have been applied by researchers in recent years for process modeling and execution, in the context of IoT and Industry 4.0, as well as for guided production [263–265]. The integration of our MD solution and MultiProLan with AR devices would require further research and development. MultiProLan models could be used for the automatic generation of human-readable instructions that are enriched with details related to AR or MR that would be sent to appropriate devices human workers wear.

During process execution, gathered feedback from resources and sensors is stored in storage. Each process step execution status or error occurrence is stored in the storage and can be used for various analyses. Stored data can be used for process analysis, leading to the detection of process anomalies, bottlenecks, and failures. Also, the data can be used for predictive analysis, preventing shutdowns or material shortages. For example, based on the process execution data, including the execution time required by various resources to perform process steps, material shortages in storage can be predicted and materials can be refilled in time. Such a prediction can save the time that a resource in charge of replacing materials in storage must invest when checking out the materials' quantity status. Based on different findings, production processes can be optimized and unscheduled system shutdowns can be prevented as failures can be mitigated. The gathered data can be used to start new research in the fields of data science and process mining, which could provide useful knowledge about how processes can be improved, and thus lower the costs of producing a product or making a product of better quality. For example, when unordered process steps are specified, various resources may perform these steps in different order. The analysis of process execution data may provide information on which process steps order is the fastest to perform. By performing the fastest way of executing unordered process steps, more products can be produced in the same amount of time or a product price can be lower compared to other ways of executing the same process steps.

We believe that our MD solution can be further improved and developed to support flexible production in the industry even more. Therefore, its full potential is yet to be discovered, alongside new application domains, supporting companies in the digital transformation and Industry 4.0 era.



## References

- [1] M.P. Groover, *Fundamentals of Modern Manufacturing: Materials, Processes, and Systems*, 4th ed., John Wiley & Sons, Inc., Hoboken, NJ, 2010.
- [2] J.T. Black, R.A. Kohser, *DeGarmo's Materials and Processes in Manufacturing*, 13th ed., John Wiley & Sons, Inc., Hoboken, NJ, 2019.
- [3] N. Keddis, *Capability-Based System-Aware Planning and Scheduling of Workflows for Adaptable Manufacturing Systems*, Ph.D. Thesis, Technical University of Munich, 2016.
- [4] M. Pisarić, V. Dimitrieski, M. Vještica, G. Krajoski, Towards a Non-Disruptive System for Dynamic Orchestration of the Shop Floor, in: *IFIP Adv. Inf. Commun. Technol. AICT*, Springer Nature, Novi Sad, Serbia, 2020: pp. 469–476.  
[https://doi.org/10.1007/978-3-030-57997-5\\_54](https://doi.org/10.1007/978-3-030-57997-5_54).
- [5] M. Pisarić, V. Dimitrieski, M. Vještica, G. Krajoski, M. Kapetina, Towards a Flexible Smart Factory with a Dynamic Resource Orchestration, *Appl. Sci.* 11 (2021) 7956:1–7956:25.  
<https://doi.org/10.3390/app11177956>.
- [6] K. Dorofeev, Skill-Based Engineering in Industrial Automation Domain: Skills Modeling and Orchestration, in: *Proc. ACMIEEE 42nd Int. Conf. Softw. Eng. Companion Proc.*, Association for Computing Machinery, Seoul, South Korea, 2020: pp. 158–161.  
<https://doi.org/10.1145/3377812.3381394>.
- [7] M. Vathoopan, K. Dorofeev, A. Zoitl, Skill-Based Engineering of Automation Systems: Use Case and Evaluation, in: R. Drath (Ed.), *Autom. Ind. Cookb.*, De Gruyter Oldenbourg, Berlin, Boston, 2021: pp. 555–578. <https://doi.org/10.1515/9783110745979-033>.
- [8] M. Vještica, V. Dimitrieski, M. Pisarić, S. Kordić, S. Ristić, I. Luković, Towards a formal description and automatic execution of production processes, in: *Proc. 2019 IEEE 15th Int. Sci. Conf. Inform.*, IEEE, Poprad, Slovakia, 2019: pp. 463–468.  
<https://doi.org/10.1109/Informatics47936.2019.9119314>.
- [9] M. Vještica, V. Dimitrieski, M. Pisarić, S. Kordić, S. Ristić, I. Luković, Towards a Formal Specification of Production Processes Suitable for Automatic Execution, *Open Comput. Sci.* 11 (2021) 161–179. <https://doi.org/10.1515/comp-2020-0200>.
- [10] M. Vještica, V. Dimitrieski, M. Pisarić, S. Kordić, S. Ristić, I. Luković, The Syntax of a Multi-Level Production Process Modeling Language, in: *Proc. 2020 Fed. Conf. Comput. Sci. Inf. Syst. FedCSIS 2020*, Polish Information Processing Society, Sofia, Bulgaria, 2020: pp. 751–760. <https://doi.org/10.15439/2020F176>.
- [11] M. Vještica, V. Dimitrieski, M. Pisarić, S. Kordić, S. Ristić, I. Luković, Multi-level production process modeling language, *J. Comput. Lang.* 66 (2021) 101053:1–101053:26.  
<https://doi.org/10.1016/j.cola.2021.101053>.
- [12] M. Todorović, Đ. Ivanišević, M. Vještica, V. Dimitrieski, I. Luković, An Automatic Generation of Production Documentation from MultiProLan Models, in: *Proc. 11th Int. Conf. Inf. Soc. Technol. ICIST 2021*, Information Society of Serbia – ISOS, Kopaonik, Serbia, 2021: pp. 96–101.

- [13] M. Vještica, V. Dimitrieski, M. Pisarić, S. Kordić, S. Ristić, I. Luković, An Application of a DSML in Industry 4.0 Production Processes, in: *IFIP Adv. Inf. Commun. Technol. AICT*, Springer Nature, Novi Sad, Serbia, 2020: pp. 441–448. [https://doi.org/10.1007/978-3-030-57993-7\\_50](https://doi.org/10.1007/978-3-030-57993-7_50).
- [14] M. Vještica, V. Dimitrieski, M.M. Pisarić, S. Kordić, S. Ristić, I. Luković, Production processes modelling within digital product manufacturing in the context of Industry 4.0, *Int. J. Prod. Res.* 61 (2023) 6271–6290. <https://doi.org/10.1080/00207543.2022.2125593>.
- [15] D. Antanasijević, S. Ristić, M. Vještica, V. Dimitrieski, M. Pisarić, Towards a Formal Specification of Human Worker for Industry 4.0, in: *Proc. 2022 IEEE 16th Int. Sci. Conf. Inform.*, IEEE, Poprad, Slovakia, 2022: pp. 33–38. <https://doi.org/10.1109/Informatics57926.2022.10083444>.
- [16] D. Antanasijević, S. Ristić, M. Vještica, D. Stefanović, V. Dimitrieski, M. Pisarić, A Prototype of a Domain-Specific Modeling Language for Formal Specification of a Human Worker, *Acta Electrotech. Inform.* 23 (2023) 33–40. <https://doi.org/10.2478/aei-2023-0010>.
- [17] D. Antanasijević, M. Vještica, L. Grubić-Nešić, V. Dimitrieski, M. Pisarić, S. Ristić, An Organizational Perspective of Human Resource Modeling, *IPSI Bgd Trans. Internet Res.* 19 (2023) 64–75. <https://doi.org/10.58245/ipsi.tir.2302.08>.
- [18] N. Todorović, M. Vještica, V. Dimitrieski, M. Zarić, N. Todorović, I. Luković, Towards Trustworthy Horizontal Integration in Industry 4.0 Based on DLT Networks, in: *Proc. 2020 Fed. Conf. Comput. Sci. Inf. Syst. FedCSIS 2020*, Polish Information Processing Society, Sofia, Bulgaria, 2020: pp. 63–69. <https://doi.org/10.15439/2020F210>.
- [19] N. Todorović, M. Vještica, N. Todorović, V. Dimitrieski, I. Luković, A Novel Approach and a Language for Facilitating Collaborative Production Processes in Virtual Organizations Based on DLT Networks, in: *Proc. 2nd Int. Conf. Innov. Intell. Ind. Prod. Logist. IN4PL 2021*, SciTePress, Science and Technology Publications, Lda, 2021: pp. 197–208. <https://doi.org/10.5220/0010720900003062>.
- [20] M.K. Sott, L.B. Furstenu, Y.P.R. Rodrigues, L.M. Kipper, G.L. Tortorella, J.R. López-Robles, M.J. Cobo, Exploring the Evolution Structure of Process Modelling for Industry 4.0: a Science Mapping for Proposing Research Paths, in: *Proc. 2nd South Am. Conf. Ind. Eng. Oper. Manag.*, IEOM Society International, Sao Paulo, Brazil, 2021: pp. 537–550.
- [21] L.D. Xu, E.L. Xu, L. Li, Industry 4.0: state of the art and future trends, *Int. J. Prod. Res.* 56 (2018) 2941–2962. <https://doi.org/10.1080/00207543.2018.1444806>.
- [22] S. Erol, A. Jäger, P. Hold, K. Ott, W. Sihn, Tangible Industry 4.0: A Scenario-Based Approach to Learning for the Future of Production, in: *Proc. 6th CIRP Conf. Learn. Factories*, 2016: pp. 13–18. <https://doi.org/10.1016/j.procir.2016.03.162>.
- [23] L.D. Xu, Enterprise Systems: State-of-the-Art and Future Trends, *IEEE Trans. Ind. Inform.* 7 (2011) 630–640. <https://doi.org/10.1109/TII.2011.2167156>.
- [24] J. Jiao, M.M. Tseng, Q. Ma, Y. Zou, Generic Bill-of-Materials-and-Operations for High-Variety Production Management, *Concurr. Eng.* 8 (2000) 297–321. <https://doi.org/10.1177/1063293X0000800404>.
- [25] American Society of Mechanical Engineers, ASME Standard: Operation and Flow Process Charts, ASME, New York, 1947.
- [26] Korean Standards Association (KSA), KS A 3002 Standard. <https://www.kssn.net/en/> (accessed April 5, 2020).
- [27] D.H. Stamatis, Failure Mode and Effect Analysis: FMEA from Theory to Execution, 2nd ed., American Society for Quality, Quality Press, Milwaukee, Wisconsin, United States, 2003.
- [28] Ford Motor Company, Failure Mode and Effects Analysis: FMEA Handbook (with Robustness Linkages), Version 4.2, Ford Motor Company, Dearborn, Michigan, United States, 2011.
- [29] D.H. Stamatis, Advanced Product Quality Planning: The Road to Success, 1st ed., CRC Press, Taylor & Francis Group, Boca Raton, Florida, United States, 2018.
- [30] Object Management Group, Business Process Model and Notation, Version 2.0.2, 2014.
- [31] S. Zor, K. Görlach, F. Leymann, Using BPMN for Modeling Manufacturing Processes, in: *Proc. 43rd CIRP Int. Conf. Manuf. Syst. ICMS 2010*, Vienna, Austria, 2010: pp. 515–522.

- [32] S. Zor, D. Schumm, F. Leymann, A Proposal of BPMN Extensions for the Manufacturing Domain, in: Proc. 44th CIRP Int. Conf. Manuf. Syst. ICMS 2011, Madison, Wisconsin, USA, 2011: pp. 1–7.
- [33] M. Polderdijk, I. Vanderfeesten, J. Erasmus, K. Traganos, T. Bosch, G. van Rhijn, D. Fahland, A Visualization of Human Physical Risks in Manufacturing Processes Using BPMN, in: Lect. Notes Bus. Inf. Process. LNBI, Springer, Barcelona, Spain, 2017: pp. 732–743. [https://doi.org/10.1007/978-3-319-74030-0\\_58](https://doi.org/10.1007/978-3-319-74030-0_58).
- [34] H. Ahn, T.-W. Chang, Measuring Similarity for Manufacturing Process Models, in: IFIP Adv. Inf. Commun. Technol. AICT, Springer Nature, Seoul, Korea, 2018: pp. 223–231. [https://doi.org/10.1007/978-3-319-99707-0\\_28](https://doi.org/10.1007/978-3-319-99707-0_28).
- [35] H. Ahn, T.-W. Chang, A Similarity-Based Hierarchical Clustering Method for Manufacturing Process Models, Sustainability. 11 (2019) 2560:1–2560:18. <https://doi.org/10.3390/su11092560>.
- [36] I. Abouzid, R. Saidi, Proposal of BPMN extensions for modelling manufacturing processes, in: Proc. 2019 5th Int. Conf. Optim. Appl. ICOA, IEEE, Kenitra, Morocco, 2019: pp. 1–6. <https://doi.org/10.1109/ICOA.2019.8727651>.
- [37] P. Michalik, J. Štofa, I. Zolotová, The Use of BPMN for Modelling the MES Level in Information and Control Systems, Qual. Innov. Prosper. 17 (2013) 39–47. <https://doi.org/10.12776/qip.v17i1.68>.
- [38] I. Graja, S. Kallel, N. Guermouche, A.H. Kacem, BPMN4CPS: A BPMN Extension for Modeling Cyber-Physical Systems, in: Proc. 2016 IEEE 25th Int. Conf. Enabling Technol. Infrastruct. Collab. Enterp. WETICE, IEEE, Paris, France, 2016: pp. 152–157. <https://doi.org/10.1109/WETICE.2016.41>.
- [39] P. Bocciarelli, A. D'Ambrogio, A. Giglio, E. Paglia, A BPMN extension for modeling Cyber-Physical-Production-Systems in the context of Industry 4.0, in: Proc. 2017 IEEE 14th Int. Conf. Netw. Sens. Control ICNSC, IEEE, Calabria, Italy, 2017: pp. 599–604. <https://doi.org/10.1109/ICNSC.2017.8000159>.
- [40] S. Meyer, A. Ruppen, C. Magerkurth, Internet of Things-Aware Process Modeling: Integrating IoT Devices as Business Process Resources, in: Adv. Inf. Syst. Eng. CAiSE 2013 Lect. Notes Comput. Sci., Springer International Publishing, Valencia, Spain, 2013: pp. 84–98. [https://doi.org/10.1007/978-3-642-38709-8\\_6](https://doi.org/10.1007/978-3-642-38709-8_6).
- [41] S. Meyer, A. Ruppen, L. Hilty, The Things of the Internet of Things in BPMN, in: Adv. Inf. Syst. Eng. Workshop CAiSE 2015 Lect. Notes Bus. Inf. Process., Springer International Publishing, Stockholm, Sweden, 2015: pp. 285–297. [https://doi.org/10.1007/978-3-319-19243-7\\_27](https://doi.org/10.1007/978-3-319-19243-7_27).
- [42] S. Schönig, L. Ackermann, S. Jablonski, A. Ermer, IoT meets BPM: a bidirectional communication architecture for IoT-aware process execution, Softw. Syst. Model. 19 (2020) 1443–1459. <https://doi.org/10.1007/s10270-020-00785-7>.
- [43] Object Management Group, Unified Modeling Language, Version 2.5.1, 2017.
- [44] Object Management Group, Systems Modeling Language, Version 1.6, 2019.
- [45] S.M. Fallah, S. Wolny, M. Wimmer, Towards model-integrated service-oriented manufacturing execution system, in: Proc. 2016 1st Int. Workshop Cyber-Phys. Prod. Syst. CPPS, IEEE, Vienna, Austria, 2016: pp. 1–5. <https://doi.org/10.1109/CPPS.2016.7483917>.
- [46] C.A. Petri, Kommunikationen mit Automaten, Ph.D. Thesis, University of Bonn, 1962.
- [47] W. Reisig, Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies, 1st ed., Springer-Verlag, Berlin Heidelberg, 2013. <https://doi.org/10.1007/978-3-642-33278-4>.
- [48] R. Müller, M. Scholer, M. Karkowski, Generic automation task description for flexible assembly systems, in: Proc. 52nd CIRP Conf. Manuf. Syst. CMS, Elsevier, Ljubljana, Slovenia, 2019: pp. 730–735. <https://doi.org/10.1016/j.procir.2019.03.185>.
- [49] R. Valk, Object Petri Nets: Using the Nets-within-Nets Paradigm, in: J. Desel, W. Reisig, G. Rozenberg (Eds.), Lect. Concurr. Petri Nets Adv. Petri Nets, Springer, Berlin, Heidelberg, 2004: pp. 819–848. [https://doi.org/10.1007/978-3-540-27755-2\\_23](https://doi.org/10.1007/978-3-540-27755-2_23).
- [50] J.-I. Latorre-Biel, J. Faulín, A.A. Juan, E. Jiménez-Macías, Petri Net Model of a Smart Factory in the Frame of Industry 4.0, in: Proc. 9th Vienna Int. Conf. Math. Model. MATHMOD 2018, Elsevier, Vienna, Austria, 2018: pp. 266–271. <https://doi.org/10.1016/j.ifacol.2018.03.046>.

- [51] R.J. Mayer, C.P. Menzel, M.K. Painter, P.S. Dewitte, T. Blinn, B. Perakath, IDEF3 Process Description Capture Method Report, Knowledge Based Systems, Inc., Texas, USA, 1995.
- [52] Q. Li, Y.-L. Chen, IDEF3 Process Capture Method, in: *Model. Anal. Enterp. Inf. Syst. Require. Realiz.*, Springer, Berlin, Heidelberg, 2009: pp. 159–168. [https://doi.org/10.1007/978-3-540-89556-5\\_8](https://doi.org/10.1007/978-3-540-89556-5_8).
- [53] R. Davis, E. Brabänder, eds., *The Event-driven Process Chain*, in: *ARIS Des. Platf. Get. Started BPM*, Springer, London, 2007: pp. 105–125. [https://doi.org/10.1007/978-1-84628-613-1\\_7](https://doi.org/10.1007/978-1-84628-613-1_7).
- [54] A. Fleischmann, What Is S-BPM?, in: *Commun. Comput. Inf. Sci. CCIS*, Springer, Berlin, Heidelberg, 2010: pp. 85–106. [https://doi.org/10.1007/978-3-642-15915-2\\_7](https://doi.org/10.1007/978-3-642-15915-2_7).
- [55] A. Fleischmann, C. Stary, Whom to talk to? A stakeholder perspective on business process development, *Univers. Access Inf. Soc.* 11 (2012) 125–150. <https://doi.org/10.1007/s10209-011-0236-x>.
- [56] M. Neubauer, F. Krenn, D. Majoe, C. Stary, Subject-orientation as design language for integration across organisational control layers, *Int. J. Prod. Res.* 55 (2017) 3644–3656. <https://doi.org/10.1080/00207543.2016.1198058>.
- [57] L. Wen, D. Tuffley, Formalizing Manufacturing Process Modeling Using Composition Trees, *Adv. Mater. Res.* 399–401 (2012) 1852–1855. <https://doi.org/10.4028/www.scientific.net/AMR.399-401.1852>.
- [58] D. Bork, H.-G. Fill, D. Karagiannis, W. Utz, Simulation of Multi-Stage Industrial Business Processes Using Metamodelling Building Blocks with ADOxx, *Enterp. Model. Inf. Syst. Archit.* 13 (2018) 333–344. <https://doi.org/10.18417/emisa.si.hcm.25>.
- [59] D. Jeong, D. Kim, T. Choi, Y. Seo, A Process-Based Modeling Method for Describing Production Processes of Ship Block Assembly Planning, *Processes.* 8 (2020) 880:1–880:21. <https://doi.org/10.3390/pr8070880>.
- [60] R. Petrasch, R. Hentschke, Towards an Internet-of-Things-aware Process Modeling Method: An Example for a House Surveillance System Process Model, in: *Proc. 2nd Manag. Innov. Technol. Int. Conf. MITiCON 2015, Information Technology Management*, Faculty of Engineering, Mahidol University, Bangkok, Thailand, 2015: pp. 168–172.
- [61] R. Petrasch, R. Hentschke, Process modeling for industry 4.0 applications: Towards an industry 4.0 process modeling language and method, in: *Proc. 2016 13th Int. Jt. Conf. Comput. Sci. Softw. Eng. JCSSE, IEEE, Khon Kaen, Thailand, 2016*: pp. 1–5. <https://doi.org/10.1109/JCSSE.2016.7748885>.
- [62] R. Lindorfer, R. Froschauer, G. Schwarz, ADAPT - A decision-model-based Approach for Modeling Collaborative Assembly and Manufacturing Tasks, in: *Proc. 2018 IEEE 16th Int. Conf. Ind. Inform. INDIN, IEEE, Porto, Portugal, 2018*: pp. 559–564. <https://doi.org/10.1109/INDIN.2018.8472064>.
- [63] M. Rother, J. Shook, *Learning to See: Value Stream Mapping to Create Value and Eliminate MUDA, Version 1.4*, Lean Enterprise Institute, Cambridge, MA, USA, 2009.
- [64] A. Salmi, P. David, J.D. Summers, E. Blanco, A modelling language for assembly sequences representation, scheduling and analyses, *Int. J. Prod. Res.* 52 (2014) 3986–4006. <https://doi.org/10.1080/00207543.2014.916432>.
- [65] N. Keddis, G. Kainz, A. Zoitl, A. Knoll, Modeling production workflows in a mass customization era, in: *Proc. 2015 IEEE Int. Conf. Ind. Technol. ICIT, IEEE, Seville, Spain, 2015*: pp. 1901–1906. <https://doi.org/10.1109/ICIT.2015.7125374>.
- [66] M. Lütjen, D. Rippel, GRAMOSA framework for graphical modelling and simulation-based analysis of complex production processes, *Int. J. Adv. Manuf. Technol.* 81 (2015) 171–181. <https://doi.org/10.1007/s00170-015-7037-y>.
- [67] B. Yang, L. Qiao, Z. Zhu, M. Wulan, A Metamodel for the Manufacturing Process Information Modeling, in: *Proc. 9th Int. Conf. Digit. Enterp. Technol. DET 2016 – Intell. Manuf. Knowl. Econ. Era, Nanjing, China, 2016*: pp. 332–337. <https://doi.org/10.1016/j.procir.2016.10.032>.
- [68] B. Yang, L. Qiao, N. Cai, Z. Zhu, M. Wulan, Manufacturing process information modeling using a metamodeling approach, *Int. J. Adv. Manuf. Technol.* 94 (2018) 1579–1596. <https://doi.org/10.1007/s00170-016-9979-0>.

- [69] E. Järvenpää, N. Siltala, O. Hylli, M. Lanz, The development of an ontology for describing the capabilities of manufacturing resources, *J. Intell. Manuf.* 30 (2019) 959–978. <https://doi.org/10.1007/s10845-018-1427-6>.
- [70] S. Cramer, M. Hoffmann, P. Schlegel, M. Kemmerling, R.H. Schmitt, Towards a flexible process-independent meta-model for production data, in: *Proc. 14th CIRP Conf. Intell. Comput. Manuf. Eng. ICME*, Elsevier, Gulf of Naples, Italy, 2020: pp. 586–591. <https://doi.org/10.1016/j.procir.2021.03.112>.
- [71] T.D. Brunoe, A.-L. Andersen, D.G.H. Sorensen, K. Nielsen, M. Bejlegaard, Integrated product-process modelling for platform-based co-development, *Int. J. Prod. Res.* 58 (2020) 6185–6201. <https://doi.org/10.1080/00207543.2019.1671628>.
- [72] C. Schlenoff, M. Gruninger, F. Tissot, J. Valois, J. Lubell, J. Lee, The Process Specification Language (PSL) Overview and Version 1.0 Specification, National Institute of Standards and Technology (NIST), 2000. <https://doi.org/10.6028/NIST.IR.6459>.
- [73] L. Qiao, S. Kao, Y. Zhang, Manufacturing process modelling using process specification language, *Int. J. Adv. Manuf. Technol.* 55 (2011) 549–563. <https://doi.org/10.1007/s00170-010-3115-3>.
- [74] M. Witsch, B. Vogel-Heuser, Towards a Formal Specification Framework for Manufacturing Execution Systems, *IEEE Trans. Ind. Inform.* 8 (2012) 311–320. <https://doi.org/10.1109/TII.2012.2186585>.
- [75] B. Weißenberger, S. Flad, X. Chen, S. Rösch, T. Voigt, B. Vogel-Heuser, Model driven engineering of manufacturing execution systems using a formal specification, in: *Proc. 2015 IEEE 20th Int. Conf. Emerg. Technol. Fact. Autom. ETFA*, IEEE, Luxembourg, 2015: pp. 1–8. <https://doi.org/10.1109/ETFA.2015.7301430>.
- [76] X. Chen, F. Gemein, S. Flad, T. Voigt, Basis for the model-driven engineering of manufacturing execution systems: Modeling elements in the domain of beer brewing, *Comput. Ind.* 101 (2018) 127–137. <https://doi.org/10.1016/j.compind.2018.07.005>.
- [77] X. Chen, C. Nophut, T. Voigt, A model-driven approach for engineering customizable MES with the application to the food and beverage industry, *Int. J. Adv. Manuf. Technol.* 115 (2021) 2607–2622. <https://doi.org/10.1007/s00170-021-07317-7>.
- [78] H. Lee, K. Ryu, Y.-J. Son, Y. Cho, Capturing green information and mapping with MES functions for increasing manufacturing sustainability, *Int. J. Precis. Eng. Manuf.* 15 (2014) 1709–1716. <https://doi.org/10.1007/s12541-014-0523-6>.
- [79] H. Lee, Y. Liao, S. Kim, K. Ryu, A Framework for Process Model Based Human-Robot Collaboration System Using Augmented Reality, in: *IFIP Adv. Inf. Commun. Technol. AICT*, Springer Nature, Seoul, Korea, 2018: pp. 482–489. [https://doi.org/10.1007/978-3-319-99707-0\\_60](https://doi.org/10.1007/978-3-319-99707-0_60).
- [80] H. Lee, Y.Y. Liao, S. Kim, K. Ryu, Model-Based Human Robot Collaboration System for Small Batch Assembly with a Virtual Fence, *Int. J. Precis. Eng. Manuf.-Green Technol.* 7 (2020) 609–623. <https://doi.org/10.1007/s40684-020-00214-6>.
- [81] The Association of German Engineers, The Association for Electrical, Electronic & Information Technologies, VDI/VDE 3682 Part 1 – Formalised process descriptions: Concept and graphic representation, VDI/VDE Society for Measurement and Automatic Control (GMA), Düsseldorf, Germany, 2015.
- [82] The Association of German Engineers, The Association for Electrical, Electronic & Information Technologies, VDI/VDE 3682 Part 2 – Formalised process descriptions: Information model, VDI/VDE Society for Measurement and Automatic Control (GMA), Düsseldorf, Germany, 2015.
- [83] K. Meixner, J. Decker, H. Marcher, A. Lüder, S. Biffel, Towards a Domain-Specific Language for Product-Process-Resource Constraints, in: *Proc. 25th IEEE Int. Conf. Emerg. Technol. Fact. Autom. ETFA*, IEEE, Vienna, Austria, 2020: pp. 1405–1408. <https://doi.org/10.1109/ETFA46521.2020.9212063>.
- [84] K. Meixner, F. Rinker, H. Marcher, J. Decker, S. Biffel, A Domain-Specific Language for Product-Process-Resource Modeling, in: *Proc. 26th IEEE Int. Conf. Emerg. Technol. Fact. Autom. ETFA*, IEEE, Vasteras, Sweden, 2021: pp. 1–8. <https://doi.org/10.1109/ETFA45728.2021.9613674>.

- [85] B. Caesar, A. Hänel, E. Wenkler, C. Corinth, S. Ihlenfeldt, A. Fay, Information Model of a Digital Process Twin for Machining Processes, in: Proc. 25th IEEE Int. Conf. Emerg. Technol. Fact. Autom. ETFA, IEEE, Vienna, Austria, 2020: pp. 1765–1772. <https://doi.org/10.1109/ETFA46521.2020.9212085>.
- [86] R. Seiger, C. Keller, F. Niebling, T. Schlegel, Modelling complex and flexible processes for smart cyber-physical environments, *J. Comput. Sci.* 10 (2015) 137–148. <https://doi.org/10.1016/j.jocs.2014.07.001>.
- [87] D. Brovkina, O. Riedel, Skill-based Metamodel for sustaining the process-oriented cyber-physical System Description, in: Proc. 39th Cent. Am. Panama Conv. CONCAPAN XXXIX, IEEE, Guatemala City, Guatemala, 2019: pp. 1–6. <https://doi.org/10.1109/CONCAPANXXXIX47272.2019.8976997>.
- [88] D. Brovkina, O. Riedel, Graph-based Data Model for Assembly-Specific Capability Description for Fully Automated Assembly Line Design, in: Proc. 2nd Eurasia Conf. IOT Commun. Eng. ECICE, IEEE, Yunlin, Taiwan, 2020: pp. 381–384. <https://doi.org/10.1109/ECICE50847.2020.9301960>.
- [89] D. Brovkina, O. Riedel, Assembly Process Model for Automated Assembly Line Design, in: Proc. 3rd Eurasia Conf. IOT Commun. Eng. ECICE, IEEE, Yunlin, Taiwan, 2021: pp. 588–594. <https://doi.org/10.1109/ECICE52819.2021.9645604>.
- [90] C. Lehnert, G. Engel, H. Steininger, R. Drath, T. Greiner, A Hierarchical Domain-Specific Language for Cyber-physical Production Systems Integrating Asset Administration Shells, in: Proc. 26th IEEE Int. Conf. Emerg. Technol. Fact. Autom. ETFA, IEEE, Vasteras, Sweden, 2021: pp. 1–4. <https://doi.org/10.1109/ETFA45728.2021.9613428>.
- [91] F. Gamboa Quintanilla, O. Cardin, A. L'Anton, P. Castagna, A modeling framework for manufacturing services in Service-oriented Holonic Manufacturing Systems, *Eng. Appl. Artif. Intell.* 55 (2016) 26–36. <https://doi.org/10.1016/j.engappai.2016.06.004>.
- [92] A. Indamutsa, D. Di Ruscio, A. Pierantonio, A Low-Code Development Environment to Orchestrate Model Management Services, in: IFIP Adv. Inf. Commun. Technol. AICT, Springer Nature, Nantes, France, 2021: pp. 342–350. [https://doi.org/10.1007/978-3-030-85874-2\\_36](https://doi.org/10.1007/978-3-030-85874-2_36).
- [93] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, A.S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Software Engineering Institute, Carnegie Mellon University, 1990.
- [94] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework, 2nd ed., Addison-Wesley Professional, Upper Saddle River, New Jersey, U.S., 2008.
- [95] Eclipse Foundation, Eclipse Modeling Framework. <https://www.eclipse.org/modeling/emf/> (accessed March 20, 2021).
- [96] Object Management Group, Object Constraint Language, Version 2.4, 2014.
- [97] J. Warmer, A. Kleppe, The Object Constraint Language: Getting Your Models Ready for MDA, 2nd ed., Addison-Wesley, Boston, MA, USA, 2003.
- [98] J. Cabot, M. Gogolla, Object Constraint Language (OCL): A Definitive Guide, in: Proc. 12th Int. Sch. Form. Methods Des. Comput. Commun. Softw. Syst., Springer, Bertinoro, Italy, 2012: pp. 58–90. [https://doi.org/10.1007/978-3-642-30982-3\\_3](https://doi.org/10.1007/978-3-642-30982-3_3).
- [99] V. Vujović, M. Maksimović, B. Perišić, Sirius: A Rapid Development of DSM Graphical Editor, in: Proc. IEEE 18th Int. Conf. Intell. Eng. Syst. INES 2014, IEEE, Tihany, Hungary, 2014: pp. 233–238. <https://doi.org/10.1109/INES.2014.6909375>.
- [100] Eclipse Sirius Documentation. <https://www.eclipse.org/sirius/doc/> (accessed March 19, 2020).
- [101] G. Kahraman, S. Bilgen, A framework for qualitative assessment of domain-specific languages, *Softw. Syst. Model.* 14 (2015) 1505–1526. <https://doi.org/10.1007/s10270-013-0387-8>.
- [102] M. Xu, J.M. David, S.H. Kim, The Fourth Industrial Revolution: Opportunities and Challenges, *Int. J. Financ. Res.* 9 (2018) 90–95. <https://doi.org/10.5430/ijfr.v9n2p90>.
- [103] P.N. Stearns, The Industrial Revolution in World History, 5th ed., Routledge, Taylor & Francis Group, New York and London, 2021.
- [104] H. Kagermann, W.-D. Lukas, W. Wahlster, Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution, *VDI Nachrichten.* 13 (2011).



- [105] K.-D. Thoben, S. Wiesner, T. Wuest, "Industrie 4.0" and Smart Manufacturing – A Review of Research Issues and Application Examples, *Int. J. Autom. Technol.* 11 (2017) 4–16. <https://doi.org/10.20965/ijat.2017.p0004>.
- [106] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, M. Hoffmann, *Industry 4.0*, *Bus. Inf. Syst. Eng.* 6 (2014) 239–242. <https://doi.org/10.1007/s12599-014-0334-4>.
- [107] H. Zhao, L. McLoughlin, V. Adzhiev, A. Pasko, "Why do we not buy mass customised products?" – An investigation of consumer purchase intention of mass customised products, *Int. J. Ind. Eng. Manag.* 10 (2019) 181–190. <https://doi.org/10.24867/IJEM-2019-2-238>.
- [108] D. Ivanov, C.S. Tang, A. Dolgui, D. Battini, A. Das, Researchers' perspectives on Industry 4.0: multi-disciplinary analysis and opportunities for operations management, *Int. J. Prod. Res.* 59 (2021) 2055–2078. <https://doi.org/10.1080/00207543.2020.1798035>.
- [109] Y. Lu, *Industry 4.0: A survey on technologies, applications and open research issues*, *J. Ind. Inf. Integr.* 6 (2017) 1–10. <https://doi.org/10.1016/j.jii.2017.04.005>.
- [110] A.G. Frank, L.S. Dalenogare, N.F. Ayala, *Industry 4.0 technologies: Implementation patterns in manufacturing companies*, *Int. J. Prod. Econ.* 210 (2019) 15–26. <https://doi.org/10.1016/j.ijpe.2019.01.004>.
- [111] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee, B. Yin, *Smart Factory of Industry 4.0: Key Technologies, Application Case, and Challenges*, *IEEE Access.* 6 (2018) 6505–6519. <https://doi.org/10.1109/ACCESS.2017.2783682>.
- [112] F. Almada-Lobo, *The Industry 4.0 revolution and the future of Manufacturing Execution Systems (MES)*, *J. Innov. Manag.* 3 (2015) 16–21. [https://doi.org/10.24840/2183-0606\\_003.004\\_0003](https://doi.org/10.24840/2183-0606_003.004_0003).
- [113] S. Wang, J. Wan, D. Li, C. Zhang, *Implementing Smart Factory of Industrie 4.0: An Outlook*, *Int. J. Distrib. Sens. Netw.* 12 (2016) 1–10. <https://doi.org/10.1155/2016/3159805>.
- [114] Y. Zhang, C. Qian, J. Lv, Y. Liu, *Agent and Cyber-Physical System Based Self-Organizing and Self-Adaptive Intelligent Shopfloor*, *IEEE Trans. Ind. Inform.* 13 (2017) 737–747. <https://doi.org/10.1109/TII.2016.2618892>.
- [115] N. Jazdi, *Cyber Physical Systems in the Context of Industry 4.0*, in: *Proc. 2014 IEEE Int. Conf. Autom. Qual. Test. Robot.*, IEEE, Cluj-Napoca, Romania, 2014: pp. 1–4. <https://doi.org/10.1109/AQTR.2014.6857843>.
- [116] E.A. Lee, S.A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed., MIT Press, 2017.
- [117] B.B. Sanchez, R. Alcarria, D. Sanchez-de-Rivera, A. Sanchez-Picot, *Enhancing Process Control in Industry 4.0 Scenarios using Cyber-Physical Systems*, *J. Wirel. Mob. Netw. Ubiquitous Comput. Dependable Appl. JoWUA.* 7 (2016) 41–64. <https://doi.org/10.22667/JOWUA.2016.12.31.041>.
- [118] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, K. Ueda, *Cyber-physical systems in manufacturing*, *CIRP Ann. - Manuf. Technol.* 65 (2016) 621–641. <https://doi.org/10.1016/j.cirp.2016.06.005>.
- [119] R. Stark, T. Damerau, *Digital Twin*, in: S. Chatti, T. Tolio (Eds.), *CIRP Encycl. Prod. Eng.*, Springer, Berlin, Heidelberg, 2019: pp. 1–8. [https://doi.org/10.1007/978-3-642-35950-7\\_16870-1](https://doi.org/10.1007/978-3-642-35950-7_16870-1).
- [120] Q. Qi, F. Tao, *Digital Twin and Big Data Towards Smart Manufacturing and Industry 4.0: 360 Degree Comparison*, *IEEE Access.* 6 (2018) 3585–3593. <https://doi.org/10.1109/ACCESS.2018.2793265>.
- [121] J. Wan, H. Cai, K. Zhou, *Industrie 4.0: Enabling Technologies*, in: *Proc. 2015 Int. Conf. Intell. Comput. Internet Things ICIT*, IEEE, Harbin, China, 2015: pp. 135–140. <https://doi.org/10.1109/ICAIOT.2015.7111555>.
- [122] S. Vaidya, P. Ambad, S. Bhosle, *Industry 4.0 – A Glimpse*, in: *Procedia Manuf.*, Maharashtra, India, 2018: pp. 233–238. <https://doi.org/10.1016/j.promfg.2018.02.034>.
- [123] S. Li, L.D. Xu, S. Zhao, *The internet of things: a survey*, *Inf. Syst. Front.* 17 (2015) 243–259. <https://doi.org/10.1007/s10796-014-9492-7>.
- [124] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, M. Gidlund, *Industrial Internet of Things: Challenges, Opportunities, and Directions*, *IEEE Trans. Ind. Inform.* 14 (2018) 4724–4734. <https://doi.org/10.1109/TII.2018.2852491>.

- [125] B. Saenz de Ugarte, A. Artiba, R. Pellerin, Manufacturing execution system – a literature review, *Prod. Plan. Control.* 20 (2009) 525–539. <https://doi.org/10.1080/09537280902938613>.
- [126] S. Mantravadi, C. Møller, An Overview of Next-generation Manufacturing Execution Systems: How important is MES for Industry 4.0?, *Procedia Manuf.* 30 (2019) 588–595. <https://doi.org/10.1016/j.promfg.2019.02.083>.
- [127] F.R. Jacobs, W.L. Berry, D.C. Whybark, T.E. Vollmann, *Manufacturing Planning and Control for Supply Chain Management: The CPIM Reference*, 2nd ed., McGraw-Hill Education, 2018.
- [128] L. Alting, H. Zhang, Computer Aided Process Planning: the state-of-the-art survey, *Int. J. Prod. Res.* 27 (1989) 553–585. <https://doi.org/10.1080/00207548908942569>.
- [129] F. Cay, C. Chassapis, An IT view on perspectives of computer aided process planning research, *Comput. Ind.* 34 (1997) 307–337. [https://doi.org/10.1016/S0166-3615\(97\)00070-5](https://doi.org/10.1016/S0166-3615(97)00070-5).
- [130] X. Xu, L. Wang, S.T. Newman, Computer-aided process planning – A critical review of recent developments and future trends, *Int. J. Comput. Integr. Manuf.* 24 (2011) 1–31. <https://doi.org/10.1080/0951192X.2010.518632>.
- [131] M. Lundgren, M. Hedlind, T. Kjellberg, Model Driven Manufacturing Process Design and Managing Quality, in: *Proc. 26th CIRP Des. Conf.*, Elsevier, Stockholm, Sweden, 2016: pp. 299–304. <https://doi.org/10.1016/j.procir.2016.07.032>.
- [132] M. Lundgren, M. Hedlind, G. Sivard, T. Kjellberg, Process Design as Fundament in Efficient Process Planning, in: *Proc. 8th Swed. Prod. Symp. SPS 2018*, Elsevier, Stockholm, Sweden, 2018: pp. 487–494. <https://doi.org/10.1016/j.promfg.2018.06.126>.
- [133] C.T. Maravelias, C. Sung, Integration of production planning and scheduling: Overview, challenges and opportunities, *Comput. Chem. Eng.* 33 (2009) 1919–1930. <https://doi.org/10.1016/j.compchemeng.2009.06.007>.
- [134] S.C. Graves, A Review of Production Scheduling, *Oper. Res.* 29 (1981) 646–675. <https://doi.org/10.1287/opre.29.4.646>.
- [135] B.W. Niebel, Mechanized process selection for planning new design, ASME Pap. No 737. (1965).
- [136] D.E. Schenk, Feasibility of automated process planning, Ph.D. Thesis, Purdue University, 1966.
- [137] Y. Yusof, K. Latif, Survey on computer-aided process planning, *Int. J. Adv. Manuf. Technol.* 75 (2014) 77–89. <https://doi.org/10.1007/s00170-014-6073-3>.
- [138] C. Elanchezian, T.S. Selwyn, G.S. Sundar, *Computer Aided Manufacturing*, 2nd ed., Laxmi Publications, 2007.
- [139] M. Mani, J. Madan, J.H. Lee, K.W. Lyons, S.K. Gupta, Sustainability characterisation for manufacturing processes, *Int. J. Prod. Res.* 52 (2014) 5895–5912. <https://doi.org/10.1080/00207543.2014.886788>.
- [140] F. Jovane, E. Westkämper, D. Williams, *The ManuFuture Road: Towards Competitive and Sustainable High-Adding-Value Manufacturing*, 1st ed., Springer, Berlin, Heidelberg, 2009. <https://doi.org/10.1007/978-3-540-77012-1>.
- [141] I.C. Garretson, M. Mani, S. Leong, K.W. Lyons, K.R. Haapala, Terminology to support manufacturing process characterization and assessment for sustainable production, *J. Clean. Prod.* 139 (2016) 986–1000. <https://doi.org/10.1016/j.jclepro.2016.08.103>.
- [142] Platform Industrie 4.0, ZVEI, Details of the Asset Administration Shell. Part 1 – The exchange of information between partners in the value chain of Industrie 4.0 (Version 3.0RC01), Federal Ministry for Economic Affairs and Energy (BMWi), Berlin, Germany, 2020.
- [143] C. Diedrich, A. Belyaev, R. Blumenfeld, J. Bock, S. Grimm, J. Hermann, T. Klausmann, A. Köcher, M. Maurmaier, K. Meixner, J. Peschke, M. Schleipen, S. Schmitt, B. Schnebel, G. Stephan, M. Volkman, A. Wannagat, K. Watson, M. Winter, P. Zimmermann, *Information Model for Capabilities, Skills & Services*, Plattform Industrie 4.0, Berlin, Germany, 2022.
- [144] S. Malakuti, J. Bock, M. Weser, P. Venet, P. Zimmermann, M. Wiegand, J. Grothoff, C. Wagner, A. Bayha, Challenges in Skill-based Engineering of Industrial Automation Systems, in: *Proc. 23rd Int. Conf. Emerg. Technol. Fact. Autom. ETFA*, IEEE, Turin, Italy, 2018: pp. 67–74. <https://doi.org/10.1109/ETFA.2018.8502635>.
- [145] V. Hammerstingl, G. Reinhart, *Skills in Assembly*, Version 1.1, Institute for Machine Tools and Industrial Management (iwb), Technical University of Munich, Munich, Germany, 2018.

- [146] P. Zimmermann, E. Axmann, B. Brandenbourger, K. Dorofeev, A. Mankowski, P. Zanini, Skill-based Engineering and Control on Field-Device-Level with OPC UA, in: Proc. 24th IEEE Int. Conf. Emerg. Technol. Fact. Autom. ETFA, IEEE, Zaragoza, Spain, 2019: pp. 1101–1108. <https://doi.org/10.1109/ETFA.2019.8869473>.
- [147] The Association of German Engineers, VDI 2860 – Assembly and handling; handling functions, handling units; terminology, definitions and symbols, VDI-Gesellschaft Produktionstechnik (ADB), Düsseldorf, Germany, 1990.
- [148] The German Institute for Standardization, DIN 8580 – Manufacturing processes: Terms and definitions, division, The DIN Standards Committee Technical Fundamentals (NATG), Berlin, Germany, 2003.
- [149] J. Pfrommer, D. Stogl, K. Aleksandrov, V. Schubert, B. Hein, Modelling and orchestration of service-based manufacturing systems via skills, in: Proc. 19th Int. Conf. Emerg. Technol. Fact. Autom. ETFA, IEEE, Barcelona, Spain, 2014: pp. 1–4. <https://doi.org/10.1109/ETFA.2014.7005285>.
- [150] T. Stahl, M. Voelter, K. Czarnecki, Model-Driven Software Development: Technology, Engineering, Management, 1st ed., John Wiley and Sons, Ltd., Chichester, England, 2006.
- [151] J. Bezivin, O. Gerbe, Towards a Precise Definition of the OMG/MDA Framework, in: Proc. 16th Annu. Int. Conf. Autom. Softw. Eng. ASE 2001, IEEE, San Diego, CA, USA, 2001: pp. 273–280. <https://doi.org/10.1109/ASE.2001.989813>.
- [152] C. Atkinson, T. Kuhne, Model-Driven Development: A Metamodeling Foundation, IEEE Softw. 20 (2003) 36–41. <https://doi.org/10.1109/MS.2003.1231149>.
- [153] Object Management Group, Meta Object Facility, Version 2.5.1, 2016.
- [154] A. van Deursen, P. Klint, J. Visser, Domain-Specific Languages: An Annotated Bibliography, ACM SIGPLAN Not. 35 (2000) 26–36. <https://doi.org/10.1145/352029.352035>.
- [155] M. Mernik, J. Heering, A.M. Sloane, When and how to develop domain-specific languages, ACM Comput. Surv. 37 (2005) 316–344. <https://doi.org/10.1145/1118890.1118892>.
- [156] V. Dimitrieski, Model-Driven Technical Space Integration Based on a Mapping Approach, Ph.D. Thesis, University of Novi Sad, Faculty of Technical Sciences, 2017.
- [157] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, 2nd ed., Morgan & Claypool Publishers, 2017. <https://doi.org/10.2200/S00751ED2V01Y201701SWE004>.
- [158] B. Vallespir, Y. Ducq, Enterprise modelling: from early languages to models transformation, Int. J. Prod. Res. 56 (2018) 2878–2896. <https://doi.org/10.1080/00207543.2017.1418985>.
- [159] G. Zacharewicz, N. Daclin, G. Doumeings, H. Haidar, Model Driven Interoperability for System Engineering, Modelling. 1 (2020) 94–121. <https://doi.org/10.3390/modelling1020007>.
- [160] K. Dorofeev, S. Profanter, J. Cabral, P. Ferreira, A. Zoitl, Agile Operational Behavior for the Control-Level Devices in Plug&Produce Production Environments, in: Proc. 24th IEEE Int. Conf. Emerg. Technol. Fact. Autom. ETFA, IEEE, Zaragoza, Spain, 2019: pp. 49–56. <https://doi.org/10.1109/ETFA.2019.8869208>.
- [161] D. Gorecky, M. Schmitt, M. Loskyll, D. Zühlke, Human-Machine-Interaction in the Industry 4.0 Era, in: Proc. 12th IEEE Int. Conf. Ind. Inform. INDIN, IEEE, Porto Alegre, Brazil, 2014: pp. 289–294. <https://doi.org/10.1109/INDIN.2014.6945523>.
- [162] H.A. Simon, The Sciences of the Artificial, 3rd ed., MIT Press, Cambridge, MA, USA, 1996.
- [163] S.T. March, G.F. Smith, Design and natural science research on information technology, Decis. Support Syst. 15 (1995) 251–266. [https://doi.org/10.1016/0167-9236\(94\)00041-2](https://doi.org/10.1016/0167-9236(94)00041-2).
- [164] A.R. Hevner, S.T. March, J. Park, S. Ram, Design Science in Information Systems Research, MIS Q. 28 (2004) 75–105. <https://doi.org/10.2307/25148625>.
- [165] J. vom Brocke, A. Hevner, A. Maedche, Introduction to Design Science Research, in: J. vom Brocke, A. Hevner, A. Maedche (Eds.), Des. Sci. Res. Cases, Springer, Cham, 2020: pp. 1–13. [https://doi.org/10.1007/978-3-030-46781-4\\_1](https://doi.org/10.1007/978-3-030-46781-4_1).
- [166] K. Peffers, T. Tuunanen, M.A. Rothenberger, S. Chatterjee, A Design Science Research Methodology for Information Systems Research, J. Manag. Inf. Syst. 24 (2007) 45–77. <https://doi.org/10.2753/MIS0742-1222240302>.

- [167] I. Luković, P. Mogin, J. Pavićević, S. Ristić, An approach to developing complex database schemas using form types, *Softw. Pract. Exp.* 37 (2007) 1621–1656. <https://doi.org/10.1002/spe.820>.
- [168] I. Luković, A. Popović, J. Mostić, S. Ristić, A tool for modeling form type check constraints and complex functionalities of business applications, *Comput. Sci. Inf. Syst.* 7 (2010) 359–385. <https://doi.org/10.2298/CSIS1002359L>.
- [169] M. Čeliković, I. Luković, S. Aleksić, V. Ivančević, A MOF based meta-model and a concrete DSL syntax of IIS\*Case PIM concepts, *Comput. Sci. Inf. Syst.* 9 (2012) 1075–1103. <https://doi.org/10.2298/CSIS120203034C>.
- [170] S. Ristić, S. Aleksić, M. Čeliković, V. Dimitrieski, I. Luković, Database reverse engineering based on meta-models, *Open Comput. Sci.* 4 (2014) 150–159. <https://doi.org/10.2478/s13537-014-0218-1>.
- [171] I. Luković, M. Čeliković, S. Kordić, M. Vještica, An Approach to the Information System Conceptual Modeling Based on the Form Types, in: D. Karagiannis, M. Lee, K. Hinkelmann, W. Utz (Eds.), *Domain-Specific Conceptual Modeling: Concepts, Methods and ADOxx Tools*, 1st ed., Springer International Publishing, Cham, 2022: pp. 589–614. [https://doi.org/10.1007/978-3-030-93547-4\\_26](https://doi.org/10.1007/978-3-030-93547-4_26).
- [172] ADOxx, ADOxx Modeling and Configuration Platform. <https://www.adoxx.org/live/home> (accessed March 20, 2021).
- [173] H.-G. Fill, D. Karagiannis, On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform, *Enterp. Model. Inf. Syst. Archit.* 8 (2013) 4–25. <https://doi.org/10.1007/BF03345926>.
- [174] H.-G. Fill, T. Redmond, D. Karagiannis, FDMM: A Formalism for Describing ADOxx Meta Models and Models, in: *Proc. 14th Int. Conf. Enterp. Inf. Syst., SciTePress – Science and Technology Publications, Wroclaw, Poland, 2012*: pp. 133–144. <https://doi.org/10.5220/0003971201330144>.
- [175] B. Terzić, V. Dimitrieski, S. Kordić, G. Milosavljević, I. Luković, Development and evaluation of MicroBuilder: a Model-Driven tool for the specification of REST Microservice Software Architectures, *Enterp. Inf. Syst.* 12 (2018) 1034–1057. <https://doi.org/10.1080/17517575.2018.1460766>.
- [176] H. Kern, F. Stefan, V. Dimitrieski, M. Čeliković, Mapping-Based Exchange of Models Between Meta-Modeling Tools, in: *Proc. 14th Workshop Domain-Specif. Model., ACM, Portland, Oregon, USA, 2014*: pp. 29–34. <https://doi.org/10.1145/2688447.2688453>.
- [177] V. Dimitrieski, M. Čeliković, N. Igić, H. Kern, F. Stefan, Reuse of Rules in a Mapping-Based Integration Tool, in: *Commun. Comput. Inf. Sci., Springer, Naples, Italy, 2015*: pp. 269–281. [https://doi.org/10.1007/978-3-319-22689-7\\_20](https://doi.org/10.1007/978-3-319-22689-7_20).
- [178] H. Kern, F. Stefan, K.-P. Fähnrich, V. Dimitrieski, A Mapping-Based Framework for the Integration of Machine Data and Information Systems, in: *Proc. 8th IADIS Int. Conf. Inf. Syst., 2015*: pp. 113–120.
- [179] H. Kern, F. Stefan, V. Dimitrieski, Intelligent and Self-Adapting Integration Between Machines and Information Systems, *IADIS Int. J. Comput. Sci. Inf. Syst.* 10 (2015) 47–63.
- [180] V. Djukić, I. Luković, A. Popović, Domain-Specific Modeling in Document Engineering, in: *Proc. 2011 Fed. Conf. Comput. Sci. Inf. Syst. FedCSIS 2011, Polish Information Processing Society, Szczecin, Poland, 2011*: pp. 817–824.
- [181] V. Djukić, I. Luković, A. Popović, V. Ivančević, Model execution: An approach based on extending domain-specific modeling with action reports, *Comput. Sci. Inf. Syst.* 10 (2013) 1585–1620. <https://doi.org/10.2298/CSIS121228059D>.
- [182] V. Djukić, A. Popović, I. Luković, V. Ivančević, Model Variations and Automated Refinement of Domain-Specific Modeling Languages for Robot-Motion Control, *Comput. Inform.* 38 (2019) 497–524. [https://doi.org/10.31577/cai\\_2019\\_2\\_497](https://doi.org/10.31577/cai_2019_2_497).
- [183] A. Wortmann, O. Barais, B. Combemale, M. Wimmer, Modeling Languages in Industry 4.0: An Extended Systematic Mapping Study, *Softw. Syst. Model.* 19 (2020) 67–94. <https://doi.org/10.1007/s10270-019-00757-6>.

- [184] M.A. Mohamed, M. Challenger, G. Kardas, Applications of model-driven engineering in cyber-physical systems: A systematic mapping study, *J. Comput. Lang.* 59 (2020) 100972:1–100972:19. <https://doi.org/10.1016/j.cola.2020.100972>.
- [185] M.A. Mohamed, G. Kardas, M. Challenger, Model-Driven Engineering Tools and Languages for Cyber-Physical Systems—A Systematic Literature Review, *IEEE Access.* 9 (2021) 48605–48630. <https://doi.org/10.1109/ACCESS.2021.3068358>.
- [186] G. Sebastián, J.A. Gallud, R. Tesoriero, Code generation using model driven architecture: A systematic mapping study, *J. Comput. Lang.* 56 (2020) 100935:1–100935:11. <https://doi.org/10.1016/j.cola.2019.100935>.
- [187] E. de Araújo Silva, E. Valentin, J.R.H. Carvalho, R. da Silva Barreto, A survey of Model Driven Engineering in robotics, *J. Comput. Lang.* 62 (2021) 101021:1–101021:14. <https://doi.org/10.1016/j.cola.2020.101021>.
- [188] G.L. Casalaro, G. Cattivera, F. Ciccozzi, I. Malavolta, A. Wortmann, P. Pelliccione, Model-driven engineering for mobile robotic systems: a systematic mapping study, *Softw. Syst. Model.* 21 (2022) 19–49. <https://doi.org/10.1007/s10270-021-00908-8>.
- [189] C. Raith, M. Woschank, H. Zsifkovits, Metamodeling in Manufacturing Systems: Literature Review and Trends, in: *Proc. 11th Int. Conf. Ind. Eng. Oper. Manag.*, IEOM Society International, Singapore, 2021: pp. 831–842.
- [190] A. Ledeczki, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, P. Volgyesi, The Generic Modeling Environment, in: *Proc. IEEE Int. Workshop Intell. Signal Process. WISP2001*, IEEE, Budapest, Hungary, 2001.
- [191] Vanderbilt University, Institute for Software Integrated Systems, GME: Generic Modeling Environment. <https://www.isis.vanderbilt.edu/projects/GME> (accessed April 22, 2022).
- [192] H. Krahn, B. Rumpe, S. Völkel, MontiCore: a framework for compositional development of domain specific languages, *Int. J. Softw. Tools Technol. Transf.* 12 (2010) 353–372. <https://doi.org/10.1007/s10009-010-0142-1>.
- [193] MontiCore. <https://monticore.github.io/monticore/> (accessed April 22, 2022).
- [194] S. Kelly, K. Lyytinen, M. Rossi, MetaEdit+ A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment, in: *Adv. Inf. Syst. Eng.*, Springer, Berlin, Heidelberg, 1996: pp. 1–21. [https://doi.org/10.1007/3-540-61292-0\\_1](https://doi.org/10.1007/3-540-61292-0_1).
- [195] MetaCase, MetaEdit+ Domain-Specific Modeling (DSM) environment. <https://www.metacase.com/products.html> (accessed April 24, 2022).
- [196] Eclipse Foundation, Xtext. <https://www.eclipse.org/Xtext/> (accessed December 29, 2021).
- [197] M.A. Wehrmeister, E.P. Freitas, C.E. Pereira, F. Rammig, GenERTiCA: A Tool for Code Generation and Aspects Weaving, in: *Proc. 11th IEEE Int. Symp. Object Compon.-Oriented Real-Time Distrib. Comput. ISORC*, IEEE, Orlando, FL, USA, 2008: pp. 234–238. <https://doi.org/10.1109/ISORC.2008.67>.
- [198] Eclipse Foundation, Xtend. <https://www.eclipse.org/xtend/> (accessed March 20, 2021).
- [199] Z. Gao, C. Cecati, S.X. Ding, A Survey of Fault Diagnosis and Fault-Tolerant Techniques—Part I: Fault Diagnosis With Model-Based and Signal-Based Approaches, *IEEE Trans. Ind. Electron.* 62 (2015) 3757–3767. <https://doi.org/10.1109/TIE.2015.2417501>.
- [200] A. Farooqui, P. Bergagard, P. Falkman, M. Fabian, Error handling within highly automated automotive industry: Current practice and research needs, in: *Proc. 2016 IEEE 21st Int. Conf. Emerg. Technol. Fact. Autom. ETFA*, IEEE, Berlin, Germany, 2016: pp. 1–4. <https://doi.org/10.1109/ETFA.2016.7733628>.
- [201] M. Švingerová, M. Melichar, Evaluation of Process Risks in Industry 4.0 Environment, in: *Proc. 28th DAAAM Int. Symp. Intell. Manuf. Autom.*, DAAAM International, Zadar, Croatia, 2017: pp. 1021–1029. <https://doi.org/10.2507/28th.daaam.proceedings.142>.
- [202] A. Barthelmey, D. Störkle, B. Kuhlenkötter, J. Deuse, Cyber Physical Systems for Life Cycle Continuous Technical Documentation of Manufacturing Facilities, in: *Proc. 47th CIRP Conf. Manuf. Syst.*, Elsevier, 2014: pp. 207–211. <https://doi.org/10.1016/j.procir.2014.01.050>.
- [203] E.G. Margherita, A.M. Braccini, Industry 4.0 Technologies in Flexible Manufacturing for Sustainable Organizational Value: Reflections from a Multiple Case Study of Italian Manufacturers, *Inf. Syst. Front.* 25 (2023) 995–1016. <https://doi.org/10.1007/s10796-020-10047-y>.

- [204] L. Li, Reskilling and Upskilling the Future-ready Workforce for Industry 4.0 and Beyond, *Inf. Syst. Front.* (2022). <https://doi.org/10.1007/s10796-022-10308-y>.
- [205] M. Wolf, M. Kleindienst, C. Ramsauer, C. Zierler, E. Winter, Current and Future Industrial Challenges: Demographic Change and Measures for Elderly Workers in Industry 4.0, *Ann. Fac. Eng. Hunedoara – Int. J. Eng.* 16 (2018) 67–76.
- [206] M. Nardo, D. Forino, T. Murino, The evolution of man–machine interaction: the role of human in Industry 4.0 paradigm, *Prod. Manuf. Res.* 8 (2020) 20–34. <https://doi.org/10.1080/21693277.2020.1737592>.
- [207] C. Wohlin, Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering, in: *Proc. 18th Int. Conf. Eval. Assess. Softw. Eng. EASE 14*, Association for Computing Machinery, London, England, United Kingdom, 2014: pp. 38:1–38:10. <https://doi.org/10.1145/2601248.2601268>.
- [208] K. Zarour, D. Benmerzoug, N. Guermouche, K. Drira, A systematic literature review on BPMN extensions, *Bus. Process Manag. J.* 26 (2019) 1473–1503. <https://doi.org/10.1108/BPMJ-01-2019-0040>.
- [209] American Production and Inventory Control Society, *APICS Dictionary: The essential supply chain reference*, 14th ed., APICS, Chicago, USA, 2013.
- [210] L.A. Shah, A. Etienne, A. Siadat, F. Vernadat, Process-oriented risk assessment methodology for manufacturing process evaluation, *Int. J. Prod. Res.* 55 (2017) 4516–4529. <https://doi.org/10.1080/00207543.2016.1268728>.
- [211] M.K. Sott, L.B. Furstenau, L.M. Kipper, Y.P. Reckziegel Rodrigues, J.R. López-Robles, F.D. Giraldo, M.J. Cobo, Process modeling for smart factories: using science mapping to understand the strategic themes, main challenges and future trends, *Bus. Process Manag. J.* 27 (2021) 1391–1417. <https://doi.org/10.1108/BPMJ-05-2020-0181>.
- [212] A. García-Domínguez, M. Marcos-Bárcena, I. Medina-Bulo, A Comparison of BPMN 2.0 with Other Notations for Manufacturing Processes, in: *Proc. 4th Manuf. Eng. Soc. Int. Conf. MESIC 2011*, AIP Publishing, Cadiz, Spain, 2012: pp. 593–600. <https://doi.org/10.1063/1.4707613>.
- [213] Yaoqiang BPMN Editor, Version 5.5.1. <http://bpmn.sourceforge.net/> (accessed January 9, 2022).
- [214] Netherlands Organization for Applied Scientific Research (TNO), Checklist Physical Load. <https://www.fysiekebelasting.tno.nl/en/instrumenten/checklist-physical-load/> (accessed September 25, 2020).
- [215] OASIS, *Web Services Business Process Execution Language, Version 2.0*, 2007.
- [216] S. Meyer, K. Sperner, C. Magerkurth, J. Pasquier, Towards modeling real-world aware business processes, in: *Proc. Second Int. Workshop Web Things*, Association for Computing Machinery, San Francisco, California, USA, 2011: pp. 8:1–8:6. <https://doi.org/10.1145/1993966.1993978>.
- [217] JGraph Ltd, *diagrams.net – Diagram Software and Flowchart Maker, Version 16.2.4*. <https://www.diagrams.net/> (accessed January 9, 2022).
- [218] W. Mahnke, S.-H. Leitner, M. Damm, *OPC Unified Architecture*, Springer, Berlin, Heidelberg, 2009. <https://doi.org/10.1007/978-3-540-68899-0>.
- [219] OPC Foundation, *Open Platform Communications Unified Architecture*. <https://opcfoundation.org/about/opc-technologies/opc-ua/> (accessed March 13, 2022).
- [220] Platform Independent Petri net Editor 2 – PIPE2, Version 4.3.0. <http://pipe2.sourceforge.net/> (accessed January 9, 2022).
- [221] P. Bonet, C.M. Llado, R. Puigjaner, W.J. Knottenbelt, PIPE v2.5: a Petri Net Tool for Performance Modeling, in: *Proc. 23rd Lat. Am. Conf. Inform. CLEI 2007*, San Jose, Costa Rica, 2007: pp. 1–12.
- [222] N.J. Dingle, W.J. Knottenbelt, T. Suto, PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets, *ACM SIGMETRICS Perform. Eval. Rev.* 36 (2009) 34–39. <https://doi.org/10.1145/1530873.1530881>.
- [223] S. Bechhofer, F. Van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein, *OWL Web Ontology Language Reference, W3C Recommendation*, 2004.

- [224] U.S. Department of Commerce, National Institute of Standards and Technology (NIST). <https://www.nist.gov/> (accessed February 19, 2022).
- [225] Eclipse Foundation, Graphiti. <https://www.eclipse.org/graphiti/> (accessed February 22, 2022).
- [226] Vaticle, TypeDB. <https://vaticle.com/> (accessed April 14, 2022).
- [227] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Ng, ROS: an open-source Robot Operating System, in: Proc. IEEE Int. Conf. Robot. Autom. ICRA2009 Workshop Open Source Softw., IEEE, Kobe, Japan, 2009: pp. 1–6.
- [228] Open Robotics, Robot Operating System (ROS). <https://www.ros.org/> (accessed April 14, 2022).
- [229] N. Koenig, A. Howard, Design and use paradigms for Gazebo, an open-source multi-robot simulator, in: Proc. 2004 IEEE/RSJ Int. Conf. Intell. Robots Syst. IROS, IEEE, Sendai, Japan, 2004: pp. 2149–2154. <https://doi.org/10.1109/IROS.2004.1389727>.
- [230] Open Source Robotics Foundation, Gazebo. <http://gazebosim.org/> (accessed April 15, 2022).
- [231] T. Qu, S.P. Lei, Z.Z. Wang, D.X. Nie, X. Chen, G.Q. Huang, IoT-based real-time production logistics synchronization system under smart cloud manufacturing, *Int. J. Adv. Manuf. Technol.* 84 (2016) 147–164. <https://doi.org/10.1007/s00170-015-7220-1>.
- [232] K. Czarnecki, S. Helsen, U. Eisenecker, Formalizing cardinality-based feature models and their specialization, *Softw. Process Improv. Pract.* 10 (2005) 7–29. <https://doi.org/10.1002/spip.213>.
- [233] J. Píkl, A. Bossert, Yet Another Feature Modeling Tool (YAFMT), (2017). <https://github.com/anb0s/YAFMT> (accessed March 10, 2023).
- [234] L. Samimi-Dehkordi, B. Zamani, S. Kolahdouz-Rahimi, Leveraging product line engineering for the development of domain-specific metamodeling languages, *J. Comput. Lang.* 51 (2019) 193–213. <https://doi.org/10.1016/j.cola.2019.02.006>.
- [235] METOP GmbH, Otto-von-Guericke-University Magdeburg, FeatureIDE. <https://www.featureide.de/> (accessed August 12, 2022).
- [236] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, T. Leich, FeatureIDE: An extensible framework for feature-oriented software development, *Sci. Comput. Program.* 79 (2014) 70–85. <https://doi.org/10.1016/j.scico.2012.06.002>.
- [237] M. Faber, J. Bützler, C.M. Schlick, Human-robot Cooperation in Future Production Systems: Analysis of Requirements for Designing an Ergonomic Work System, in: Proc. 6th Int. Conf. Appl. Hum. Factors Ergon. AHFE 2015 Affil. Conf., Elsevier, Las Vegas, Nevada, USA, 2015: pp. 510–517. <https://doi.org/10.1016/j.promfg.2015.07.215>.
- [238] D.A. Dinero, Training Within Industry: The Foundation of Lean, 1st ed., CRC Press, Taylor & Francis Group, New York, 2005. <https://doi.org/10.1201/b18692>.
- [239] I. Dejanovic, M. Tumbas, G. Milosavljevic, B. Perisic, Comparison of Textual and Visual Notations of DOMMLite Domain-Specific Language, in: Local Proc. Fourteenth East-Eur. Conf. Adv. Databases Inf. Syst., Novi Sad, Serbia, 2010: pp. 131–136.
- [240] M. Kocbek, G. Jošt, M. Heričko, G. Polančič, Business process model and notation: The current state of affairs, *Comput. Sci. Inf. Syst.* 12 (2015) 509–539. <https://doi.org/10.2298/CSIS140610006K>.
- [241] W. Behutiye, P. Karhapää, L. López, X. Burgués, S. Martínez-Fernández, A.M. Vollmer, P. Rodríguez, X. Franch, M. Oivo, Management of quality requirements in agile and rapid software development: A systematic mapping study, *Inf. Softw. Technol.* 123 (2020) 106225:1–106225:23. <https://doi.org/10.1016/j.infsof.2019.106225>.
- [242] I. Salman, A.T. Misirli, N. Juristo, Are Students Representatives of Professionals in Software Engineering Experiments?, in: Proc. 2015 IEEE/ACM 37th IEEE Int. Conf. Softw. Eng., IEEE, Florence, Italy, 2015: pp. 666–676. <https://doi.org/10.1109/ICSE.2015.82>.
- [243] V. Dimitrieski, M. Čeliković, S. Aleksić, S. Ristić, A. Alargt, I. Luković, Concepts and evaluation of the extended entity-relationship approach to database design in a multi-paradigm information system modeling tool, *Comput. Lang. Syst. Struct.* 44 (2015) 299–318. <https://doi.org/10.1016/j.cl.2015.08.011>.
- [244] International Organization for Standardization and International Electrotechnical Commission, ISO/IEC 25010:2011, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, 2011.

- [245] A.B. Chaudhuri, *Flowchart and Algorithm Basics: The Art of Programming*, Mercury Learning and Information, Dulles, VA, 2020.
- [246] M. Toghraei, *Piping and Instrumentation Diagram Development*, 1st ed., John Wiley & Sons, Inc., Hoboken, NJ, USA, 2019. <https://doi.org/10.1002/9781119329503>.
- [247] Novalys, SAP PowerDesigner. <https://www.powerdesigner.biz/> (accessed September 30, 2022).
- [248] Oracle, Oracle SQL Developer Data Modeler. <https://www.oracle.com/database/sqldeveloper/technologies/sql-data-modeler/> (accessed September 30, 2022).
- [249] Camunda, Camunda Platform. <https://camunda.com/> (accessed October 2, 2022).
- [250] Dassault Systèmes, CATIA – No Magic. <https://www.3ds.com/products-services/catia/products/no-magic/> (accessed October 2, 2022).
- [251] Oracle, Oracle Designer. <https://www.oracle.com/database/technologies/developer-tools/designer.html> (accessed October 2, 2022).
- [252] Microsoft, Microsoft Visio. <https://www.microsoft.com/en-us/microsoft-365/visio/flowchart-software> (accessed October 2, 2022).
- [253] Autodesk, AutoCAD. <https://web.autocad.com/> (accessed October 2, 2022).
- [254] Sparx Systems, Enterprise Architect. <https://sparxsystems.com/> (accessed October 2, 2022).
- [255] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, Í. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0: fundamental algorithms for scientific computing in Python, *Nat. Methods*. 17 (2020) 261–272. <https://doi.org/10.1038/s41592-019-0686-2>.
- [256] C. Spearman, The proof and measurement of association between two things, *Am. J. Psychol.* 15 (1904) 72–101. <https://doi.org/10.2307/1412159>.
- [257] D. Narandžić, I. Spasojević, T. Lolić, D. Stefanović, S. Ristić, Human Roles, Competencies and Skills in Industry 4.0: Systematic Literature Review, in: *Proc. 32nd Cent. Eur. Conf. Inf. Intell. Syst. CECIIS*, University of Zagreb, Faculty of Organization and Informatics, Varaždin, Croatia, 2021: pp. 359–369.
- [258] G. Kiš, N. Todorović, V. Dimitrieski, A Model-Driven Approach to Establishment of DLT Networks Based on a Description of Collaborative Production Processes, in: *Proc. 12th Int. Conf. Inf. Soc. Technol. ICIST 2022*, Information Society of Serbia – ISOS, Kopaonik, Serbia, 2022: pp. 110–115.
- [259] A. Neb, Review on Approaches to Generate Assembly Sequences by Extraction of Assembly Features from 3D Models, in: *Proc. 52nd CIRP Conf. Manuf. Syst. CMS*, Elsevier, Ljubljana, Slovenia, 2019: pp. 856–861. <https://doi.org/10.1016/j.procir.2019.03.213>.
- [260] D. Gors, J. Put, B. Vanherle, M. Witters, K. Luyten, Semi-automatic extraction of digital work instructions from CAD models, in: *Proc. 8th CIRP Conf. Assem. Technol. Syst.*, Elsevier, Athens, Greece, 2020: pp. 39–44. <https://doi.org/10.1016/j.procir.2020.05.202>.
- [261] C. Gonnermann, D. Gebauer, R. Daub, CAD-Based Feature Recognition for Process Monitoring Planning in Assembly, *Appl. Sci.* 13 (2023) 990:1–990:19. <https://doi.org/10.3390/app13020990>.
- [262] A. Ahmad, M. Haslgrübler, A. Ferscha, B. Ettinger, J. Cho, Macro workstep detection for assembly manufacturing, in: *Proc. 13th ACM Int. Conf. Pervasive Technol. Relat. Assist. Environ. PETRA 20*, ACM, Corfu, Greece, 2020: pp. 41:1–41:6. <https://doi.org/10.1145/3389189.3397976>.
- [263] R. Seiger, R. Kühn, M. Korzetz, U. Aßmann, HoloFlows: modelling of processes for the Internet of Things in mixed reality, *Softw. Syst. Model.* 20 (2021) 1465–1489. <https://doi.org/10.1007/s10270-020-00859-6>.
- [264] S. Hoffmann, A.F. Pinatti de Carvalho, M. Schweitzer, N.D. Abele, V. Wulf, Producing and Consuming Instructional Material in Manufacturing Contexts: Evaluation of an AR-based



- Cyber-Physical Production System for Supporting Knowledge and Expertise Sharing, Proc. ACM Hum.-Comput. Interact. 6 (2022) 366:1–366:36. <https://doi.org/10.1145/3555091>.
- [265] T. Lavric, E. Bricard, M. Preda, T. Zaharia, A Low-Cost AR Training System for Manual Assembly Operations, Comput. Sci. Inf. Syst. 19 (2022) 1047–1073. <https://doi.org/10.2298/CSIS211123013L>.



## Appendix A. Evaluation Experiment Tasks and Questionnaire

In this appendix, we present the tasks given to the participants of the experiment as part of the evaluation process, discussed in Section 9, to test MultiProLan and Process Modeling Tool (see Appendix A.1). We also present the solution to these tasks in the form of MultiProLan process models, that the participants were meant to create (see Appendix A.2). After the participants finished the tasks and tested the possibilities of MultiProLan and Process Modeling Tool, they were asked to complete the questionnaire (see Appendix A.3).

### Appendix A.1. Experiment Tasks

The following text represents the document with the tasks that were given to the experiment participants to test MultiProLan and Process Modeling Tool. These tasks represent a simplified version of the wooden box assembly production process models, presented in Section 8.1. These simplified production process models are similar to the ones presented in our previously published paper [11].

#### MultiProLan Tasks

The tasks on top of which the MultiProLan tool evaluation is performed are given in this document.

You are a process engineer working in a carpentry factory and you are given a task to model a production process to assemble a wooden box, presented in Figure A.1. The box frame is composed



**Figure A.1.** The wooden box whose production process is to be modeled.

of four wooden planks – four sides, that have wooden pins and holes for the assembly. The back side wooden panel needs to be hammered into the assembled frame to create the box. Afterward, an inspection of the box needs to be performed to check whether there is any damage or defect. If damage is detected, the box should be discarded, otherwise, it should be put in storage.

There are three tasks that need to be completed: (i) create a Master-Level (MasL) model; (ii) specify production errors on the previously created MasL model; and (iii) create a part of a Detail-Level (DetL) model.

## I Create a Master-Level Model

First, the MasL model of the wooden box production process needs to be created by using the MultiProLan tool. Further in this section, a textual description of the task is provided:

- The wooden box production process model has arbitrary values for ID, name, and version.
- The wooden box production process starts with the assembly of a left-bottom frame side and a right-upper frame side:
  - To assemble the left-bottom frame side, the following items are needed:
    - The left plank (width 0.5 m, length 0.2 m) retrieved from storage.
    - The bottom plank (width 1 m, length 0.2 m) retrieved from storage.
    - The *Assemble* capability with parameters: n: 2 (a number of wooden pins) and r: 0.07 m (space between pins).
    - The output should be the assembled left-bottom frame side (width 1 m, length 0.5 m, thickness 0.2 m).
  - The assembly of the right-upper side is equivalent to the previously described assembly of the left-bottom side, except that right and upper planks are used. The assembly of the left-bottom side and the right-upper side can be done independently.
- After the left-bottom side and the right-upper side are assembled individually, they need to be assembled to create a frame. To assemble the frame, the following items are needed:
  - The left-bottom and right-upper sides that are assembled in the previous process steps.
  - The *Assemble* capability with parameters: n: 4 (a number of wooden pins) and r: 0.07 m (space between pins).
  - The assembled frame (width 1 m, length 0.5 m, thickness 0.2 m).
- Afterwards, a back side needs to be hammered into the frame, so that the box is created. For such an activity, collaboration is needed:
  - One activity in the collaboration is to hold the frame, and the following items are needed:
    - The frame that is assembled in the previous process step.
    - The *Hold* capability.
    - The assembled box (width 1 m, length 0.5 m, thickness 0.2 m).
  - Another activity in the collaboration is to hammer the back side into the frame, and the following items are needed:
    - The back side (width 1 m, length 0.5 m) retrieved from storage.
    - The *Hammer* capability with a parameter: n: 8 (a number of nails).
    - The back side will be a part of the assembled box created in the first process step of the collaboration.
  - Hammering of the back side into the frame must not start before the frame is being held. Also, holding the frame must not end before the hammering of the back side is finished.
- After the box is assembled, an inspection of the box is needed to check whether there is any damage or defect on the box that occurred during production. To inspect the box, the following items are needed:
  - The box that is assembled in the previous process step.
  - The *Inspection* capability with a parameter: type: visual.
  - The inspected box (width 1 m, length 0.5 m, thickness 0.2 m).

- After the box is inspected, a decision needs to be made whether the box is to be discarded or stored:
  - If there is no damage or defect on the box, it should be stored in storage. To store the inspected box represents a separate process.
    - Note: it is possible to reference a process from a sub-process, but to reduce the volume of the task, this is not required. Optionally, a new process model can be created, with ID, name, and version set, and referenced from a sub-process.
  - If there is any damage or defect on the box, it should be discarded. To discard the inspected box represents a separate process.
- After the decision is made, the wooden box production process is finished.

## II Specify Production Errors at a Master-Level Model

To add production errors in the MasL process model created in the previous task, the error handling modeling layer needs to be turned on. To reduce the volume of the task, only a few production errors are going to be modeled.

The following production errors need to be modeled:

- During the assembly of the frame, it is possible that the left-bottom and right-upper sides are not fitting well. Thus, it is needed to disassemble the frame into the sides, and then try again to assemble the frame.
- If an unidentified error occurs during the assembly of the frame, it is necessary to remove the frame and stop the process.
- When the frame is being held, it is possible for the frame to crack. If this happens, it is necessary to reference another process for discarding and recycling a product that is defined by another process engineer.
  - Note: it is possible to reference a process from an error, but to reduce the volume of the task, this is not required. Optionally, a new process model can be created, with ID, name, and version set, and referenced from an error.
- If an unidentified error occurs during the hammering of the back side, it is necessary to call the process for discarding and recycling a product.

## III Create a Part of a Detail-Level Model

DetL models contain the same modeling concepts as there are in MasL models, but there are also some additional modeling concepts. DetL models are much more complex than MasL models as they have more details included in the models, i.e., details that are specific to a production system in which the models are to be executed. As DetL models are much larger than MasL models, only a part of the wooden box production process will be modeled. The rest of the process is modeled in a similar manner.

An assembly of the left-bottom side and right-upper side of the previously described wooden box production process is going to be modeled.

*Mary Smith* is assigned to assemble the left and bottom sides. To achieve this, it is necessary to:

- Move to the *Shelf* with ID 123. The following item is needed:
  - The *Move* capability with a parameter: location: shelf 123.
- Pick the left side and the bottom side in any order. The following items are needed:
  - The left side (width 0.5 m, length 0.2 m) retrieved from the *Shelf 123* storage.
  - The bottom side (width 1 m, length 0.2 m) retrieved from the *Shelf 123* storage.
  - The *Pick* capability for both process steps.
- Move to *Assembly Table 456*. The following item is needed:
  - The *Move* capability with a parameter: location: assembly table 456.

## 200 Evaluation Experiment Tasks and Questionnaire

- Assemble the left-bottom side. The following items are needed:
  - The left side and the bottom side picked in the previous process steps.
  - The *Assemble* capability with parameters:  $n: 2$  (a number of wooden pins) and  $r: 0.07$  m (space between pins).
  - The assembled left-bottom side (width 1 m, length 0.5 m, thickness 0.2 m).

*Robot 789* is assigned to assemble the right and upper sides. The assembly of the right-upper side is similar to the assembly of the left-bottom side, except that additional configuration process steps are needed. The robot must determine its position after any movement, as it is not equipped with a machine vision module. To configure a robot, only the *Determine* capability is needed with a parameter: type: position.

## Appendix A.2. Experiment Solution

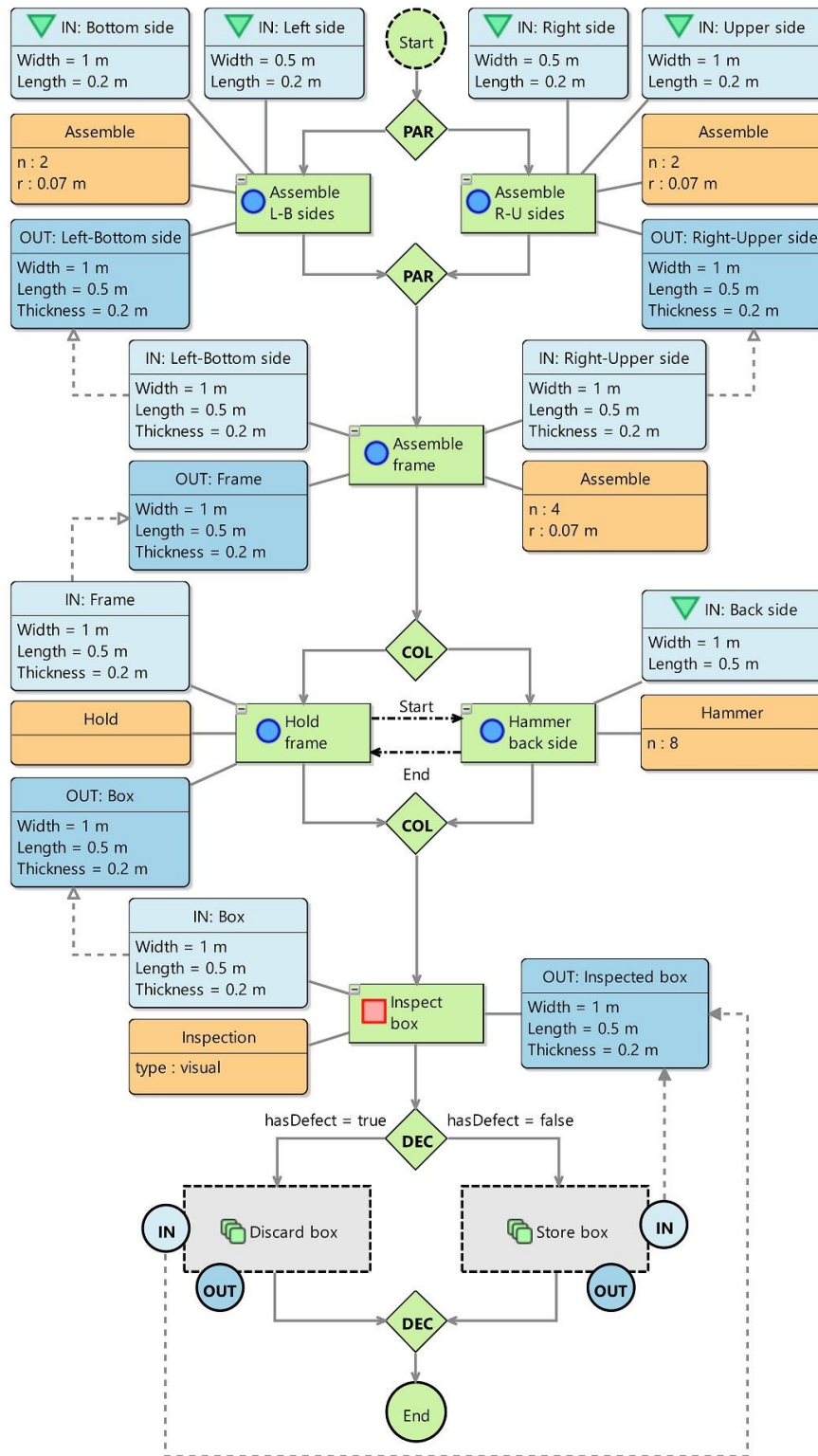


Figure A.2. The solution of the first experiment task.

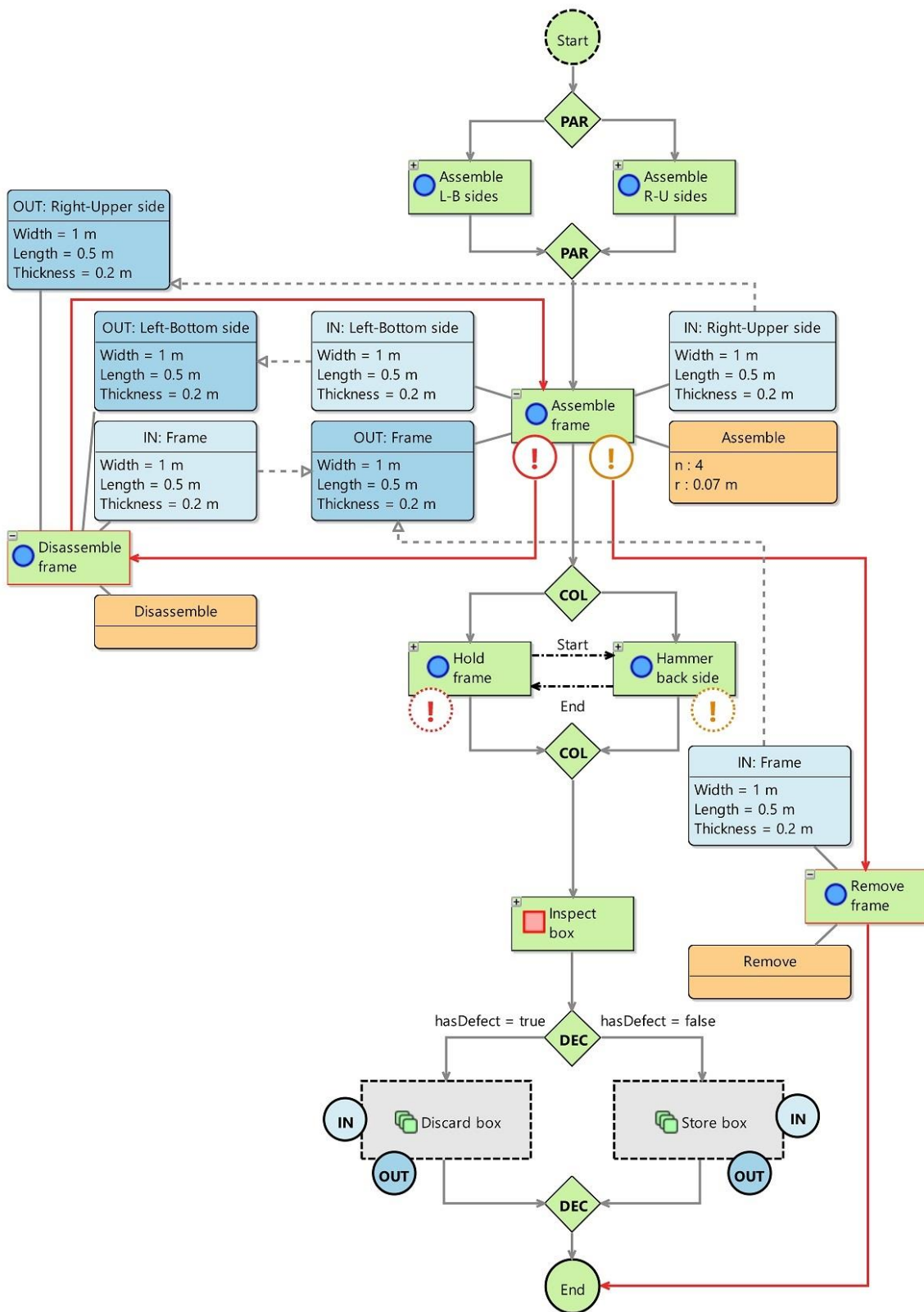


Figure A.3. The solution of the second experiment task.



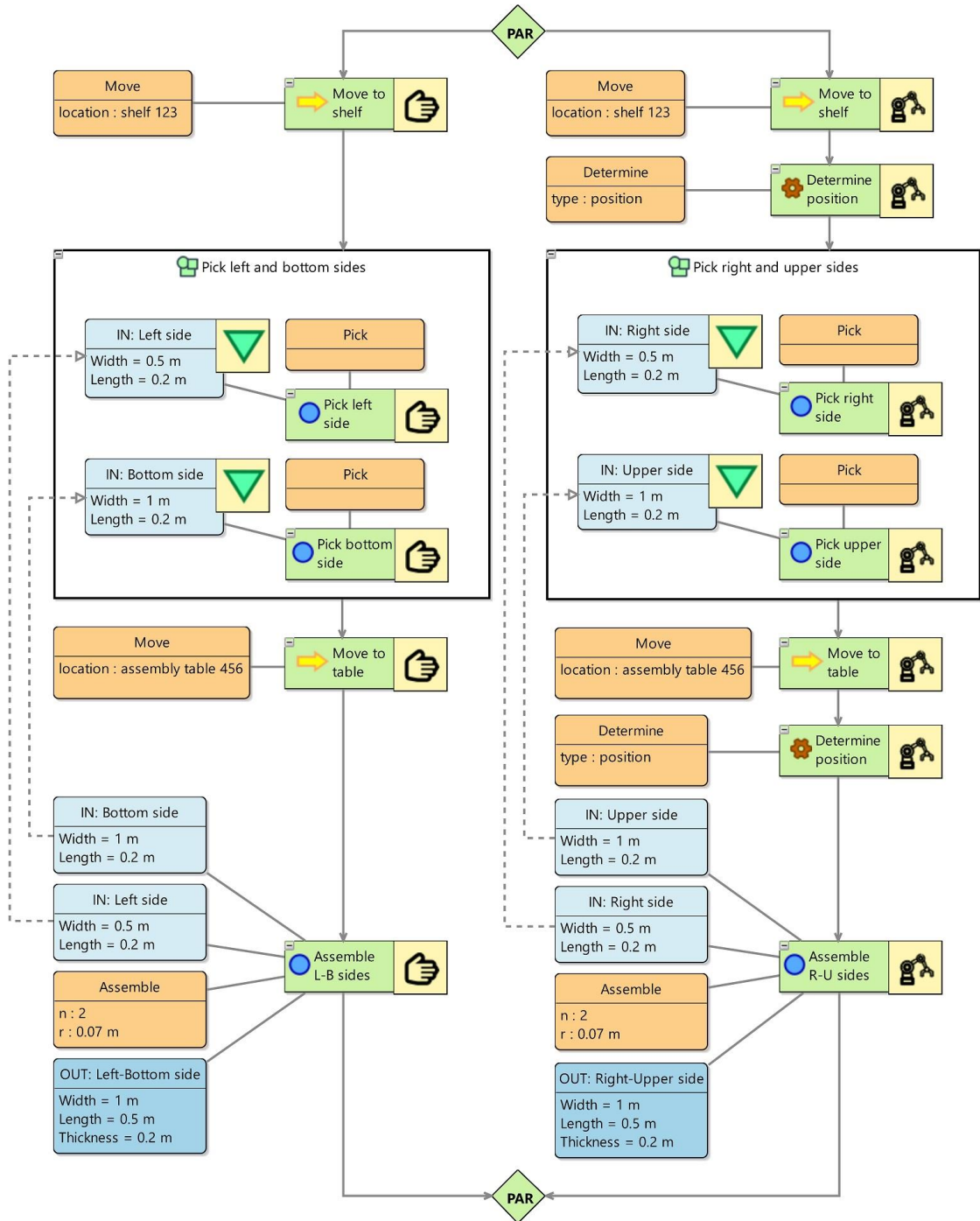


Figure A.4. The solution of the third experiment task.

## Appendix A.3. Experiment Questionnaire

<b>Name and surname</b>	_____
<b>E-mail</b>	_____
<b>Affiliation</b>	Process engineer, Quality engineer, Software developer, Researcher, Ph.D. Student, M.Sc. Student, Other: _____

### Section 1: Previous experience

Experience					
Question	Inexperienced	Relatively inexperienced	Medium experienced	Relatively experienced	Experienced
How would you describe your previous experience in designing business processes?	1	2	3	4	5
Which business process modeling languages or methods have you used?	_____				
How would you describe your previous experience in designing production processes?	1	2	3	4	5
Which production process modeling languages or methods have you used?	_____				
How would you describe your previous experience with Computer Aided Software Engineering (CASE) tools for modeling?	1	2	3	4	5
Which CASE tools for modeling have you used?	_____				

### Section 2: MultiProLan quality characteristics

Functional suitability					
Question	Very low	Low	Medium	High	Very high
How would you describe the scope of production process domain concepts and scenarios that can be expressed in MultiProLan?	1	2	3	4	5
How would you describe MultiProLan's level of suitability for the production process specification?	1	2	3	4	5
Usability					
Question	Strongly disagree	Disagree	No opinion	Agree	Strongly agree
MultiProLan language elements are understandable.	1	2	3	4	5
The concepts and symbols of MultiProLan are learnable and rememberable.	1	2	3	4	5
MultiProLan has capability to help users achieve their tasks in an acceptable number of steps.	1	2	3	4	5
MultiProLan is appropriate for your needs.	1	2	3	4	5
MultiProLan Eclipse environment has elements that facilitate to operate and control the language.	1	2	3	4	5
MultiProLan has graphical symbols that are good looking/attractive.	1	2	3	4	5
By separating MasL and DetL models, users can model production processes easier.	1	2	3	4	5
By creating different modeling layers, models become more readable.	1	2	3	4	5

Reliability					
Question	Strongly disagree	Disagree	No opinion	Agree	Strongly agree
MultiProLan protects users against making errors.	1	2	3	4	5
MultiProLan has a functional model validator.	1	2	3	4	5
Expressiveness					
Question	Strongly disagree	Disagree	No opinion	Agree	Strongly agree
A problem-solving strategy can be mapped into a specification easily.	1	2	3	4	5
MultiProLan is at the right abstraction level such that it is not more complex or detailed than necessary.	1	2	3	4	5
MultiProLan provides one and only one good way to express every concept of interest.	1	2	3	4	5
Productivity					
Question	Long	Relatively long	Medium	Relatively short	Short
How would you describe the specification time of a production process model with MultiProLan?	1	2	3	4	5

Section 3: Free comments

<b>Additional comments</b>	_____
	_____
	_____
	_____
	_____



## План третмана података

Назив пројекта/истраживања
Пристап спецификацији и генерисању производних процеса заснован на инжењерству вођеном моделима (енгл. <i>A Model-Driven Approach to the Production Process Specification and Generation</i> )
Назив институције/институција у оквиру којих се спроводи истраживање
а) Универзитет у Новом Саду, Факултет техничких наука б) <i>KEBA Group AG</i> , Линц, Република Аустрија в)
Назив програма у оквиру ког се реализује истраживање
Истраживање је реализовано у оквиру израде докторске дисертације на студијском програму Рачунарство и аутоматика. Такође, истраживање је подржано од стране следећих пројеката: <ul style="list-style-type: none"><li>• "Интелигентни системи за развој софтверских производа и подршку пословања засновани на моделима ", ИИИ-44010, Министарство просвете, науке и технолошког развоја Републике Србије.</li><li>• "Иновативна научна и уметничка испитивања из домена делатности ФТН-а", 451-03-68/2020-14/200156, 451-03-68/2021-14/200156, 451-03-68/2022-14/200156, 451-03-47/2023-01/200156, Министарство науке, технолошког развоја и иновација Републике Србије.</li><li>• "Дигитална фабрика", Индустијски истраживачко-развојни пројекат, <i>KEBA Group AG</i>, Линц, Република Аустрија.</li></ul>
1. Опис података
1.1. Врста студије  <i>Укратко описати тип студије у оквиру које се подаци прикупљају</i>  У овој докторској дисертацији представљена је анализа оцена различитих учесника о наменском језику и софтверском алату за моделовање производних процеса.
1.2. Врсте података  а) <b><u>квантитативни</u></b> б) <b><u>квалитативни</u></b>
1.3. Начин прикупљања података  а) <b><u>анкете, упитници, тестови</u></b> б) клиничке процене, медицински записи, електронски здравствени записи в) генотипови: навести врсту _____ г) административни подаци: навести врсту _____ д) узорци ткива: навести врсту _____ ђ) снимци, фотографије: навести врсту _____ е) текст, навести врсту _____

- ж) мапа, навести врсту \_\_\_\_\_  
з) остало: описати \_\_\_\_\_

#### 1.4. Формат података, употребљене скале, количина података

##### 1.4.1. Употребљени софтвер и формат датотеке:

- а) **Excel фајл, датотека** .xlsx  
б) SPSS фајл, датотека \_\_\_\_\_  
в) PDF фајл, датотека \_\_\_\_\_  
г) Текст фајл, датотека \_\_\_\_\_  
д) JPG фајл, датотека \_\_\_\_\_  
ђ) **Остало, датотека** *Google Forms, .csv*

##### 1.4.2. Број записа (код квантитативних података)

- а) **број варијабли** 19  
б) **број мерења (испитаника, процена, снимака и сл.)** 25

##### 1.4.3. Поновљена мерења

- а) да  
б) **не**

Уколико је одговор да, одговорити на следећа питања:

- а) временски размак између поновљених мера је \_\_\_\_\_  
б) варијабле које се више пута мере односе се на \_\_\_\_\_  
в) нове верзије фајлова који садрже поновљена мерења су именоване као \_\_\_\_\_

Напомене: \_\_\_\_\_

*Да ли формати и софтвер омогућавају дељење и дугорочну валидност података?*

- а) **Да**  
б) **Не**

*Ако је одговор не, образложити* \_\_\_\_\_  
\_\_\_\_\_

## 2. Прикупљање података

### 2.1. Методологија за прикупљање/генерисање података

#### 2.1.1. У оквиру ког истраживачког нацрта су подаци прикупљени?

- а) **експеримент** попуњавање упитника од стране учесника оцене наменског језика и софтверског алата за моделовање производних процеса  
б) **корелационо истраживање** рачунање Спирмановог коефицијента корелације  
в) анализа текста, навести тип \_\_\_\_\_  
г) **остало** дескриптивна анализа коментара из попуњених упитника

#### 2.1.2. Навести врсте мерних инструмената или стандарде података специфичних за одређену научну дисциплину (ако постоје).

Креиран је упитник уз помоћ платформе *Google Forms*, који су учесници оцене наменског језика и алата за моделовање производних процеса попуњавали електронским путем.

### 2.2. Квалитет података и стандарди

### 2.2.1. Третман недостајућих података

а) Да ли матрица садржи недостајуће податке? Да **Не**

Ако је одговор да, одговорити на следећа питања:

- а) Колики је број недостајућих података? \_\_\_\_\_  
б) Да ли се кориснику матрице препоручује замена недостајућих података? Да Не  
в) Ако је одговор да, навести сугестије за третман замене недостајућих података
- 

### 2.2.2. На који начин је контролисан квалитет података? Описати

Већина питања (19 од 23) садржала је понуђене одговоре у виду петостепене Ликертове скале. Платформа *Google Forms* онемогућава унос било каквих других података осим одговора датих на петостепеној Ликертовој скали. Преостала питања (4 од 23) садржала су слободну форму за унос текста, чији су одговори улазили искључиво у дескриптивну анализу.

### 2.2.3. На који начин је извршена контрола уноса података у матрицу?

Платформа *Google Forms* на аутоматизован начин трансформише одговоре прикупљене од учесника у *.csv (comma-separated values)* датотеку.

## 3. Третман података и пратећа документација

### 3.1. Третман и чување података

3.1.1. Подаци ће бити депоновани у Репозиторијум докторских дисертација Универзитета у Новом Саду.

3.1.2. URL адреса: <https://www.cris.uns.ac.rs/dmi.jsf>

3.1.3. DOI

---

3.1.4. Да ли ће подаци бити у отвореном приступу?

- а) **Да**  
б) Да, али после ембарга који ће трајати до \_\_\_\_\_  
в) Не

Ако је одговор не, навести разлог \_\_\_\_\_

3.1.5. Подаци неће бити депоновани у репозиторијум, али ће бити чувани.  
Образложење

---

---

### 3.2. Метаподаци и документација података

3.2.1. Који стандард за метаподатке ће бити примењен? Стандард који примењује Репозиторијум Универзитета у Новом Саду

3.2.2. Навести метаподатке на основу којих су подаци депоновани у репозиторијум.

Марко Вјештица, Приступ спецификацији и генерисању производних процеса заснован на инжењерству вођеном моделима

*Ако је потребно, навести методе које се користе за преузимање података, аналитичке и процедуралне информације, њихово кодирање, детаљне описе варијабли, записа итд.*

---

---

---

---

3.3. Стратегија и стандарди за чување података

3.3.1. До ког периода ће подаци бити чувани у репозиторијуму? **Неограничено**

3.3.2. Да ли ће подаци бити депоновани под шифром? Да **Не**

3.3.3. Да ли ће шифра бити доступна одређеном кругу истраживача? Да **Не**

3.3.4. Да ли се подаци морају уклонити из отвореног приступа после извесног времена?

Да **Не**

Образложити

---

---

#### 4. Безбедност података и заштита поверљивих информација

Овај одељак МОРА бити попуњен ако ваши подаци укључују личне податке који се односе на учеснике у истраживању. За друга истраживања треба такође размотрити заштиту и сигурност података.

4.1. Формални стандарди за сигурност информација/података

Истраживачи који спроводе испитивања с људима морају да се придржавају Закона о заштити података о личности

([https://www.paragraf.rs/propisi/zakon\\_o\\_zastiti\\_podataka\\_o\\_licnosti.html](https://www.paragraf.rs/propisi/zakon_o_zastiti_podataka_o_licnosti.html)) и одговарајућег институционалног кодекса о академском интегритету.

4.1.1. Да ли је истраживање одобрено од стране етичке комисије? Да **Не**

Ако је одговор Да, навести датум и назив етичке комисије која је одобрила истраживање

---

4.1.2. Да ли подаци укључују личне податке учесника у истраживању? Да **Не**

Ако је одговор да, наведите на који начин сте осигурали поверљивост и сигурност информација везаних за испитанике:

а) Подаци нису у отвореном приступу

б) **Подаци су анонимизирани**

в) Остало, навести шта

---

---

#### 5. Доступност података

5.1. Подаци ће бити

а) **јавно доступни**

---



б) доступни само уском кругу истраживача у одређеној научној области  
в) затворени

Ако су подаци доступни само уском кругу истраживача, навести под којим условима могу да их користе:

---

---

Ако су подаци доступни само уском кругу истраживача, навести на који начин могу приступити подацима:

---

---

5.2. Навести лиценцу под којом ће прикупљени подаци бити архивирани.

Ауторство – некомерцијално – без прераде

## 6. Улоге и одговорност

6.1. Навести име и презиме и мејл адресу власника (аутора) података

Марко Вјештица, marko.vjestica@uns.ac.rs

6.2. Навести име и презиме и мејл адресу особе која одржава матрицу с подацима

Марко Вјештица, marko.vjestica@uns.ac.rs

6.3. Навести име и презиме и мејл адресу особе која омогућује приступ подацима другим истраживачима

Марко Вјештица, marko.vjestica@uns.ac.rs