

Data Exchange for the Physics-based Simulation of Material Handling Systems in the Digital Factory

Dissertation

zur Erlangung des akademischen Grades

**Doktoringenieur
(Dr.-Ing.)**

von M.Sc. Ender Yemenicioglu

geb. am 21.07.1982 in Ankara, Türkei

genehmigt durch die Fakultät Maschinenbau der
Otto-von-Guericke-Universität Magdeburg

Gutachter:

apl. Prof. Dr.-Ing. habil. Arndt Lüder

Prof. Dr.-Ing. habil. Björn Hein

Promotionskolloquium am 02.03.2016

Acknowledgements

This doctoral thesis is a result of my occupation in tarakos GmbH and my activity as a part of ITEA 2 project AVANTI, which took place from 2014 to 2016. I would like to express my special appreciation to my principal, Dipl.-Ing. Herbert Beesten, for his support from the very beginning. I am grateful to all AVANTI project partners for the best cooperation.

I wish to thank my advisor Professor Dr.-Ing. habil. Arndt Lüder for his caressing guidance, and the motivation he has given during the thesis work. I could not manage to get this far if he has not “tricked” me to engage in the data exchange subject when I was a student looking for a master thesis subject for six years. He was always there with his experience and wisdom when I needed some orientation.

I also thank Professor Dr.-Ing. habil. Björn Hein for his appraisal, careful reporting and corrections of the thesis.

I want to thank all my colleagues but especially B.Sc. Christian Göbeler, Dipl.-Ing. Klaus Hanisch, Dipl.-Ing. Michael Dietz and Dipl.-Ing. Henry Cermann for their great help and assistance. I also want to express my appreciation to all colleagues in AutomationML association as they have provided vital technical help.

Never to forget, my wife Canan Yemenicioglu must have to endure my absence as I often had to work for the thesis in my free time. I am grateful for her patience, understanding, and love. My mother and father have always motivated me to do my best. Thank you to all my beloved ones for believing in me.

Schriftliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die Hilfe eines kommerziellen Promotionsberaters habe ich nicht in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen. Verwendete fremde und eigene Quellen sind als solche kenntlich gemacht.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann.

Ich erkläre mich damit einverstanden, dass die Dissertation ggf. mit Mitteln der elektronischen Datenverarbeitung auf Plagiate überprüft werden kann.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 16.03.2017

Ender Yemenicioğlu

Abstract

Cost constraints, higher quality requirements, increasing time pressure together with decreasing product life cycles and demand for improved agility to support a variety of products are the challenges of the current engineering design processes. Digital factory methods offer significant quality and efficiency improvements already in the early phases of the plant design. A variety of information types like 3D geometry, physical and kinematic behavior models and control approaches are necessary for the application of digital factory. This diversity of information is an indication that a single engineering tool cannot be sufficient for all the tasks in the mechatronic engineering design process. The data exchange between different engineering tools, which are specialized in their particular fields, is a critical issue in this sense.

This work explains an implementation of a data exchange functionality for a factory layout planning tool. The focus lies on the material handling components and processes in the sense of mechatronics concept. The example application is for the virtual commissioning. AutomationML (AML) data format was selected for this purpose as it can be used to achieve a consistent and non-redundant data exchange for mechatronic production components like production machines or conveyor systems as well as logical planning data such as material handling and manufacturing process definitions.

As the first step of the preparation for the data exchange, the proprietary format is converted into the AML format with topology, geometry, kinematics, and physics aspects for the modeling of production, logistics, and material handling systems. The necessary components and their properties, as well as process flow and parameters, can be described with the help of semantic roles and classes in the standard format.

The software components to execute the data exchange steps, like exporting and importing, are realized parallel to the modeling. The implemented solutions enhance the existing layout planning tool *taraVRBuilder* and process visualization tool *taraVRControl* with AML data exchange functionalities.

At the end, applications of the software components are provided to demonstrate how data exchange problem is solved in a material handling use-case and a virtual commissioning use-case. An exemplary scene for a material handling system and three demonstrators for the virtual commissioning, which are realized together with AVANTI project partners, are introduced.

The result of this doctoral research is applied in tarakos software toolkit.

Keywords: Data exchange, Layout-planning, Material handling, AutomationML, Mechatronics, Virtual commissioning, 3D Visualization, Software engineering

Kurzfassung

Zunehmender Zeit- und Kostendruck, höhere Qualitätsanforderungen zusammen mit abnehmenden Produktlebenszyklen und die Nachfrage nach erhöhter Agilität zur Unterstützung der Vielzahl von Produkten sind die Herausforderungen der gegenwärtigen ingenieurwissenschaftlichen Planungsprozesse. Methoden der digitalen Fabrik bieten erhebliche Qualitäts- und Effizienzverbesserungen bereits in den frühen Phasen der Anlagenplanung. Eine Vielfalt von Informationstypen wie 3D-Geometrie, kinematische und physikalische Verhaltensmodelle und Steuerungsansätze sind notwendig für die Anwendung der digitalen Fabrik. Diese Informationsvielfalt ist ein Hinweis darauf, dass ein einziges Engineering-Tool für alle Aufgaben im mechatronischen Engineering-Design nicht ausreichen kann. Der Datenaustausch zwischen unterschiedlichen Engineering-Tools, die auf bestimmten Gebieten spezialisiert sind, ist ein kritisches Thema in diesem Sinne.

Diese Arbeit präsentiert die Implementierung einer Datenaustausch-Funktionalität für ein Anlagenlayout-Planungstool. Der Fokus liegt auf den Materialflusstechnik-Komponenten und Prozessen im Sinne des Mechatronik-Konzepts. Die Beispielanwendung bildet die virtuelle Inbetriebnahme. Das AutomationML(AML)-Datenformat wurde für diesen Zweck ausgewählt. Ein konsistenter und nicht-redundanter Datenaustausch für mechatronische Produktionskomponenten wie Produktionsmaschinen oder Förderanlagen und logische Planungsdaten wie Definitionen von Materialfluss- und Produktionsprozesse kann damit erreicht werden.

Als erster Schritt der Vorbereitung auf den Datenaustausch wird das proprietäre Format in das AML-Format mit Topologie-, Geometrie-, Kinematik- und Physik-Aspekten für die Modellierung von Produktions-, Logistik- und Materialflusssystemen umgesetzt. Die erforderlichen Komponenten und ihre Eigenschaften sowie Prozess-Abläufe und Parameter können mit Hilfe von semantischen Rollen und Klassen im standardisierten Format beschrieben werden.

Die Softwarekomponenten, die Datenaustauschschritte wie Exportieren und Importieren ausführen, werden parallel zur Modellierung umgesetzt. Realisierte Lösungen erweitern das bestehende Layout-Planung-Tool *taraVRBuilder* und das Prozess-Visualisierung-Tool *taraVRControl* um die Datenaustausch-Funktionalitäten für AML.

Am Ende werden exemplarische Anwendungen der Software-Komponenten vorgestellt um zu zeigen, wie das Datenaustausch-Problem für die Anwendungsfälle wie Materialflusstechnik und virtuelle Inbetriebnahme gelöst ist. Eine Beispiel-Szene für ein Materialflusssystem und drei Demonstratoren für die virtuelle Inbetriebnahme, die gemeinsam mit AVANTI - Projektpartnern realisiert sind, werden präsentiert.

Das Ergebnis dieser Forschung wird im tarakos Software Toolkit angewendet.

Schlüsselwörter: Datenaustausch, Layout-Planung, Materialflusstechnik, AutomationML, Mechatronik, Virtuelle Inbetriebnahme, Software-Engineering

Özet

Maliyet kısıtlamaları, yükselen kalite beklentileri, ürünlerin kullanım ömrünün azalmasıyla paralel gelişen kısa zaman içinde teslimat yapabilme ihtiyacı, bunların yanı sıra ürün çeşitliliğinin giderek artması ile çevik (agile) üretim yöntemlerinin yaygınlaşması günümüz mühendislik tasarım süreçlerinde dikkate alınması gereken önemli meselelerdir. Dijital fabrika yöntemleri ise üretim sistemleri tasarımının erken evrelerinde dahi ciddi kalite ve verim artışları sağlamaktadır. 3 boyutlu (3B) geometri, fiziksel ve kinematik davranış modelleri, kontrol yaklaşımları gibi çeşitli bilgi tipleri dijital fabrika uygulamaları için gereken bilgi türleridir. Gereken bilginin çeşitliliği ise mekatronik tasarım süreçlerinin tek bir mühendislik yazılımı ile çözülemeyecek kadar geniş olduğunun bir göstergesidir. Bu sebepten, hepsi kendi alanında uzmanlaşmış çeşitli mühendislik yazılımları arasındaki veri aktarımı kritik bir sorundur.

Bu çalışma bir fabrika yerleşim planlaması yazılımı için yeni veri aktarım fonksiyonlarının yaratılması üzerinedir. Odak noktasında mekatronik kavramıyla bağlantılı değerlendirilen malzeme aktarımı ve süreçleri bulunmaktadır. Örnek uygulama sanal işletmeye alma (virtual commissioning) için yapılmıştır. Veri aktarımı için AutomationML (AML) formatı kullanılmıştır, çünkü bu format üretim makineleri, konveyör sistemleri gibi mekatronik üretim elemanlarının yanı sıra malzeme aktarım ve üretim süreçleri gibi mantıksal planlama verilerinin de tutarlı ve efektif bir şekilde taşınmasını sağlamaktadır.

Veri aktarımı uygulamasının ilk aşamasında programda kullanılmakta olan veri formatı topoloji, geometri, kinematik bilgileri ve üretim, lojistik ve malzeme aktarımı sistemleri için fiziksel verileri de içeren bir AML formatına dönüştürülmüştür. İhtiyaç duyulan sistem elemanları ve özelliklerinin yanı sıra süreç akışı ve parametreleri de semantik roller ve sınıflandırmalar üzerinden bu standart formatın içinde modellenmiştir.

Modellemeye paralel olarak veri aktarımını yani içe aktarma ve dışarı yazma adımlarını yerine getirecek yazılım elemanları geliştirilmiştir. Oluşturulan çözümler *taraVRBuilder* isimli fabrika yerleşim planlaması yazılımı ve *taraVRControl* isimli süreç görüntüleme yazılımını AML formatını destekleyecek şekilde geliştirilmesinde kullanılmıştır.

Sonuç olarak, bir malzeme aktarımı sistemlerinde ve sanal işletmeye alma kullanım örneğinde geliştirilen yazılım elemanlarının uygulaması gösterilmiştir. Malzeme aktarım sistemleri için örnek bir sahne yaratılmış, ayrıca *AVANTI* proje ortakları ile beraber oluşturulan üç adet sanal işletmeye alma örneği tanıtılmıştır.

Bu çalışmanın sonucu tarakos yazılım kitinde ticari bir ürün olarak uygulamaya sokulmuştur.

Anahtar kelimeler: Veri aktarımı, Yerleşim planlaması, Malzeme aktarımı, AutomationML, Mekatronik, Sanal işletmeye alma, 3 boyutlu görüntüleme, Yazılım mühendisliği

Contents

| | |
|---|--------------|
| Acknowledgements | ii |
| Schriftliche Erklärung | iii |
| Abstract | iv |
| Kurzfassung | v |
| Özet | vii |
| Contents | viii |
| List of Figures | xi |
| List of Tables | xv |
| List of Abbreviations | xvi |
| Glossary | xviii |
| 1. Introduction | 1 |
| 1.1. Problem definition and motivation | 1 |
| 1.2. Aim of the work | 4 |
| 1.3. Structure of the work..... | 4 |
| 2. Fundamentals | 7 |
| 2.1. Basics of automation engineering and mechatronics..... | 7 |
| 2.1.1. Mechatronic system..... | 7 |
| 2.1.2. The mechatronic design object..... | 8 |
| 2.1.3. Mechatronical engineering process | 9 |
| 2.2. Digital factory methods in the mechatronical engineering process | 11 |
| 2.2.1. Product design | 11 |
| 2.2.2. Layout planning | 12 |
| 2.2.3. Functional engineering | 13 |
| 2.2.4. Virtual commissioning..... | 14 |
| 2.2.5. Digital construction site management | 16 |
| 2.3. Mathematical and software related aspects | 17 |
| 2.3.1. Model description | 17 |
| 2.3.2. Physics simulation | 18 |
| 2.3.3. Co-simulation | 19 |
| 2.3.4. Virtual reality..... | 20 |
| 2.3.5. Programming aspects..... | 22 |
| 2.4. Summary | 25 |

| | |
|---|-----------|
| 3. Literature Review | 27 |
| 3.1. Literature about modeling and data exchange in the mechatronical engineering process | 27 |
| 3.2. Literature about physics-based simulation regarding virtual commissioning applications | 28 |
| 3.3. Summary | 29 |
| 4. Requirements Analysis | 31 |
| 4.1. Methodological requirements | 31 |
| 4.2. Technical requirements | 32 |
| 4.3. Summary | 33 |
| 5. Data Exchange Formats | 35 |
| 5.1. Selection of the data exchange formats | 35 |
| 5.1.1. Selection criteria | 35 |
| 5.1.2. Overview of data exchange formats | 36 |
| 5.1.3. Discussion and decision | 39 |
| 5.2. Overview of the selected data exchange formats | 39 |
| 5.2.1. AutomationML | 39 |
| 5.2.2. COLLADA | 46 |
| 5.2.3. Functional Mockup Interface | 58 |
| 5.3. Overview of the native format (X3D/VRML) | 59 |
| 5.3.1. Meta information | 61 |
| 5.3.2. Geometry primitives | 61 |
| 5.3.3. Grouping nodes | 62 |
| 5.3.4. Prototypes | 62 |
| 5.4. Summary | 63 |
| 6. Modeling of Material Handling System Elements with AutomationML | 65 |
| 6.1. Problem description | 65 |
| 6.2. Geometric model of the system components | 66 |
| 6.3. Physical model | 70 |
| 6.4. Topology and component description for material handling technology | 72 |
| 6.4.1. Preparation of a standardized data exchange solution for material handling ... | 73 |
| 6.4.2. Process description of material handling | 76 |
| 6.4.3. An example usage of the concepts | 77 |
| 6.5. Summary | 78 |
| 7. Implementation Concept | 80 |
| 7.1. Introduction to the software components of the digital factory | 80 |
| 7.1.1. Layout planning tool | 80 |
| 7.1.2. Process visualization tool | 80 |
| 7.1.3. Further development | 81 |
| 7.2. Software components for the data exchange | 82 |

- 7.2.1. Selection of technology82
- 7.2.2. Architecture of the implemented components84
- 7.2.3. Development of software libraries.....86
- 7.3. Summary96
- 8. Application98**
 - 8.1. Sample material handling system.....98
 - 8.1.1. Exporting into AutomationML.....98
 - 8.1.2. Assignment to standard components.....102
 - 8.1.3. Geometry conversion104
 - 8.1.4. Importing from AutomationML105
 - 8.2. Application in virtual commissioning.....106
 - 8.2.1. Material handling systems in virtual commissioning toolchain.....106
 - 8.2.2. Demonstrators.....108
 - 8.3. Summary111
- 9. Technical and Economic Evaluation112**
 - 9.1. Technical evaluation112
 - 9.2. Economic evaluation113
- 10. Summary and Results115**
- References.....117**
- Addendum129**
 - Appendix A Material Handling Role Class and Interface Libraries in AutomationML129

List of Figures

| | |
|--|----|
| Figure 1 Loss of information during the engineering design process based on [KIE07] | 2 |
| Figure 2 Structure of the work | 6 |
| Figure 3 Mechatronics: synergistic integration of different disciplines based on [Ren16] | 8 |
| Figure 4 Schematic representation of mechatronical engineering design object (Material handling perspective) | 9 |
| Figure 5 The coarse structure of the mechatronical engineering process | 11 |
| Figure 6 Layout planning of the plant structure based on [Paw14]..... | 13 |
| Figure 7 Virtual commissioning setup for SIL and HIL approaches based on [Spi09] | 16 |
| Figure 8 Placement of the digital construction site management in the process chain for plant construction based on [Dre13] | 17 |
| Figure 9 Master-Slave architecture in co-simulation based on [Blo12]..... | 19 |
| Figure 10 Usage of wearable devices (immersive VR) for layout planning, shown with taraVRBuilder and Oculus Rift VR headset..... | 20 |
| Figure 11 Object oriented aspects of a mechatronic design object for material handling (Exemplary in C# language) | 23 |
| Figure 12 Relative cost to fix due to the development phase. Based on [Tas02] | 25 |
| Figure 13 AML as a combination of the topology, geometry, kinematics, physics, logic information as well as possibly other XML-based information [AML13]...... | 40 |
| Figure 14 Fundamental AML elements based on [DrSc10]..... | 41 |
| Figure 15 Elements of the PPR concept [ScDr09] | 43 |
| Figure 16 An example conveyor object in the plant topology shown in AML-Editor..... | 44 |
| Figure 17 An ideal data exchange with AML based on [Dra13]..... | 45 |
| Figure 18 An example COLLADA document exported from taraVRBuilder software tool..... | 46 |
| Figure 19 BREP model of a hull and tessellated representation based on [Lip09] | 49 |
| Figure 20 Structure of a triangle based geometry in COLLADA..... | 50 |
| Figure 21 A simple product tree example | 51 |

| | |
|--|----|
| Figure 22 Joint definition in COLLADA [Lip09]..... | 53 |
| Figure 23 A simple kinematic model based on [Lip09] | 54 |
| Figure 24 Binding a joint axis to a transformation node | 55 |
| Figure 25 Combining geometry and kinematics in scene..... | 56 |
| Figure 26 An example physics model for two rigid bodies combined with one constraint..... | 57 |
| Figure 27 Combining visual and physics scene | 57 |
| Figure 28 Components of the Functional Mock-up Unit based on [Blo12]..... | 58 |
| Figure 29 Integration of FMI into AML based on [LNY16] | 59 |
| Figure 30 A graphical depiction of main X3D profiles | 60 |
| Figure 31 Header element in X3D for the definition of Meta information | 61 |
| Figure 32 Node syntax for a shape with a box..... | 62 |
| Figure 33 Physics transform prototype declaration and instantiation | 63 |
| Figure 34 Plant simulation process and implementation of AML..... | 66 |
| Figure 35 An example conveyor system for representation of visual, static and dynamic components along with sensors..... | 71 |
| Figure 36 Graph representation of a material handling system..... | 72 |
| Figure 37 Role class library for material handling components | 74 |
| Figure 38 Example system unit classes for material handling components..... | 75 |
| Figure 39 Interface classes for material handling systems..... | 75 |
| Figure 40 Role class library for material handling processes (Part 1) | 76 |
| Figure 41 Role class library for material handling processes (Part 2) | 77 |
| Figure 42 An example usage of the concepts with a linear conveyor and a turntable | 78 |
| Figure 43 Toolchain of tarakos software components..... | 81 |
| Figure 44 Overview of the used technologies and implemented libraries..... | 82 |
| Figure 45 Software components of the AML importing and exporting functionalities..... | 84 |
| Figure 46 Sequence diagrams of the import and export functions | 86 |

| | |
|---|-----|
| Figure 47 An exemplary part of COLLADA POCOs in UML representation | 87 |
| Figure 48 An exemplary part of COLLADA readers in UML..... | 88 |
| Figure 49 An exemplary part of COLLADA writers in UML | 88 |
| Figure 50 Triangulation functionality for geometry primitives in X3D..... | 90 |
| Figure 51 A coarse representation of COLLADA to X3D/VRML converter structure | 91 |
| Figure 52 A coarse representation of X3D/VRML to COLLADA converter structure | 92 |
| Figure 53 A coarse representation of prototypical physics engine implementation | 93 |
| Figure 54 Dependency graph of the COLLADA Physics Converter library extracted from Visual Studio | 93 |
| Figure 55 UML diagram of the COLLADA to X3D/VRML external prototype converter | 94 |
| Figure 56 UML diagram of the AML exporter for straight conveyor example..... | 95 |
| Figure 57 UML diagram of the AML importer library | 96 |
| Figure 58 Sample scene for modeling of material handling components created in taraVRBuilder | 99 |
| Figure 59 The screenshot of the exported AML document | 100 |
| Figure 60 Example implementation for Product-Process-Resource (PPR) concept..... | 101 |
| Figure 61 PPR connection between source, good and connection point | 102 |
| Figure 62 System unit classes of the example scene | 103 |
| Figure 63 The intermediate layer places the nodes into the origin and transmits the reference from AML element into the original COLLADA file. | 104 |
| Figure 64 Import result of the example scene in Autodesk 3dsMax with OpenCollada plugin | 105 |
| Figure 65 Example scene is imported to taraVRControl software tool with own importer... .. | 106 |
| Figure 66 Positioning of tarakos software tools in virtual commissioning toolchain..... | 107 |
| Figure 67 Example scene in RF::SGView to demonstrate possible connection to the virtual commissioning toolchain..... | 108 |
| Figure 68 Demonstrator pneumatic station in taraVRControl..... | 109 |
| Figure 69 Material handling system use-case in taraVRControl..... | 110 |

Figure 70 Demonstrator marriage station for car assembly in taraVRControl 110

List of Tables

| | |
|--|----|
| Table 1 Relevant subjects and authors mentioned in the literature review | 30 |
| Table 2 Comparison of data exchange formats | 38 |
| Table 3 Joint types and definitions based on [Lip09] | 52 |
| Table 4 Comparison of COLLADA and X3D geometry components | 68 |

List of Abbreviations

2D: Two-dimensional

3D: Three-dimensional

AML: Automation Markup Language

BREP: Boundary Representation

CAD: Computer-aided Design

CAE: Computer-aided Engineering

CAEX: Computer Aided Engineering Exchange

CAM: Computer-aided Manufacturing

CNC: Computer Numerical Control

COLLADA: Collaborative Design Activity

EP: Engineering Process

FMI: Functional Mockup Interface

FMU: Functional Mockup Unit

IEC: International Electrotechnical Commission

ISO: International Organization for Standardization

IT: Information technology

MEP: Mechatronical Engineering Process

NURBS: Non-uniform Rational Basis Spline

OEM: Original Equipment Manufacturer

OPC: Open Platform Communications

PC: Personal Computer

PLC: Programmable Logic Controller

UML: Unified Modeling Language

URL: Uniform Resource Locator

X3D: Extensible 3D Graphics

XML: Extensible Markup Language

XSD: XML Schema Definition

VDI: Verein Deutscher Ingenieure (English: Association of German Engineers)

VRML: Virtual Reality Modeling Language

VR: Virtual reality

Glossary

AutomationML® (Automation Markup Language)

XML-based data exchange format for plant engineering data

CAEX (Computer Aided Engineering Exchange)

CAEX is a neutral data format, which is used for the storage of hierarchical object information, e.g. the topology of a manufacturing system.

COLLADA (COLLABorative Design Activity)

COLLADA is an interchange file format for interactive 3D applications.

Co-Simulation

Co-Simulation is a general approach to simulate coupled technical systems in a master-slave concept. The slaves simulate sub-problems. The master is responsible for coordinating the simulation and transferring data.

Data Exchange

Data exchange is the process of taking data structured under a source database system and actually transforming it into data structured under a target database system so that the target data is an accurate representation of the source data.

Engineering Tool

Engineering tool is a computer software system, which is used to fulfill or aid the engineering tasks.

MathML (Mathematical Markup Language)

MathML is an application of XML for describing mathematical notations and capturing both its structure and content.

Mechatrical Engineering Process

The mechatrical engineering process includes the product development, but mainly describes the development of the manufacturing plant with a combination of mechanical, electronic and software engineering.

Mechatrical Engineering Object

The term of the mechatrical engineering object is referred to all the components of a system in the mechatrical engineering process, providing the technical control for the automation.

Mechatronic System

Integrated mechanical/electronic (mechatronic) systems are an eligible merging of mechanics, electronics and control/information processing to form the desired system behavior.

Mechatronic Unit

The mechatronic unit is the combination of energetically, informational and material flows, which are connected via sensors and actuators to an information-processing unit, usually establishing a hierarchical structure.

Physics engine

A physics engine is a software component that provides an approximate simulation of certain physical systems, like rigid body dynamics (including collision detection), soft body dynamics, and fluid dynamics.

UML (Unified Modeling Language)

UML is a graphical modeling language mostly used in the field of software engineering, which is intended to provide a standard method to visualize the design of a system.

X3D (Extensible 3D Graphics)

X3D is a royalty-free ISO standard XML-based file format for representing 3D computer graphics.

XML (Extensible Markup Language)

XML is a set of rules for encoding documents in machine-readable form.

VRML (Virtual Reality Modelling Language)

VRML is a standard file format for representing 3D interactive vector graphics, which is superseded by X3D format.

1. Introduction

'The journey of a thousand miles begins with a single step.' – Lao Tzu in *'Tao Te Ching'*

1.1. Problem definition and motivation

Cost constraints, higher quality requirements, increasing time pressure through decreasing product life cycles, and supporting a wide variety of products for the customers are the challenges of the today's engineering design processes. The coordination of material handling and manufacturing processes should be optimized for cost/time reduction without a quality decrease in the production system. This situation demands the preparation of as detailed as possible planning of the manufacturing facility.

However, this circumstance conflicts with the necessity of a rapid, low effort solution at the tender stage. In this stage, the focus lies on the rough layout, which enables participants to communicate with each other for possible solutions in future. It is better to get feedback from all participants about the design concepts in the early stages of engineering process, because design iterations getting costly and time-consuming in the late stages. Digital factory concept gains a growing importance as an answer to this conflict between the need of an early feedback and as detailed as possible solution; as it provides continuous usage of IT-tools and digital models in all phases of the facility planning.

The term *digital factory* can be defined as a network of digital models, methods, and tools related to them, which among others consists of simulation and 3D visualization. Its purpose is the integrated planning, implementation, control and continuous improvement of all essential plant processes and resources in connection with the product [VDI08].

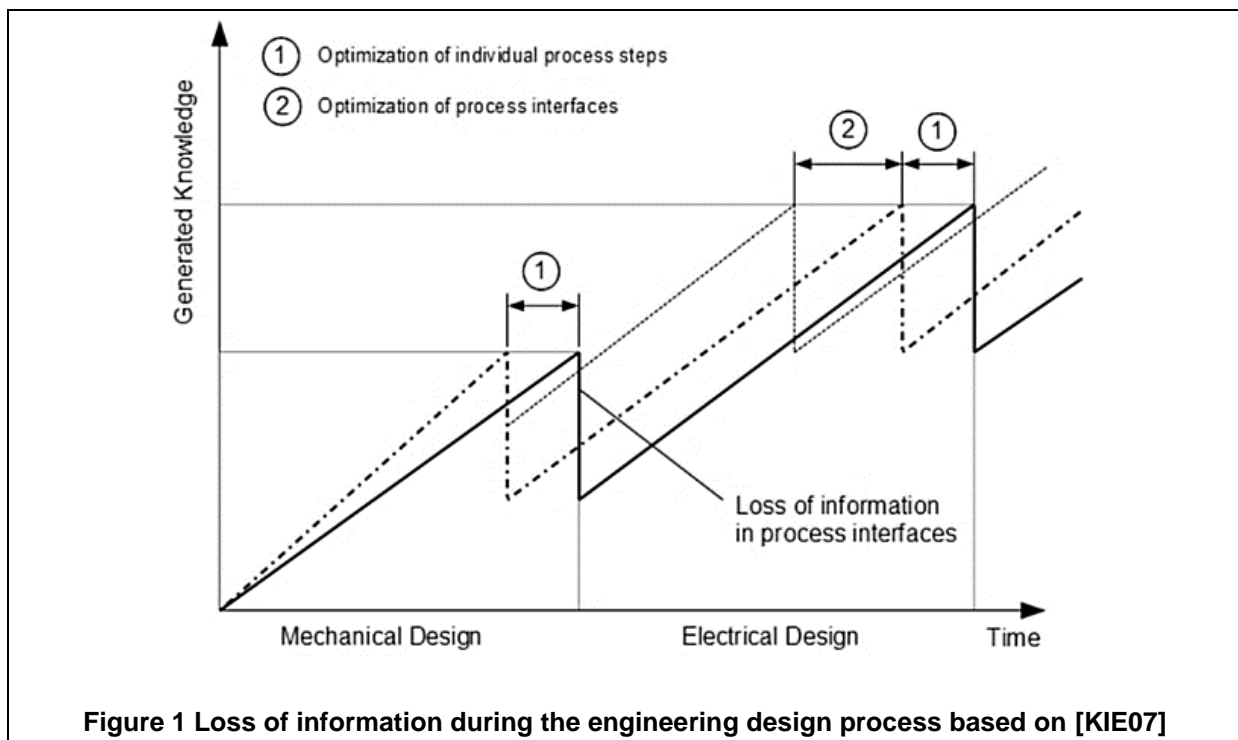
To enhance the validity of simulation methods inside the digital factory concept, simulation models should be realized as realistic as possible. The physical properties of a component, like kinematic dependencies, material properties, and affecting physical forces, should be included in the simulation model to reach this goal. Component motions can be automatically derived from the description of the component's physical/kinematic properties with the application of physics simulation methods, which is useful for testing the system against collisions, unexpected behavior, and faulty operations. Existing solutions like physics engines from the game industry, which are used to calculate the necessary physical/motional equations, can be applied for this purpose [Spi09] [Lac12].

The digital factory methods cover many engineering fields, simulation methods, separate software tools, and their associated data models. They are used in the engineering chain of production planning, which contains disciplines from technical, economical, and natural sciences. It is required to have a project-wide observation and controlling independently of individual engineering tools. Therefore, data management is one of the major tasks within the digital factory [Lüd13].

The essential basement of the digital factory is digital data models of all relevant components of a production system including products and production systems covering structure, behavior, and relations/dependencies. Providing these models is usually a task of the OEMs or the production system developer/user. The geometry, kinematics and behavior information from the models can be used for simulation-based analysis. Another important output is the visualization of the production system based on the component models.

The accumulated data can be processed by many different software tools, which are specialized in fields like mechatronic/physic/material flow simulations, control design, virtual commissioning, construction monitoring or maintenance. Therefore, data exchange is an important part of production system engineering exploiting the digital factory. Nevertheless, it is also a cost factor in the workflow [Dra08] as bad implementations of the data exchange (for example using only PDF files or proprietary data formats) can cause errors and redundant engineering actions.

Figure 1 demonstrates how the loss of information occurs between two exemplary design phases and how much time/cost could be saved through the optimization of the data exchange. The cost of direct translators to realize individual point-to-point connections between tool pairs and the need of personal training to operate them should also be taken into consideration [Saw16].



According to a study [Gal04] for U.S. industry in 2004, the cost of inadequate interoperability for computer, automobile, and aircraft manufacturers is estimated as \$15.8 billion per year. The scope of the study contains design, engineering, facilities management, and business processes software systems and redundant paper records management across facility life-

cycle phases. Although the cost analysis is not actual, it is evident that there is a need for awareness for interoperability-related issues and cost optimization potentials.

Enhancing the data exchange is the most important task regarding the interoperability-related issues. There are mainly two approaches to solve the data exchange problem between engineering tools. The first solution is the usage of a single standard system (an integrated tool suite), in which the responsibility for data exchange lies with the system vendor. Different engineering tools are here integrated into an overall, comprehensive, and leading tool or in a coordinated suite. Unfortunately, they do not fulfill the increasingly progressing specialization needs of suppliers. Adaptation to individual needs and the integration of another tool is a time-consuming task. The second approach is vendor cooperation and support for a common standard data exchange format, which provides the needed flexibility but still has many gaps in the engineering work flow [Saw16] [DrSc10].

This work pursues the second approach and concentrates on a data exchange solution for a factory layout planning tool to improve the interoperability in the engineering process. The layout planning is a structure planning activity for all systems and functional blocks in the production system. It contains all production and logistics processes, material flow design and other specific engineering fields [Paw14]. It is, therefore, an interdisciplinary planning task.

Unfortunately, a solution for the computer-aided layout planning of production systems, in which not only the geometric structure but also the material flow can be designed in detail, simulated, tested, and executed seamlessly into operation, is not yet existent. Many layout planning tools have limited abilities for exchanging data with tools from other engineering fields/disciplines-like simulation or virtual commissioning. A neutral data exchange format could reach across the different engineering steps, though. Significant time and cost savings during data acquisition and processing as well as during the model creation could be achieved with the help of such medium [Wid09]. AutomationML (AML) can fulfill this requirement as an open, vendor-independent, and standardized exchange format. It can be used to achieve a consistent and non-redundant data exchange for mechatronic components like production machines or conveyor systems as well as logical planning data such as material handling and manufacturing process definitions [Dra08].

As a description format, AML has the technical qualifications for the modeling of production, logistics, and material handling systems. The standard supports the definition of semantic roles and classes, through which the necessary components and their properties, as well as process flow and parameters, can be described. So far, the format has been used mainly in other areas like virtual commissioning, robotic systems, or process communication systems [Wid09]. On this basis, there are only a few and rudimentary elements for material handling and logistics systems being used. A gap in this field is visible in this regard.

1.2. Aim of the work

To close the named gaps, the first objective of the work is to enhance the interoperability of the software tools in the mechatronical engineering process, specifically for the layout planning phase. The description/creation of the mechatronical object models for material handling components in a simulation environment is an important part of this task. It is also desired that a standardized material handling model library is generated through the modeling efforts. The models, which are the subject of the data exchange, are to be used in:

- realistic illustration of material handling and manufacturing systems using a physics simulation and virtual reality techniques,
- automatically derivation of test cases in virtual commissioning applications,
- material flow simulation for analyzing and optimization.

Development of new software-based solutions like new importer/exporter libraries to enhance the data exchange in the digital factory toolchain is another primary goal. The emphasis of this work is on:

- Contemplation of material handling systems as a composition of mechatronic systems,
- Modeling of material handling/logistics components and processes containing geometrical, kinematical and physical perspectives, and
- Implementation of software based data exchange solutions for a layout planning software tool in a virtual commissioning toolchain.

The boundaries of the research topic should be clearly defined to prevent misinterpretations. This work is NOT directly about:

- How to realize virtual commissioning in material handling systems
- How to implement physics simulation for mechatronic systems using physics engines
- How to simulate material flow with visualization, analyzing and optimization

It is mainly about preparing the component models for data exchange and realizing data exchange solutions to optimize and enhance the workflow of the engineering tools for the subjects mentioned above.

The results of this work will be applied to tarakos software tools, taraVRBuilder for layout planning and taraVRControl for process visualization, to demonstrate the validity of implemented concepts [tar15] [tar08].

1.3. Structure of the work

There are three main subject categories in this work. State of the art is the first one. In this part, the fundamentals of automation engineering and mechatronical engineering process, an introduction to the digital factory concept and the basics of mathematical and software-

technical aspects are given. The state of the art is followed by a literature review to introduce the essential sources of the thesis and specify the differences. After that, an analysis of requirements is provided to create a metrics for the solution strategy. The section “data exchange formats” begins with the introduction of the candidate formats for data exchange to decide, which ones fulfill the defined requirements. An overview of the relevant data exchange formats like AML, COLLADA, X3D/VRML and FMI is also a part of this section.

The second category is the implementation of the solution. The modeling chapter illustrates how the models are prepared for the data exchange, including the geometric, kinematical, and physical description of the material handling resources and process definitions. The implementation concept illustrates a particular software related solution for the data exchange of material handling components in the layout planning phase.

The third category is the application of the created models and self-developed software libraries. A sample system is shown as an example for the usage of the models and the data exchange software components. The demonstrators for the AVANTI project [AVA16] and the commitment to these demonstrators are parts of the application, too.

In the end, a brief technical and economic evaluation are also provided.

There are some special notations in the formatting of the text. The XML elements in the text are shown between characters “<” and “>”, i.e. <element>. If the name of an element is defined by an XML attribute, for example, <example Name = “ElementName”>, then it is depicted as “*ElementName*”. The class names and other software-related elements like class attributes or members are written like *ClassName* as they should be differentiated from the other content.

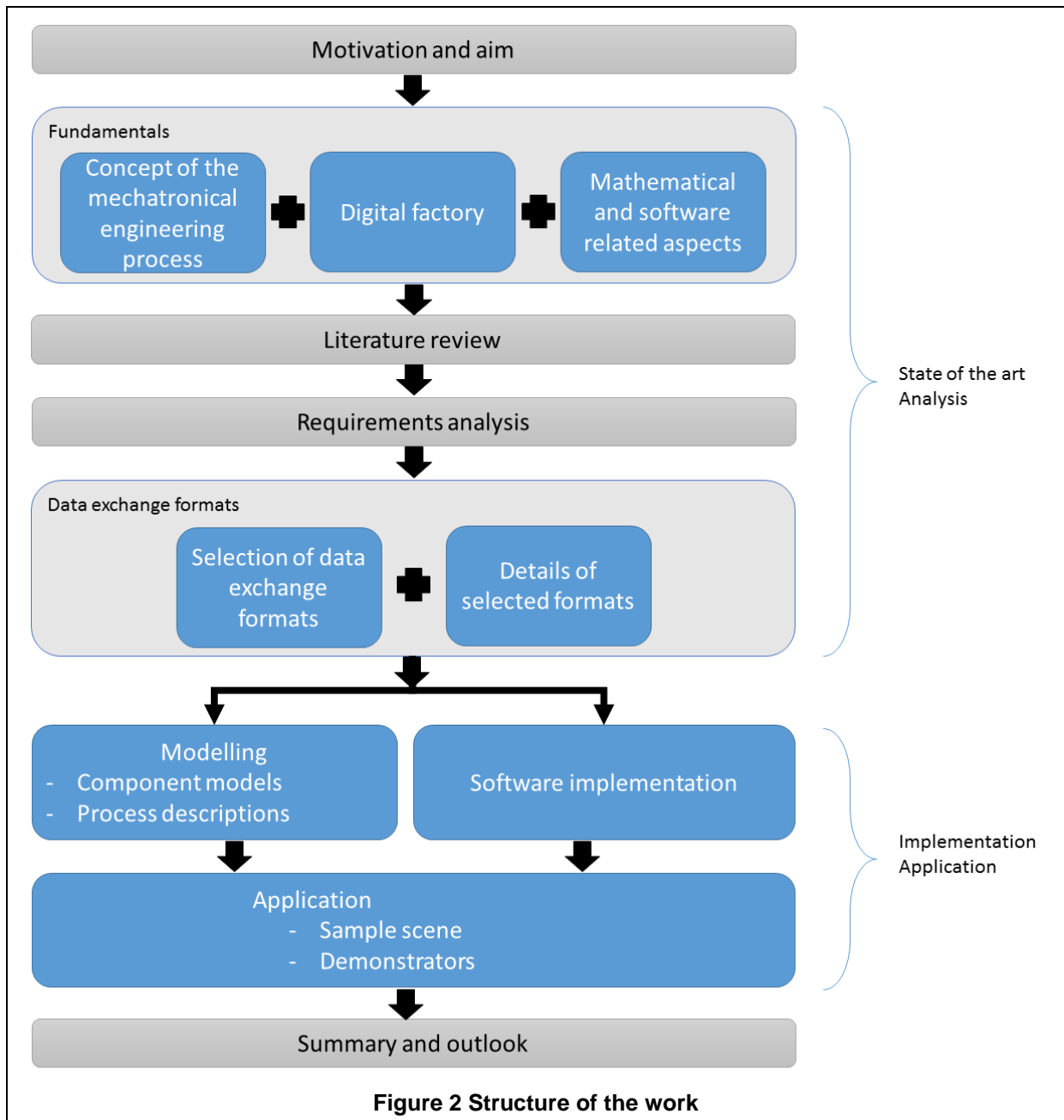


Figure 2 Structure of the work

2. Fundamentals

“For we do not think that we know a thing until we are acquainted with its primary conditions or first principles, and have carried our analysis as far as its simplest elements. Plainly therefore in the science of Nature, as in other branches of study, our first task will be to try to determine what relates to its principles.” – Aristotle in ‘Physics’

This chapter provides the necessary information for mechatronics notions, mechatronical engineering process phases, and mathematical and software-related aspects. It is a collection of fundamentals to understand the further sections.

2.1. Basics of automation engineering and mechatronics

The following section gives an overview of the mechatronics concept, which is used in the context of the planning of production plants. It is also described which information is required to illustrate a mechatronic system and how that information is used to describe the design objects in the planning process. The primary definitions and mechatronical engineering process (MEP) phases should be explained to understand the information types and the role of data exchange in the MEP, which is convenient to comprehend the fundamental ideas of this work.

2.1.1. Mechatronic system

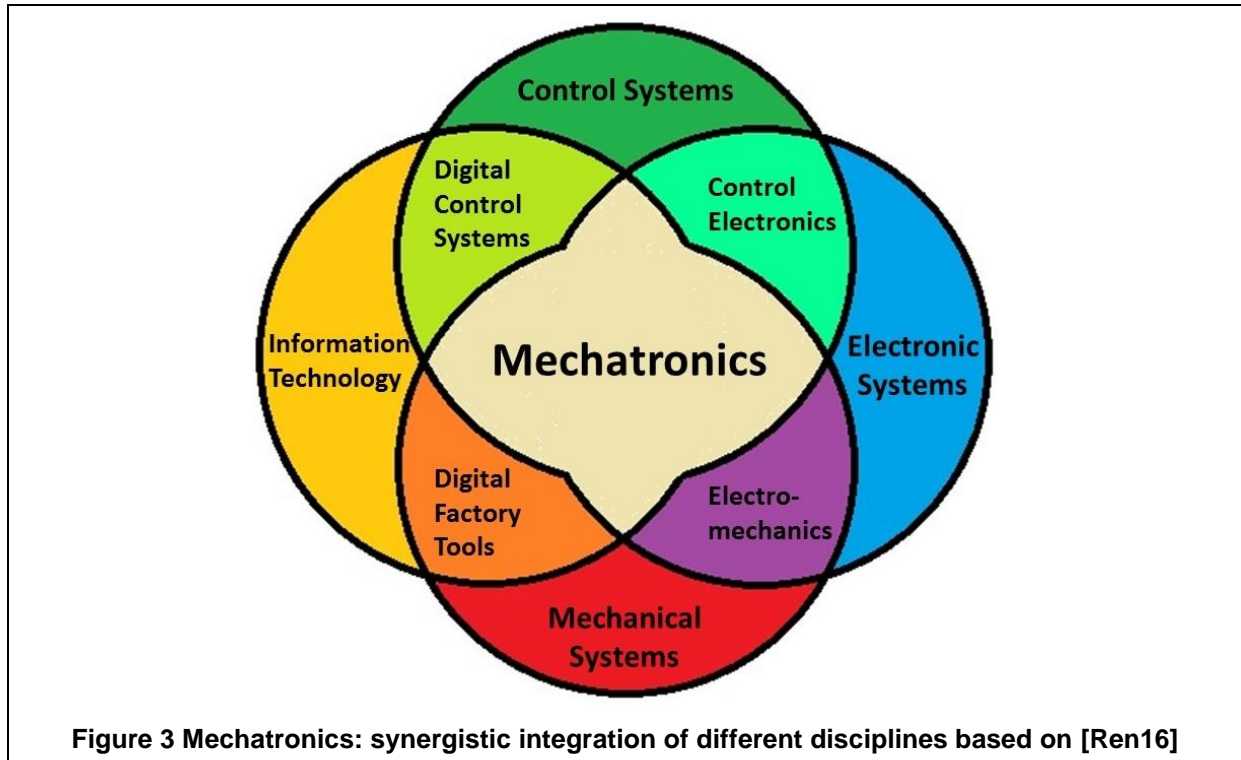
The notion mechatronic system cannot be thought separated from the general definition of mechatronics. Several descriptions of mechatronics/mechatronic systems can be found in the literature. Some important ones, which are convenient with the context of this work, are given in this chapter.

Integrated mechanical, electronic systems result from an eligible merging of mechanics, electronics and control/information processing. Therefore, these areas influence each other correspondingly. First, a displacement of functions from mechanics to electronics and then the addition of extended and new features is considered to reach the actual concept idea. Finally, systems with certain intelligence or autonomous capabilities are produced. For this kind of mechanical electronic systems, the term "mechatronics" has been used for several years [Ise07].

The International Federation of Automatic Control (IFAC) Technical Committee on Mechatronic Systems uses the following description [Int16]: “Many technical processes and products in the area of aerospace, mechanical and electrical engineering show an increasing integration of mechanics with electronics and information processing. This integration is between the components (hardware) and the information-driven functions (software), resulting in integrated systems called mechatronic systems.”

Mechatronics is an interdisciplinary field, in which the following disciplines work together, as seen in Figure 3:

- Mechanical systems (mechanical elements, machines, precision mechanics);
- Electronic systems (microelectronics, power electronics, sensor and actuator technology);
- Information technology (systems theory, software engineering, artificial intelligence).
- Control systems (digital control systems, control electronics)



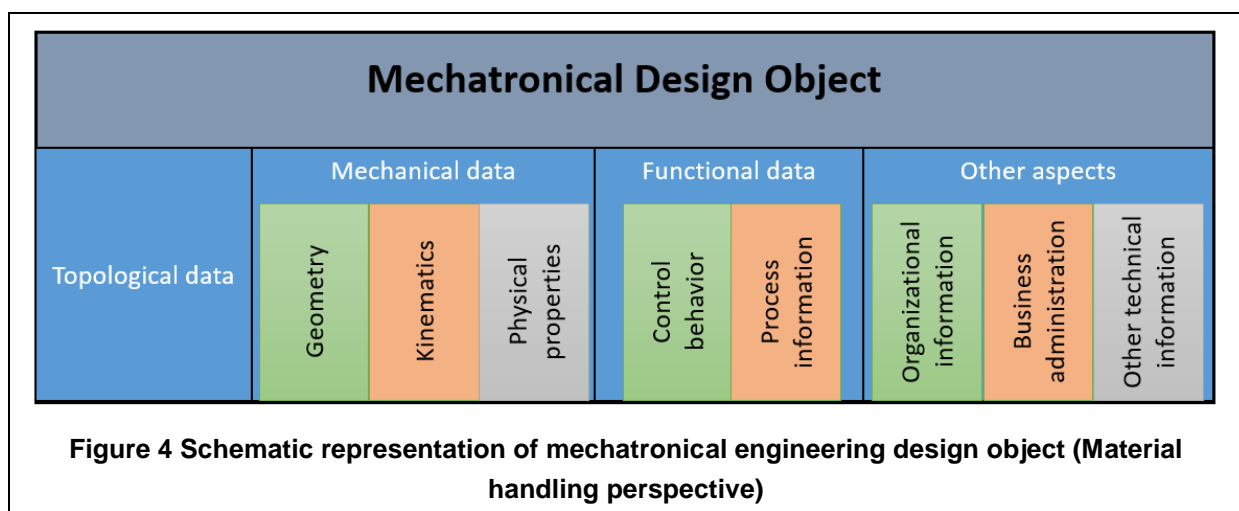
To ensure a clear understanding of the term mechatronics concept in this work, it should be understood as follows: A mechatronic system consists of subsystems in engineering disciplines like mechanical, electrical, pneumatic, hydraulic and information/control technology, which cooperate through a spatial and/or functional integration of the different working principles with the purpose of forming synergy effects [KIE07].

2.1.2. The mechatronic design object

The term of the mechatronic design object is referred to all the components of a system in the MEP, providing the technical control for the automation. It can be used to control functions (actors), measurement capabilities (sensors), kinematic features (conveyor) and other composite plant components [Bart09].

A mechatronic system consists of from one to many mechatronic objects, where they can also realize a mechatronic subsystem and include several mechatronic objects [Hun12].

The mechatronic design object regards not "only" the functioning way of a system but also the interaction of various disciplines in the MEP. The function of planning objects from a user perspective is an engineering interface, representing information, which depends on the respective engineering activity. Which information it contains changes due to the tasks, which this object should fulfill for mechatronic system notions. The mechatronic object is a "collector" of all needed information for the whole life-cycle of a system design and is in interaction with data structures of engineering modeling tools [Hun12] [Tet09]. The main subject of this work is the simulation process in material handling, so in Figure 4 a schematic representation of mechatronic engineering object can be seen according to the information types in material handling.



A basic concept for the mechatronic engineering object is object-orientation in the software engineering according to [Ber06], [Kas95], [Weu98] (For more information about object-orientation, please see chapter 2.3.5.1). The functional models are transformed into an object-oriented description. Components, interfaces, and connections are considered as objects. The functional behavior of the physical component is taken over by the component objects.

A mechatronic engineering object for a material handling system covers the information ranging from mechanical data like geometry, kinematics, physical properties of the mechatronic unit over functional data and control data, communication system descriptions and control code up to technical data, like energy consumption, and financial data, such as purchase costs or maintenance cycles [Hun10].

2.1.3. Mechatronic engineering process

The term "mechatronic engineering process" (MEP) covers the planning process of the production systems, the main phases and main goals of it, and the hierarchical order/sequence between these phases [Lud10].

The description of the MEP includes the parts of product development, but mainly describes the development of the plant to manufacture the products. All necessary steps for planning

and commissioning of a plant should be listed, and sorted into a logical time sequence to have an overview of the necessary tasks.

The determination of essential system components or manufacturing cells results from the characterization of the MEP. Necessary tasks are often specified in simple description methods such as Gantt charts using Microsoft Excel or any other project management tool at the beginning. An in-depth knowledge of the product is needed for the further detailing of the processes, especially during joining sequences for the production process, which in principle functions like the blueprint of a block model. The plant components must be connected effectively with each other involving different engineering disciplines like mechanical, hydraulic, electrical and control engineering. Many tests must be made such as collision calculations or system security procedures. Robotics and control systems require route planning and process descriptions. Subsequently, the plant can be built, tested, commissioned and operated.

All data types, describing the product, the plant, and the production process, should be mastered in the system design. First, comes the topological information such as hierarchies or groupings of system components, each with type descriptions, requirement specifications of the operator, characteristics, interfaces, dependencies, and connections to other elements. Geometry (and partly kinematics) information is required not only for the components and for the product itself, but also for the production halls with fixed facilities such as columns, walls, and doors, supply and disposal peripherals such as electricity or compressed air infrastructure for the engineering design and visualization.

Robotics is also critical for the factory automation. Kinematic relationships of robotic cells should be described on trajectories and enriched with process information such as e.g. welding or gluing. To understand the production process, sequence descriptions are created for the production lines and the cells or modules, which are enriched with signaling information and used in the control programming. The individual components include uncontrolled behavior that can be described by phase diagrams and used for instance in simulations. A variety of devices are connected via field buses and communicate with each other. This list is not exhaustive, and other aspects can be said e.g. the supply of electricity, hydraulics, and pneumatics or the design of material routes.

The MEP for manufacturing systems is characterized by different engineering branches and different types of data involved within the engineering task areas. There is a variety of approaches for the subdivisions of the MEP, for more information about the subject the literature [Hun10], [Lud10], [Pah13], [Ise07], [Sau04], [Pro14] can be examined.

The entire MEP is divided into four phases with six steps in this work. These are shown in Figure 5. More explanation about these phases related to the digital factory concept can be found in the next chapter.

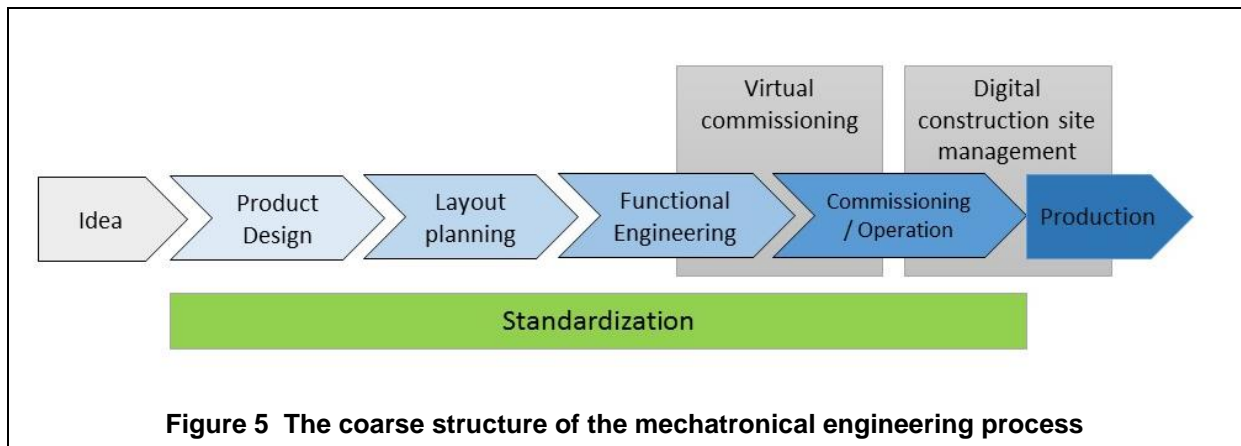


Figure 5 The coarse structure of the mechatronics engineering process

2.2. Digital factory methods in the mechatronics engineering process

The definition of the digital factory due to [VDI08] was already given in the introduction. The digital factory methods do not provide an integrated solution, within which all engineering activities can be executed continuously from factory planning to the operation of the facility. It rather stands for several engineering tools, which form a heterogeneous system landscape and their coupling is done via interfaces [Hun12]. The usage of digital factory methods for each MEP phases and their connections will be explained in the following chapters. Thereby it is aimed to represent what are the data sources for a layout planning tool and to which type of tools the information can be transferred for further data handling.

2.2.1. Product design

Product design can be defined as a problem-solving exercise, oriented to a certain purpose. Its objective is the conversion of a need into a product, designated in the first instance as an abstract concept regarding its general functionality. It is the first step in the MEP. The information, from which materials and which production steps the product has to be manufactured, is already generated here. This information is provided to the next step facility layout planning in the form of bill of materials (BOM), production instructions, bill of operations (BOO) and material flow plans [Hun12].

A systematic development and use of modern software design tools are required for the product design in mechatronic systems. Product design in mechatronics is also an iterative procedure as with any design. However, it is wide-ranging than pure mechanical or electrical systems. The integration of engineering across traditional boundaries is characteristic in the development of mechatronic systems [Ise07].

The product design is the first step in the MEP as like in any investment planning. The actual production item is devised in this step. There must be an idea for a product that meets a particular function. In this highest level of creative action, there are no limitations in thoughts such as feasibility, finance, or market interest. Innovative techniques are used such as a

morphological box or mind mapping. Once an idea is originated, follows a basic test whether the covered product has an interest group. If there is a potential market, the design of the actual product could start. For this purpose, the previously vague notions of the product are implemented in detailed technical drawings. It is followed by computations regarding mechanical, electrical, pneumatic, hydraulic, information technology, manufacturing orders and ease of use. These procedures result in lists of the necessary materials and instructions on how the individual components must be produced. In particular, a list of steps and processes should be performed in a fixed sequence, if necessary with alternatives, to complete the design of the end product. However, it is not yet clear how these steps are implemented. Results of the product design and material lists are thus manufacturing instructions, operation sequence graph, material flow diagrams, which are re-used in the next step, in facility layout planning [Bau09].

2.2.2. Layout planning

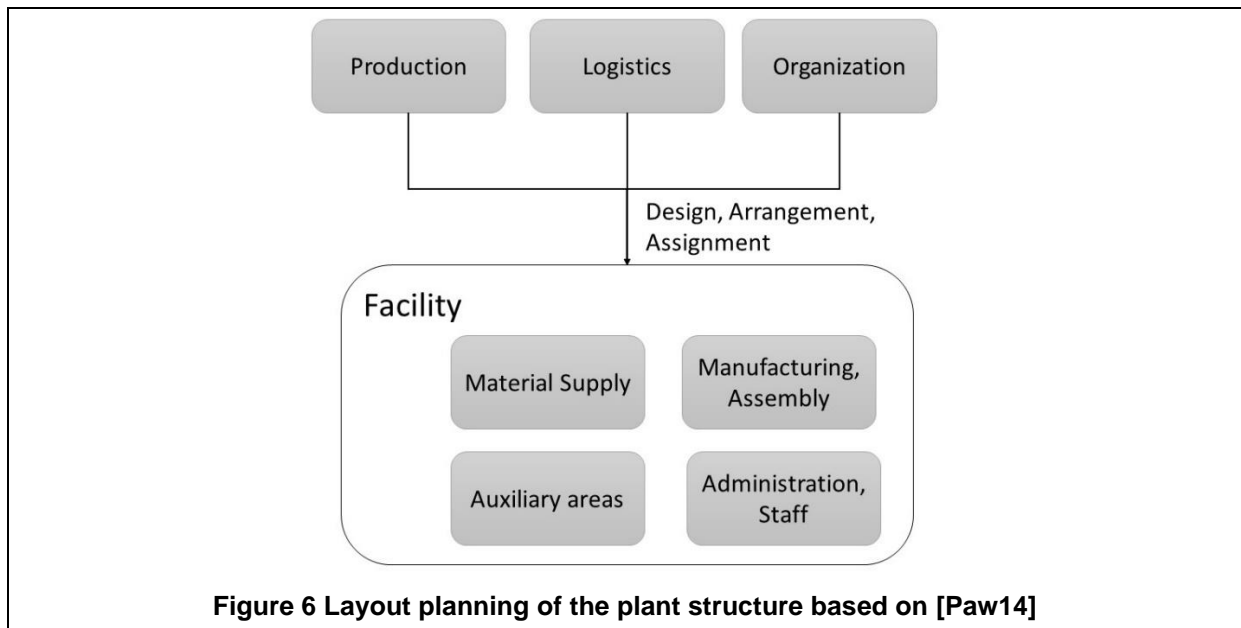
The facility layout planning is the actual technical implementation of the material lists and preparation instructions from the product design. It is clarified in this stage which devices should be implemented for the production processes of the product. For this purpose, either existing standardized components are used and adjusted or -if necessary- new components are designed from scratch [Bau09].

The layout planning process is a structure planning activity for all systems and functional blocks in the factory. It contains all production and logistics processes, material flow design and other specific engineering fields. It is, therefore, an interdisciplinary planning task. The various factory elements and their mutual influence are analyzed in this step and integrated to an overall economic solution.

The planning of the plant structure and thus the arrangement of functional units are carried out according to production, logistical and organizational conditions (see Figure 6).

Functional units are mostly:

- Production (manufacturing, assembly)
- Logistics (material handling, storage, transport)
- Auxiliary areas (maintenance, supply, etc.)
- Administration and staff areas



The essential characteristic function units are production and logistics. They depend on the products to be manufactured (product structure) and the necessary procedures (technology structure).

Layout planning contains the ideal arrangement of the individual functional units along with the material flow analysis. For this purpose, material flow simulation is used which can be for the entire system or individual cells. The flow behavior of a complex system or specifically the behavior of the material flow through the time can be analyzed [Hun12]. Results of this analysis and target area calculation are implemented in an optimized layout schema. The individual functional units are linked and arranged for the optimized material flow; thereby a first rough draft of the future facility can be determined [Paw14].

tarVRBuilder is a layout planning tool, which utilizes virtual reality methods to make material flow analysis [tar15]. More information about this tool will be given in chapters 2.3.4 and 7.1.1. There are also several discrete event simulation tools for material handling systems like PlantSimulation, AutoMOD or Flexsim to provide solutions for layout planning, checking the system's overall function, determination of capacity limits and bottlenecks/jam risks regarding the material flow, development, and test of storage and routing strategies [Sei12].

2.2.3. Functional engineering

Functional engineering is the core process of the MEP. The concrete data on the system are here created from the results of the facility layout planning. The principle is as already known: which components and which tasks are executed in what order, to create the desired product.

However, it should also be clarified [Bau09]:

- Which specific equipment from which manufacturer are used at what point on the factory floor,

- How the necessary media connections are designed,
- How the control code looks like,
- Which possibilities of manual interference are there, and
- How the various devices and cells communicate with each other and are connected.

To resolve these open points, the functional engineering is divided into many sub-activities, which are carried out partly independently of each other, but are related to each other. Such as [Hun12]:

- Layout and process refinement (including accessibility and maintainability simulations),
- Mechanical design of the production units,
- The electrical design for the connection and wiring diagrams, the individual input/output (I/O) points, the components relevant to the automation,
- PLC programming,
- Offline robot programming,
- Human-Machine-Interface development,
- Robot simulation.

These tasks can be ordered in a sequence. For an overview, the list of fields is created in the same order as in the real world exists. However, sometimes these steps proceed also parallel. The parallelism of the tasks is a goal of further developments, as part of shortening the development time in the MEP [Bar091].

At the end of the functional engineering, a complete specification of the detailed implementation for the manufacturing plant is generated. This specification is, on the one hand, a concrete statement to build the facility with construction plans and wiring plans, like a technical drawing of a building. On the other hand, includes not only the plain statements but also programs, such as robotics and control programs and CNC codes.

Nowadays, the step of functional engineering ends mostly at this point. However, it is feasible and desirable, that a simulation technique is used to simulate the functions of the system and to enable a virtual run-off test. The individual modules of the control software can be tested in the simulation model if they can be integrated into the model. Test cases for the control modules can be generated automatically during the simulation execution. The testing enables the review and correction of PLC code at an early stage [Gut03]. This approach will be discussed in the next subsection “virtual commissioning.”

2.2.4. Virtual commissioning

The simulation of the manufacturing cells or maybe -in the future- the whole facility can be used to verify control programs and thus increase their overall quality before real commissioning. This method is called virtual commissioning [Sch09]. In this approach, virtual prototypes are created for the commissioning of control software in parallel to the manufacture

and assembly of the particular production system. Virtual commissioning is an intermediate step crosses the boundary between the functional engineering and real commissioning/operation [Zäh06].

As the virtual commissioning is used in a late phase of the planning process (from the beginning of PLC and robot programming), no fundamental decisions about the production concept or the processes can be taken. Nevertheless, valuable information on the quality of engineering results can be found out. These are in detail [Ber09]:

- Validation of PLC and robot program logic (Sequence, interaction, error behavior),
- Ensuring collision prevention between robots and between PLC-controlled mechanics, and
- Deriving exact statements of cycle time.

Also, boundary conditions can be simulated in the test environment of the virtual commissioning without any danger to the operator and the production plant. That is usually not possible in real commissioning because of the operation risks and lack of time. Virtual commissioning can also be helpful for remote diagnostics and remote maintenance. For this purpose, the actual status of the control system can be transmitted from manufacturing system to the simulation environment through a communication pipeline. Diagnostics and enhancements can be determined by the analysis of the simulation [Spi09]. The simulation events can be visualized in runtime or post-runtime with a process visualization tool like taraVRControl [tar08].

Therefore, virtual commissioning has several advantages in the MEP, like [Gri10]:

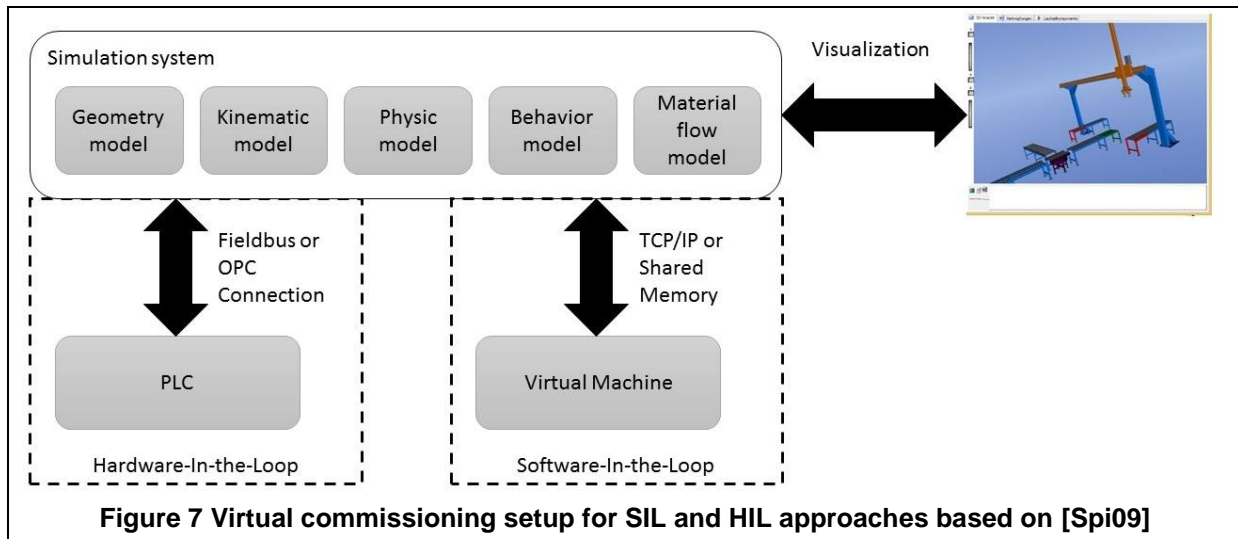
- More robust, higher quality, and more mature PLC programs,
- Less physical presence of the control engineer in the real plant due to the development of the PLC programs towards a virtual plant,
- Earlier and faster ramp-up of new manufacturing lines,
- Better optimization due to the ease of experimenting with the virtual plant,
- Early feedback from the person in charge for the production whether the design suits his requirements and specifications,
- Ease of modification for the new products through the simulation.

The workflow of the virtual commissioning can be divided into five phases [Spi09] [Str15]:

- Gathering of the all necessary information like specification of the components, existing standard component models, data for the creating new component models e.g. CAD models or physical properties,
- Creating the simulation model with software tools with the help of gathered information,
- Development of the PLC programs for the desired kinematic and dynamic behavior,
- The configuration of the simulation environment,

- Test and validation of the PLC programs, and
- Optimization of the PLC programs due to the results of testing and validation.

There are two approaches for creating the modeling of production equipment in the virtual commissioning: Software-in-the-Loop and Hardware-in-the-Loop (see Figure 7):



1) Software-in-the-loop (SIL): This consists of a simulation of the peripheral production equipment as well as the control hardware itself. It can be done either with the compiled software in the host system or with the usage of a software emulator. In the SIL approach, the simulated control program is connected to the simulation system via TCP/IP or shared memory interface so that it can be executed on more than one computer. A standard PC hardware can be used for the simulation environment. The benefit is the usability of a virtual machine, which means no need of hardware components during the test phase [Spi09]. The low availability of up-to-date control simulation packages for a particular control version and the rather high abstraction levels of generalized control simulation models are the main problems of this approach [Rei071].

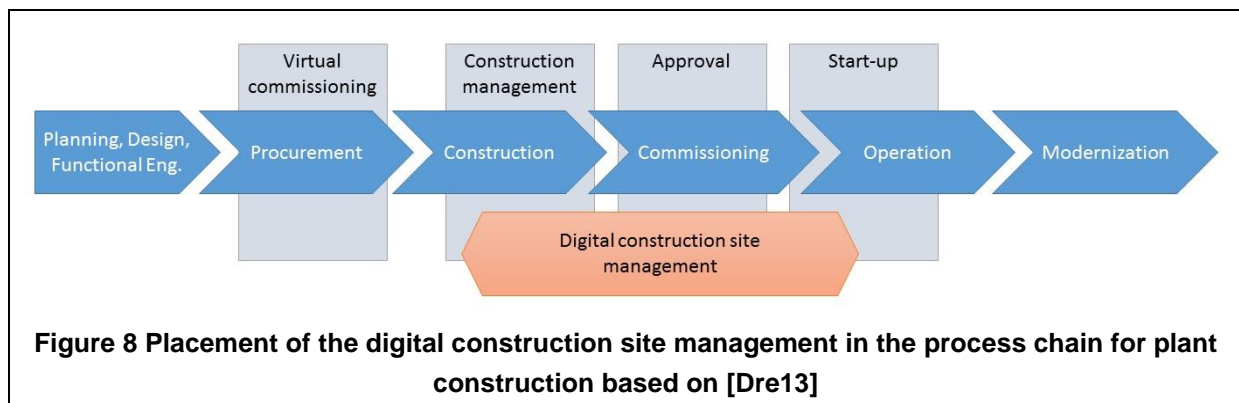
2) Hardware-in-the-loop (HIL): Here, the peripheral production equipment is simulated in real time and connected to the real control hardware. The HIL approach allows conducting commissioning and testing of complex control and automation scenarios under laboratory conditions. It can be applied on different levels of plant hierarchy from field level up to the line or plant level [Rei071]. The main advantage of this approach is the possibility of transferring the PLC code into the real production after virtual commissioning [Spi09].

2.2.5. Digital construction site management

Digital construction site management is a new engineering field in the MEP, which is coming after the virtual commissioning phase. The basic idea behind this approach is creating a bi-directional link between the project plan and the technical installation layout. This connection allows the graphical display of the system structure and the interactive display of the current construction progress and project status in a layout-related representation. Reported defects

can thus be automatically redirected to the responsible project staff by the system. Potential problems are detected early, and appropriate countermeasures can be initiated in time. Since all responsible parties are informed in real time about the status, the gap between the project management level and site controlling is closed. This concept can significantly reduce the effort for progress tracking, the construction documentation and status reporting [Bie14].

Figure 8 shows the position of the digital construction site management in the process chain and the relationship with other steps in the MEP.



The basis for the visualized construction site layout is the project planning tools (e.g. MS Project, RPlan) and layout planning tools (e.g. Microstation, ProcessDesigner). The schedule structure and layout of the construction are connected with each other to create the digital management of the construction site [Dre13].

2.3. Mathematical and software related aspects

As this work is introducing a software solution for data exchange between engineering tools, it is necessary to understand mathematical and software related basic principles. The following chapters explain the essential notions, which are applied in the modeling process (Chapter 6) and software implementation (Chapter 7).

2.3.1. Model description

Models are important in every scientific research, but what it means can vary from the context. They can be used to fulfill various requirements. A model can be a substitution of a selected part of the real physical world (target system). Such models can be models of phenomena or models of data depending on the nature of the target system. Other than that, a model can be a representation of theory in a way that it interprets the laws and axioms of that theory. Scientific models can be defined in both senses at the same time. Idealized models are used in most engineering applications. An idealization is a deliberate simplification of a complex reality to make it manageable for scientific/engineering purposes. i.e. frictionless planes, point masses, infinite velocities, isolated systems, and similar assumptions [Fri12].

There is a variety of model definitions from the various fields of science in literature. Parallel to the explanation above, [Epp08] assumes that a model can be defined as the projection of a real or intellectual object that reflects only the properties of the object in the context of a particular task of interest. The model definition, which is used in this work, should contain applied science (engineering discipline) aspects. Therefore, it can be understood due to [VDI93]:

A model is a simplified representation of a planned or real origin system together with its process parameters in another conceptual or actual system. It regards analysis-related properties and differs from the actual object inside purpose-dependent tolerances.

2.3.2. Physics simulation

The VDI standard 3633 defines the simulation as "imitation of a system with its dynamic processes in an experiment capable model to reach statements, which are transferable to the reality" [VDI93]. The goal of the simulation is to acquire statements about the behavior of the system, which are close to the reality as much as possible. The reality of the simulation can be increased through the integration of the physical aspects into the system model. In this context, the system model is a mathematical equation in a computer-representable form, which defines the dynamics of the physical system [Fri11]. The part of the simulation environment, which is responsible for the numerical calculation of the differential equations, is called a Solver [Pri04]. There are different kinds of numerical methods in engineering simulation such as Finite Element Method (FEM), Finite Difference Method (FDM), multi-body simulation (MBS) or rigid body simulation. Rigid body simulation is often used for collision detection, collision response, and treatment [Spi09]. An example application of an engineering simulation: the installation process of a car, which is to be tested under various operating conditions.

The traditional method to establish the visualization of material flow is animating the components with the help of scripting¹ without the physics aspects. That means, forces, collisions or joints has no effect on the rigid body. The rigid body will be under full control of the scripted behavior during a change in the position, which can produce unrealistic results under some unexpected incidents e.g. a collision. With the help of a physics-based animation excluding scripting, kinematic rigid bodies affect the motion of other rigid bodies through collisions or joints, which improves the reality of the simulation. It is also possible to model effects of sensors, actors and other material handling elements. The simulation results like the sensor signals or collision reports can be used by a test system to improve the validity of the virtual commissioning. Physics engines are coupled with the simulation environment to provide such physics-based simulation.

¹ Scripts are programs to extend the features of the existing software components but they are rarely used for complex algorithms and data structures [Ous98].

2.3.3. Co-simulation

The growing complexity of the simulation process often causes hybrid solutions, which contain several models with differential or algebraic, time continuous or time/event-discrete equations. Such complex multidisciplinary systems are not easy to handle alone with a single software tool. It is also possible that only a particular tool supports the format of a subsystem model. Moreover, sometimes an engineering problem should be solved by a simulation tool, which produces the best results for a specific field. It is then very useful or even necessary to couple various simulation tools with each other. This concept brings significant advantages in simulation environments with real system components such as in the hardware-in-the-loop simulation (HIS) [Bas11].

Co-simulation is a method to simulate the interconnected systems that can be time continuous or discrete time. The modular structure through the distribution can be used in all stages of the simulation process (time integration, pre-processing, post-processing). The data exchange between subsystems takes place only in discrete communication points (certain time points). The subsystems are independently calculated by their proprietary solver in the time step between the communication points. A subsystem can only be coupled with co-simulation environment if it can support data exchange at the communication points during the simulation [MOD14].

A common method for the co-simulation is the master-slave architecture. In this approach, the simulation tools are not directly coupled, but a so-called co-simulation-master handles the entire communication and synchronization. The tasks of the master are the distribution of signals, analysis of the connection graph, choose of the appropriate simulation algorithm and control the simulation according to this algorithm. Slaves are the individual solvers/tools, which are responsible for their part of the problem. The slaves can communicate directly only with the master, not with each other. The tasks of the slaves are data transmission, execution of the control commands, and the return of the status information. (See Figure 9) [Blo12].

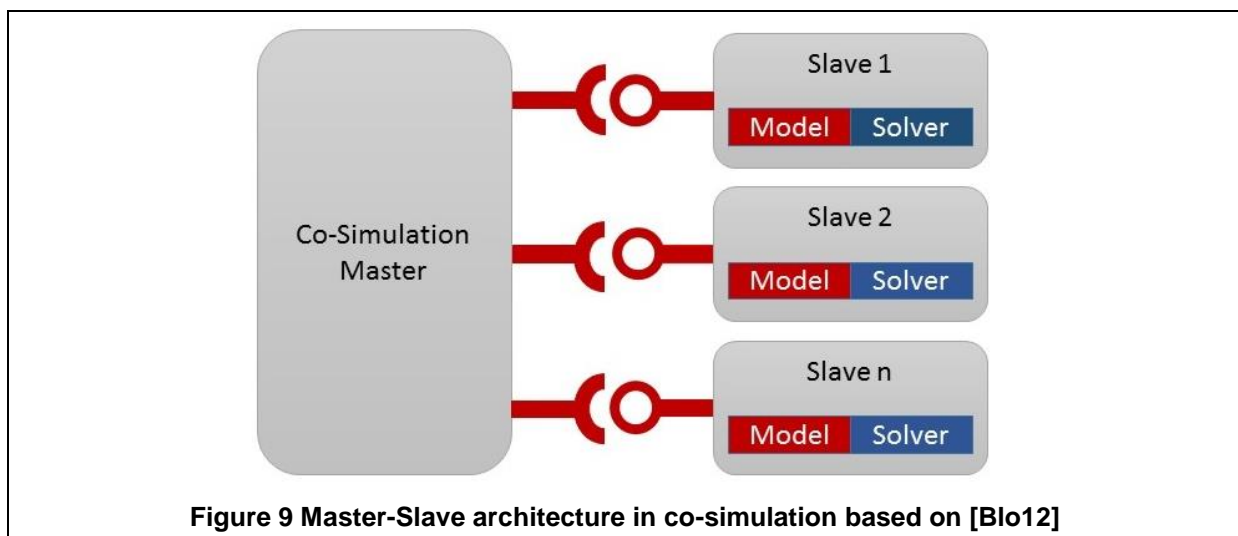


Figure 9 Master-Slave architecture in co-simulation based on [Blo12]

2.3.4. Virtual reality

“(...)On the one hand, VR marks the radical reduction of the wealth of our sensory experience to -not even letters, but- the minimal digital series of 0 and 1, of passing and non-passing of the electrical signal. On the other hand, this very digital machine generates the "simulated" experience of reality which tends to become indiscernible from the "real" reality, with the consequence of undermining the very notion of "real" reality - VR is thus at the same time the most radical assertion of the seductive power of images.” – Slavoj Žižek [Žiž06]

Virtual reality (VR) can be defined as a simulated (i.e., virtual) environment that can be experienced as comparable to real world objects and events with the help of technologies including computers and various multimedia peripherals. Users can interact, move and manipulate virtual objects, and perform other actions in a way that creates a feeling of actual presence in the simulation scene [Wei98].

There are mainly two types of VR applications: 1) The system model can be viewed using the peripheral monitor (non-immersive VR), 2) It can be utilized using wearable systems which are providing the features like full-scale representation, stereoscopic viewing, and head-referenced navigation (immersive VR) [Bei00]. See Figure 10.



Figure 10 Usage of wearable devices (immersive VR) for layout planning, shown with taraVRBuilder and Oculus Rift VR headset²

² <https://www3.oculus.com/en-us/rift/>

Material handling systems and logistics networks should be adapted continuously to changing customer demands due to the increasing need for agility. As mentioned before in chapter 2.2.2, layout-planning tools can be used for a detailed and well-proven system model before the real commissioning. There are many possible material flow strategies during the design of logistics systems for an individual task. A VR-based planning tool with production and logistics visualization (for example taraVRBuilder in this case) can help to choose the convenient strategy as it offers a conversation platform to everybody involved in the process, from the planner up to the operative staff [Rei08]. It is also possible to check the physical validity and efficiency of co-working machines and control algorithms [Par08]. Moreover, the placement of the machinery, the storage places for manufacturing materials, the supporting structures (like ceilings and walls) and travel paths for workers and vehicles can be optimized easily. The effect of these changes on the material flow and the execution of the manufacturing can be observed in the early phases of the EP with the help of the virtual environment. The geometrical representation of the system can be viewed from any desired angle in the virtual scene, which gives a deeper understanding of functionality [Wei98].

Already at the beginning of the MEP, CAD/CAM tools are used to construct both products and manufacturing and logistics facilities. They provide 3D geometrical data to the planner, which could be used as a basis for VR [Rei08]. In practice, the geometry model is a polygonal representation of the simulated objects to allow fast rendering in real-time. Tessellation algorithms are usually used to substitute the object surface as a mesh of polygons. The number of polygons (polygon count) for the entire model is a critical factor in VR applications since it affects the rendering speed [Bei00]. Real-time rendering requires the generation of at least 20-30 Hz frame rate or even 60 Hz if cooperation with PLC needed because the PLC in production systems are executed with a cycle time of 10ms. The cycle time of the simulation should be less than the cycle time of the tested control program [Spi09]. Increasing complexity (higher polygon count) requires more computational power to reach this requirement.

Some additional information i.e. appearance of objects (color, reflection characteristics, textures), the lighting environment, possible animations, interactions, sound, as well as behavior and functionality should be added to the plant model to allow realistic interaction in the virtual environment [Bei00]. These object properties can be experienced using different senses. For example, the color of an object can be sensed visually. The texture or the shape can be sensed in visual and possibly also in haptic domains.

In [Mih14] following content categories are defined for the virtual environment:

- Topology of the components
- Objects as three-dimensional forms. The user can observe and manipulate these entities.
- User interactions with the environment (actions)
- The forms, which are controlled via interfaces, or avatars (a virtual object that represents a real object) of users themselves (intermediaries).

- User interface elements within the virtual environment for control applications such as virtual buttons, switches, or sliders.

However, they are not sufficient to define information types in industrial VR applications. The motion of the objects according to the scripted or physics based animation, process information, sensor feedback, and other inputs should be taken into consideration as well. The components in the VR environment should not be considered as static objects; they should be understood as a dynamic entity as defined in Figure 4 in Chapter 2.1.2 in this regard.

By creating the virtual environment, it is possible to replace the time-consuming scripting and material flow modeling with the application of physics simulation methods and physical models of the production system, which is faster and more intuitive. Another important point is that the VR technics allow the user interaction on the virtual system. Therefore the objects should behave and answer to the actions like in the real environment [Spi09]. The physics simulation methods enhance the VR technology in this regard.

2.3.5. Programming aspects

This chapter is not about the specific software solutions or programming languages, but more about the general aspects behind them. They are explained in a software-related context in this chapter, but it should not be forgotten that these aspects extend along the mechatronics engineering mindset. The information provided in this section is relevant to understand the software related aspects introduced in Chapter 7 “Implementation Concept.”

2.3.5.1. Object-orientation

Object-oriented design is a part of object-orientation in the development process and concerned with producing an object-oriented model of a system to implement the necessary requirements. This approach is used for the mechatronic engineering object concept in the MEP, and the realization of the software implementation is also achieved with an object-orientated language C#, as it is easier to implement the MEP concept with similar structures.

Object-orientation is a modeling and programming method using "objects" – data structures consisting of data fields and methods together with their interactions – to design data models, applications, and computer programs.

An object is an entity with a state and a set of operations that operate on that state. Object attributes represent the state. The activities included in the object can also provide services to other objects (clients) when some computation is required.

Objects are created by an object class definition, which is both a type specification and a template to create objects. It includes declarations of attributes and operations [Som07].

An object-oriented system consists of objects that interact with each other. Objects maintain their local state and provide operations on that. The classes define the objects and their interactions.

The general object-orientation-model has the following characteristics according to [Weu98]:

- States, parameters, inputs, and outputs are combined into sub-models (*encapsulation*).
- The internal behavior of a sub-model is defined as a combination of lower level sub-models or a set of equations (*part of the model hierarchy*).
- A structure of interconnected sub-models can be described without particular determining the computational causality (*non-causal modeling*).
- The instance creation of a sub-model is differentiated from its definition in a library (*abstraction*).
- A separation between externally accessible attributes (ports, parameters, states) and internal behavior is possible in the sub-model definition. Multiple versions for the internal behavior is supported (polymorphism).
- Sub-model definitions can be defined as an extension of a more general sub-model definition (kind of hierarchy in the sub-model library).

Object-oriented programming is a software design process using object-oriented programming languages, such as C#, Java, C++, etc. Such languages provide constructs for defining object classes and a run-time system to build objects.

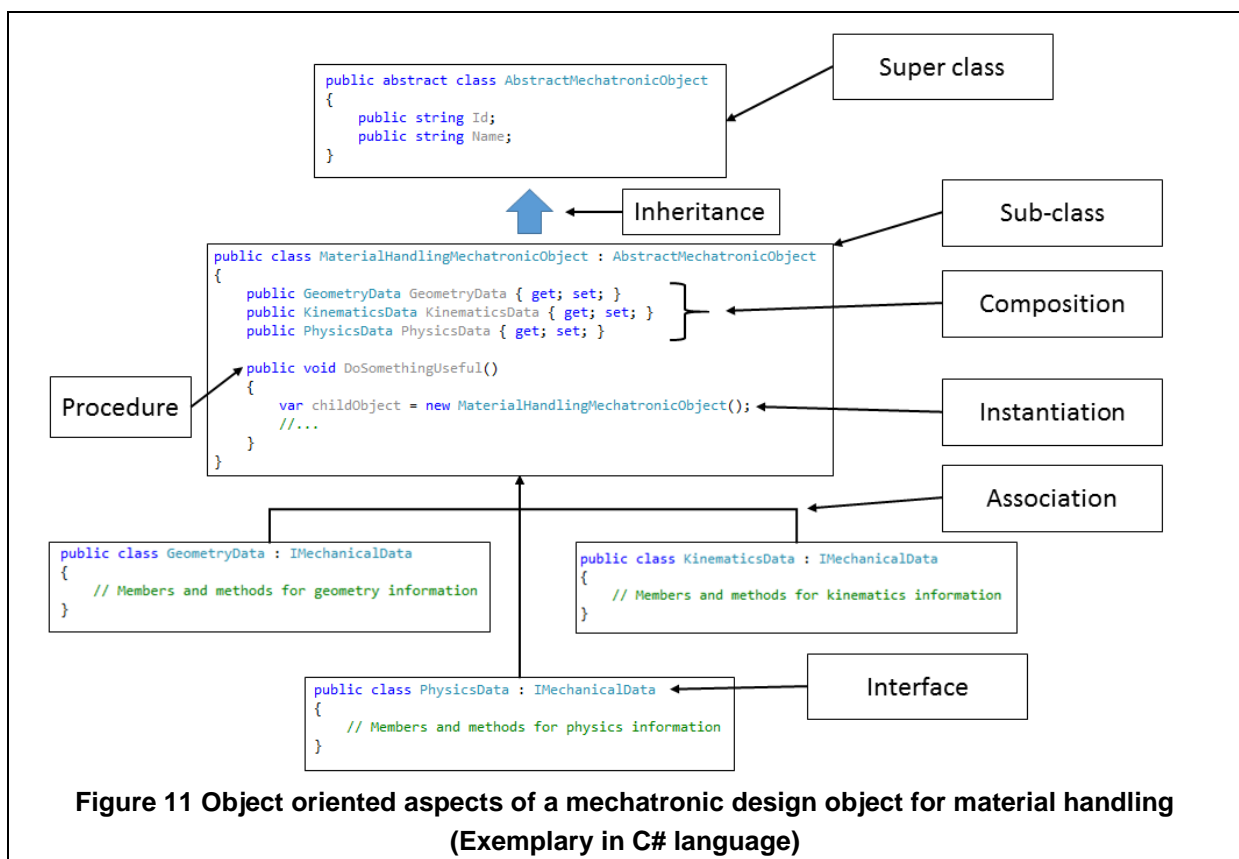


Figure 11 Object oriented aspects of a mechatronic design object for material handling (Exemplary in C# language)

Figure 11 shows how the object-oriented aspects can be used to describe a mechatronic design object for material handling, as explained in chapter 2.1.2. An object with the name *MaterialHandlingMechatronicObject* represents the primary element. It has a super class (“Inheritance”) named *AbstractMechatronicObject*, which defines the general properties of all type of mechatronic design objects, i.e. *Id* and *Name*. *MaterialHandlingMechatronicObject* contains members (“Composition”) *GeomeryData*, *KinematicsData*, and *PhysicsData*, which define the different information aspects of the object. They all implement the *IMechanicalData* interface, which means they all support some predefined structure. In this regard, it shows that they all agree to support the mechanical data members and methods. *MaterialHandlingMechatronicObject* has a method *DoSomethingUseful()*, which executes a procedure to generate or manipulate data. i.e. A child object is created as an instance in this method for further processing. Only mechanical data aspects are considered in this example, but other information types, defined in chapter 2.1.2, can be generated in almost the same manner.

2.3.5.2. Software Testing

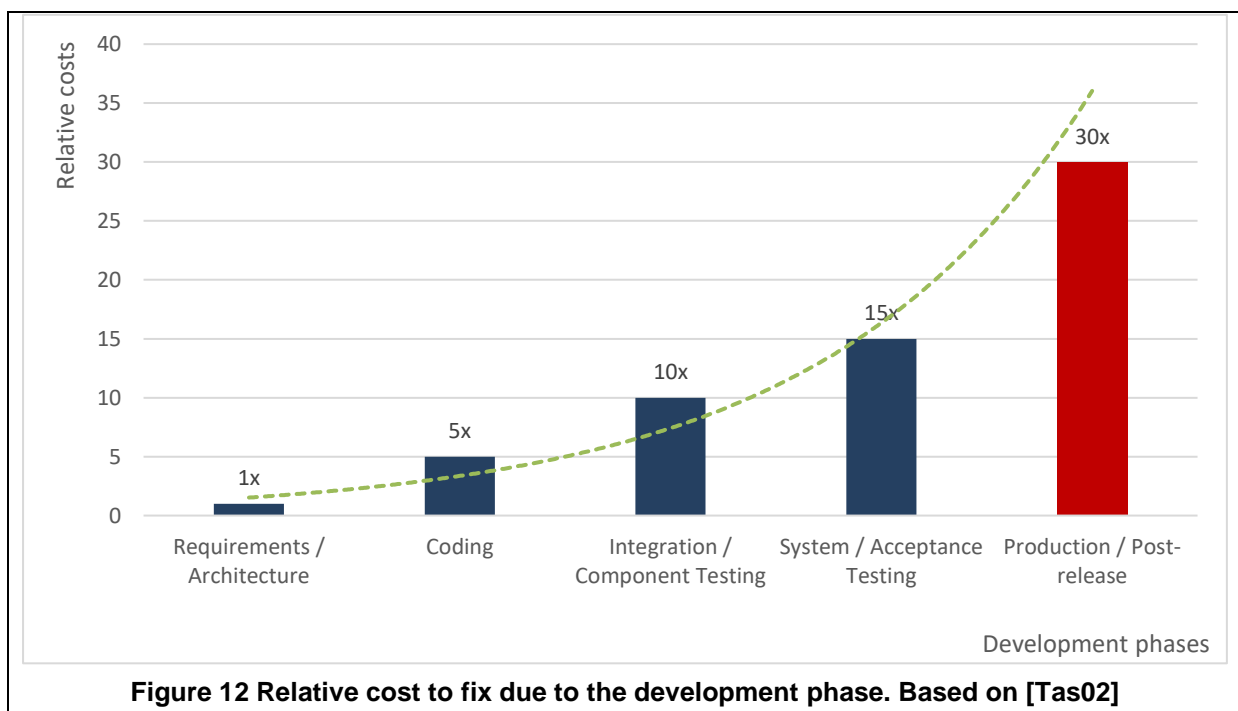
Modern production planning methods try to reduce iterations within engineering stages by integration of test phases. The primary goal of testing is to detect an engineering problem as early as possible so that it does not induce cost intensive corrections in the following phases [Bar091]. Testing activities include not only virtual commissioning as mentioned in 2.2.4 but also the software testing to ensure the validity of developed software systems.

A software test is a technical operation to validate one or more characteristics of a target software element or system. The action of executing one or more tests is called software testing. The medium of software testing could be in hardware and/or software. The specified procedures and an executable test suite define the testing environment. The aim of software testing is increasing the quality of the software and thereby reducing the maintenance and service costs. [Tas02] The subject of testing can be PLC code like in virtual commissioning or any other software component like the data exchange software libraries, which are generated during this work.

The [IEE90] defines following type of software tests:

- **Component testing:** Testing of individual hardware or software components or groups of related components.
- **Interface testing:** Testing conducted to evaluate whether systems or components pass data and control correctly to one another.
- **System testing:** Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.
- **Integration testing:** Testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them.
- **Unit testing:** Testing of individual hardware or software units or groups of related units.

Testing in the earlier phases of design brings significant advantages. It is better to detect defects and problems early in the development process before they are migrated to the next design stage. That way, the shipped product contains fewer problems and error correction costs less than if errors are discovered later in the process. The longer an error remains undetected, and the more additional code is written around the defected part, it results in digging out more layers of code to find out what the actual problem is. That multiplies the costs and time for maintenance [Tas02]. If a bug (software defect) is figured out in the final product, it also affects the work of the customer. A defect in software, which is used in safety critical fields, can also cause damage to property or human health. Experienced software developers know this phenomenon well so that practices like unit testing or test driven development are very common nowadays. Figure 12 shows an exemplary chart how the costs of bug fixing raise through the development phases.



To gather better quality, unit testing is also practiced in the implementation of data exchange software components, which will be introduced in chapter 7.2. Unit tests are low-level, focusing on a small part of the software system i.e. one unit test for one method. Unit tests also have the advantage that they are usually written by the developers themselves using some regular tools. Other than that, unit tests are expected to bring faster results than other kinds of tests; they can be run very frequently during programming. Programmers can execute unit tests after making any change to the code. With the help of test suites, it is also possible to automate the test execution, which enables a so-called “self-testing code” [Fow14].

2.4. Summary

The focus of this work lays on the layout planning of the material handling systems as a part of mechatronical engineering process, which has been examined in different steps.

A material handling system is a typical example of a mechatronics system as it demands the collaboration of different engineering fields. A mechatronics system/subsystem can be considered as a combination of several mechatronic objects. A data structure, which represents a real mechatronics object in a digital platform, is called a mechatronic design object.

The layout planning is an activity inside the mechatronical engineering process. MEP also contains other steps beginning with the product design and ending with the real commissioning. The information, which is used in one step, serves as the fundament for the next step, and will be enriched through the journey between different engineering software tools.

A wrong decision in the design or a defect in the control software should be prevented in the early phases of the MEP, as the costs of correction increase rapidly in each step. As a result, digital factory methods like the application of different simulations, usage of virtual environments and virtual commissioning are almost inevitable. However, it should not be forgotten that digital factory is a cost factor itself. A critical bottleneck is the data exchange between different engineering tools, which are specialized in a particular engineering field.

3. Literature Review

"Other people's stories may become part of your own, the foundation of it, the ground it goes on." – Ursula K. LeGuin in 'Gifts'

The content of this thesis requires an overall understanding of the MEP, and knowledge about information types used in the MEP, data exchange methods, physics-based simulation for virtual commissioning, and digital factory methods in material handling system planning. The literature, which provides the background of this thesis, can be divided into two categories: 1) About modeling and data exchange in the MEP, 2) About physics-based simulation (especially) regarding virtual commissioning applications.

3.1. Literature about modeling and data exchange in the mechatronical engineering process

This work can be seen as the continuation of the master thesis from the author [Yem11], which describes graphical and kinematical information in the MEP for automation systems. It presents the information types in relation to the engineering phases and discusses the need for data exchange solutions. The introduced concepts are validated by a prototypical implementation of a data exchange component, however, limited to the geometry information.

The doctoral thesis of Jens Kiefer [KIE07] from 2007 is an early but important work to understand the mechatronics concept in the production planning process on the use-case automotive body shop. He introduces reusable mechatronical planning objects for resource and process elements, which combine all relevant mechanical, electrical and control technical data together. He explains the usage of the (PPR: product, process, resource) data model based on the planning objects to validate and optimize real PLC programs and associated factory information systems within the virtual commissioning. He also mentions the software-related side of mechatronics planning process and the significance of data management and data exchange.

A major literature about the data exchange subject is the book "*Datenaustausch in der Anlagenplanung mit AutomationML*" (Eng. Data exchange in the facility planning with AutomationML) from different authors and Rainer Drath as the editor [Dra09]. This book gives detailed information about the interoperability between digital engineering tools for all steps of the engineering process in the plant design. It is arranged as a handbook and a decision aid for the "AutomationML" format. There are many modeling and implementation examples and guidelines provided in the book.

The doctoral work of Lorenz Hundt [Hun12] gives an overview of existing description concepts of mechatronical units and develops a model for the definition of mechatronical information objects. It includes not only the list of information types and interfaces but also different views on the information objects that are required for the engineering disciplines involved in the MEP.

Mr. Hundt examines the information flow between the various engineering phases to determine gaps within the data exchange of behavior models. He introduces meta models in IML (Intermediate Modelling Layer) format for defining an exchange format for behavior description of automation systems and shows a prototype implementation. He utilizes AutomationML format for the definition of production systems.

Malte Proesser [Pro14] provides an analysis of systems integration in the MEP considering different data modeling strategies like top-down modeling, bottom-up modeling and middle-in modeling. He does an elaborate literature review to determine the problems in the data exchange and data sharing concepts. He categorizes AutomationML format as a good example of top-down modeling. After that, he explains the middle-in modeling strategy and provides validation of his concepts in the native file format of *Tecnomatix ProcessDesigner* tool from Siemens.

The graduation work of Sebastian Rank [Ran11] investigates the usage of virtual reality in the context of material handling system engineering. He introduces how virtual reality and digital factory methods can be applied to material handling systems. This work explains the state of the art and economic and technical advantages of virtual reality for material handling systems in detail. He also presents a prototype implementation written in Visual Basic 6 to connect a layout planning tool *LPTool* and a virtual environment *VRed* through a data connection plugin *VRBridge*. Mr. Rank introduces an inter-process communication between these tools through Socket connection or Java/COM bridges, but does not interfere with data exchange between engineering tools from different engineering phases.

3.2. Literature about physics-based simulation regarding virtual commissioning applications

The doctoral thesis of Michael Spitzweg [Spi09] concentrates on the usage of physical simulation in the virtual commissioning of production and material handling systems. He introduces methods for geometrical, kinematical and physical modeling of production components. These models are used prototypically in realistic visualization and collision detection in a simulation environment named *Virtual Engineering Environment Extended* (*Ve³*). He utilizes VRML format for geometry information and proprietary XML-schema for physical information. However, data exchange of this information in the virtual commissioning toolchain is not a topic in his work.

Frédéric-Felix Lacour resumes and widens the concepts, which are introduced by Michael Spitzweg, in his doctoral thesis [Lac12]. He defines his aim is to extend the previous methods for physics-based virtual commissioning for the integration of material handling system components. He introduces concepts for optimizing the geometrical models to reduce the computational expense and improve the performance so that the number of components in the physical simulation can be increased. He uses COLLADA format for geometry information and proprietary XML-schema for physical information. He mentions a possible usage of

AutomationML format in the future but does not engage in the data exchange subject. He uses the simulation environment *Virtual Engineering (Ve)* for his prototypical implementation, which is most likely an enhanced version of *Ve*³ with more modules and functions.

The doctoral thesis of Anton Strahilov [Str15] presents a digital mechatronic validation of automated assembly systems, considering the physical behavior of the system parts and components. His concept is divided into three steps: preparation of the simulation model, physics-based mechanical validation, and physics-based virtual commissioning. He generates the necessary data with the software tool CATIA V5 and SiX (Simulation for X)-Macros to be used in physics simulation tool V-Rep.

3.3. Summary

As the sources in the first section 3.1 gives the necessary background for the modeling of material handling systems and data exchange methods, the literature in section 3.2 provides the knowledge about the physical model and physics-based simulation without containing data exchange subject. Table 1 illustrates an overview of the subjects, which are mentioned in these sources and relevant for this thesis. A combination of physical modeling and data exchange for material handling systems with a focus on layout planning phase in the MEP is a subject, which is pointing a gap in the research. This doctoral thesis tries to fill this gap together with an implementation of a software solution to be applied in tarakos engineering tools as validation.

Table 1 Relevant subjects and authors mentioned in the literature review

| | Modeling with mechatronics concept | | | | Data exchange | Virtual reality applications |
|------------------------|------------------------------------|-------------|----------|---------------------------|---------------|------------------------------|
| | Geometrical | Kinematical | Physical | Material handling systems | | |
| Kiefer, Jens | X | X | | | X | |
| Drath, Rainer (ed.) | X | X | | | X | |
| Hundt, Lorenz | | | | X | X | |
| Proesser, Malte | | | | | X | |
| Rank, Sebastian | | | | X | | X |
| Spitzweg, Michael | X | X | X | X | | X |
| Lacour, Frédéric-Felix | X | X | X | X | | |
| Strahilov, Anton | X | X | X | | | |

4. Requirements Analysis

“If we can really understand the problem, the answer will come out of it, because the answer is not separate from the problem.” - Jiddu Krishnamurti in ‘Life Ahead: On Learning and the Search for Meaning’

The requirements for realizing a data exchange solution for a layout planning tool are analyzed in this chapter. There are overall two sections: Methodological requirements, which define the preconditions for the model creation and data exchange implementation and technical requirements, which define the essential functions to realize a data exchange solution.

4.1. Methodological requirements

Modern production systems are good examples of efficient usage of engineering tools from different branches like mechanics, electronics, and informatics. This situation results in a growing complexity and a challenge for the companies to combine all these different disciplines into a structured engineering process. A production system goes through various development phases from product design to real commissioning. A structural or functional error in the system design can be solved only with significant efforts and costs in the later planning phases or after commissioning. For this reason, it is important to use digital factory methods in the concept design and validation of functionalities as it provides an early feedback.

An essential requirement for the digital factory is the existence of geometrical and physical models. It is easier to automate the creation of the simulation if the needed information is already supplied in digital form. The topological structure of the plant, process definitions, and transport strategies are necessary information types for material handling systems. Physical aspects including kinematic and dynamic processes as well as external stimuli should be taken into account for physics-based simulation. Therefore, kinematic constraints such as joints, joint types, links, and attachments should be provided. Such information about individual parts of the production system should also be known: Geometry, weight, center of the gravity, moment of inertia and friction coefficient [Spi09]. A categorization of the components relating to their kinematic properties –as dynamic or static bodies- is required, too.

The main application of this work is layout planning for material handling systems in the engineering tool chain. Material handling is a part of the productive system, and layout planning cannot be viewed isolated from the other MEP phases, as it is also a part of the factory planning itself. There are also many different planning tools in use, which are compatible with other tools only through individual and special interfaces -if at all-. The lack of a standardized data exchange is a main problem in the MEP. Another problem is the exchange of the data itself. Because of encoding and decoding operations during data transport, there may be different interpretations of the data, which causes planning defects [Ran11]. It is a usual practice to keep information in tables or spreadsheets, which often means humans do

the data transfer, usually by hand. It is also demanded to have a standard information structure and synchronization between engineering tools to increase the automation of model and simulation creation. Realization of data exchange interfaces between engineering tools/phases is, therefore, another important methodical requirement.

4.2. Technical requirements

Not only the existence of the component models but also the preparation of them for the data exchange, is an essential requirement. That means the models should be defined in a neutral standardized format. Mapping of component entities, functional elements, process definitions and their attributes between proprietary and standard formats is the first step in enhancing the interoperability in the toolchain.

The models, which are prepared in a standard neutral format, should be used in a physics-based virtual commissioning system. Behavior models of some components are defined in complex algebraic or differential equations. These equations are solved by specialized physics solvers. Complex multidisciplinary systems require the interconnection of several solvers as they cannot be handled in a single software system. Co-simulation is a method to resolve such complexity as explained in chapter 2.3.3. Therefore, the standard neutral format should be able to contain and/or reference complex behavior models.

After creating the models in a standard neutral format, the second step is the implementation of software components, which execute the data exchange action. These components should fulfill two main functional requirements in the toolchain: 1) Describing and exporting the proprietary data model as the standard neutral format, 2) Importing a file in the standard neutral format from a different type of source.

For a successful export function, the quality of the exported information should be considered as an important factor. Not only the position and the geometry of the object but also the hierarchical structure and workflow should be transferred. All exported components must have a unique identifier for the identification of objects. The identifiers are not allowed to be changed in the further phases of the toolchain [DrSc10]. The tool itself should have an inner mapping of the objects and their identifiers for the recognition of the own objects after the export. The versioning of the file should also be supplied. The Meta information (i.e. name of the company/project, contact information like email, telephone number, etc.) should be included as a header element to ease the communication between the vendors. If the receiver cannot parse the file, they should be able to find a contact information.

Although a 3D layout planning tool is relatively at the beginning of the engineering toolchain, it is sometimes required to import files, especially the ones originated from 3D content design tools. Three types of importing tasks can be defined:

1. Importing the own created files (i.e. from taraVRBuilder to taraVRControl),
2. Importing a file which is created by own tool but changed from another tool afterward (i.e. exporting from taraVRBuilder, further editing by an external tool and importing to taraVRControl),
3. Importing a file from an entire foreign origin.

It can be guessed that importing a proprietary file is the easiest task among these. The scene can be created with the known hierarchical structure, and the objects can be established via internal mapping of identifications. The possible changes in the attributes could be applied by known internal software methods.

The second importing type is more challenging than the first one. It is possible that some unknown attributes or properties are additionally on the scene, and they should not be overwritten during the further work in the tool.

The third type of import is the most difficult one. However, there is still some information in the file, which can be recognized. In principle, information can be evaluated syntactical and semantical if it is defined in a standard form. Standardization for data exchange means that the common data models and concepts for each participating engineering tools are described in a neutral form and all participating vendors agree to support this neutral form [Dra13]. The modeling effort should also contribute to creating such a data exchange standard for material handling systems.

4.3. Summary

A data exchange solution for material handling components in a layout planning software tool is the main requirement. The component models will be used in virtual commissioning applications, which also include rigid-body physics simulation. Therefore, a data exchange concept is needed, which covers these aspects:

- **Visual aspects:** Geometry, position, topology of the component models. Animation of rigid bodies according to kinematical and physical properties can also be mentioned.
- **Control aspects:** Control behavior of a component and process parameters including input/output und state variables.
- **Physics aspects:** Description of energy and material transformations during the processes

The standard format should also support the definition of complex behavior models and co-simulation bindings.

Implementation of software components, which can export/import the data structures involving aspects mentioned above is the second requirement. It should be considered that these

components should support different data exchange scenarios. Therefore, the standardization effort is a part of the task to provide a common solution.

5. Data Exchange Formats

“The limits of my language mean the limits of my world” – Ludwig Wittgenstein in ‘Tractatus Logico-Philosophicus’

Data exchange is a task within the data integration concept. As many software tools participate in the MEP, it is needed to share data among multiple software tools from different vendors, and to integrate data in a flexible and efficient way. The goal of the data exchange is to create an appropriate instance of a “target schema” from a given “source schema” by using a semantic mapping between them [Doa12].

A format is a structure, which defines the layout for the organization of the data [Tec16]. A widely-used file format for the data exchange is Extensible Markup Language, abbreviated XML. XML is a markup language for documents containing structured information. A markup language is used to identify structures in a document, which means how the content of the document should be interpreted [Wal98]. XML is widely used as a data exchange structure for both database and document sources. It brings standardization into data formats and corresponding data exchange interfaces and tools [Doa12]. The structure, content, and semantics of XML documents are defined by the schema language XSD (XML Schema Definition) [Wor00]. It also eases the implementation of a software tool support for the data model, as automatic generation of classes from XSD is possible with existing tools like JAXB for Java language or XML Schema Definition Tool for .Net Framework [Gla16] [Mic162].

The native format of the proprietary software toolkit can be represented in a text-based structure or an XML-structure. Also, there are two XML-based data exchange formats and one model exchange format using XML format to define variables, which are utilized within this work. This chapter explains why these formats are selected and gives the useful overview about them.

5.1. Selection of the data exchange formats

5.1.1. Selection criteria

As defined in chapter 4.3, the data exchange concept should cover several aspects like visual aspects, control aspects and physics aspects. There are several description languages, which cover one of these aspects, but the solution demands a combination of them. Therefore, a container structure is required to envelop different formats specialized in their fields. The information types include geometrical, kinematical, physical, topological information as well as behavior description, process parameters, and other production relevant data. The selected data exchange format should be lossless, that means it should contain structures to represent all these mentioned information types related to the MEP.

If the selected format is open-source and cost-free (royalty-free), that would make the format acceptable for many software tool vendors. It is an enormous advantage for the

standardization, as a consensus between different actors in the market is required. A well-structured documentation is also a factor, which could accelerate the standardization process.

The vendor-independency is also indispensable to be accepted by different vendors. The format should not depend on only one company or one product chain. The reliability and availability should not be endangered if the firm goes out of the market for some reason and is not able to supply and develop the format anymore. Also, small, and medium-sized enterprises are not very willingly to step in if they should be dependent on a monopoly-like supplier.

The format would rather be easily human understandable, so mistakes -if there are any- could be easily found and corrected. This property eases the implementation of a software component, which parses and edits the format.

5.1.2. Overview of data exchange formats

A brief introduction to the main data exchange formats, which could be applied to material handling systems, is provided. The relevant formats are STEP, JT, CMSD, SysML(XMI), and AML. After the introduction, their advantages and disadvantages will be discussed on the selection criteria to decide.

STEP (Standard for the Exchange of Product Model Data): STEP is a data exchange format, which became an ISO standard in 1993 (ISO 10303). STEP includes the following parts: description methods, integrated resources, application protocols, abstract test suites, implementation methods and conformance testing. An information modeling language called EXPRESS is used to manage the format operations. STEP uses application protocols to describe the product information for one or more applications [Bha00].

STEP format includes information of the geometric shape, topology, features, tolerance specifications, material properties, and some others which are necessary to define a product for design, analysis, manufacture, test, inspection, and product support [Fow95].

JT (Jupiter Tessellation): The JT format is a system-neutral, industry focused, lightweight 3D data format. It is developed by Siemens PLM Software for 3D product definition involving collaboration, validation, visualization, and data exchange. The JT Open Program is a C++ library for read/write access. The JT format defines a scene graph with CAD specific node and attributes support. JT can store facet information (triangles), BREP definitions, product, and manufacturing information (PMI), Meta information and representation configurations [Sie10].

Core Manufacturing Simulation Data (CMSD): is a data exchange format from Simulation Interoperability Standards Organization (SISO), which is developed by Researchers at the National Institute of Standards and Technology (NIST) in collaboration with industrial partners. This format focuses on data exchange between simulation systems (especially Discrete Event Simulation) and other manufacturing applications [Bar16].

The purposes of this format are [Cor12]:

- enabling data exchange between simulation applications and other software applications,
- supporting the construction of manufacturing simulators,
- supporting the testing and evaluation of manufacturing software,
- enhancing interoperability between manufacturing software applications.

The CMSD structure is defined as a suite of interrelated collections of information modeled as UML classes within UML packages. The packages are presented visually as a series of UML class and package diagrams. The CMSD format consists of six major UML packages: Layout, Support package, Resource information package, Production planning package, Production operations package, Part information [Bar16].

Object Management Group Systems Modeling Language (SysML): SysML is a general-purpose modeling language for systems engineering applications. SysML supports the specification, analysis, design, verification, and validation of complex systems, which include hardware, software, information, processes, personnel, and facilities. It is developed and maintained by Object Management Group, Inc. (OMG), which is an open membership, not-for-profit computer industry standards consortium. SysML uses the OMG **XML Metadata Interchange (XMI)** format to exchange modeling data between tools [Obj12].

SysML consists of structure diagrams, including the package diagram, block definition diagram, internal block diagram, and parametric diagram. Following clauses are part of the language [Obj12]:

- Model Elements extend the UML 2 standard to provide capabilities for model management.
- Blocks describe system decomposition and interconnection. They define different types of system properties i.e. value properties with optional units of measure.
- Ports and Flows provide the semantics to define the interaction of blocks and parts through ports and the flow of items across connectors.
- Constraint Blocks define the usage of blocks on parametric diagrams. Parametric diagrams can be described as a network of constraints on system properties to support engineering analysis, i.e. performance, reliability, and mass properties analysis.

Automation Markup Language (AutomationML-AML): is a free, open, and neutral format for storing automation data. The primary goal of the format is serving as an exchange format in a heterogeneous tool environment. AutomationML is a collection of three (free and open) XML-based languages, which are domain-specific to the different aspects of automation resources [AML13]:

- CAEX (Version 2.15) is for storing plant automation data and topology
- COLLADA (Version 1.5&1.4.1) is for storing geometrical, kinematical, and physical information
- PLCOpenXML (Version 2.0&2.0.1) is for storing control and behavior data.

The format can be expanded to any valid XML-based format. Object-oriented aspects are used for defining the relationships between plant components.

Table 2 Comparison of data exchange formats

| Format \ Criteria | STEP | JT | CMSD | SysML(XMI) | AML |
|-------------------------------|---------|-------------|-----------|------------|------------------|
| Graphical content | Yes | Yes | No | Possible | Yes |
| Kinematical content | Limited | No | No | Possible | Yes |
| Physical content | No | No | No | Possible | Yes (Rigid body) |
| Behavioral/functional content | No | No | Partially | Yes | Limited |
| Lossless | No | No | No | Yes | Yes |
| Vendor-independent | Yes | No | Yes | Yes | Yes |
| Royalty-free | No | Conditional | Yes | Yes | Yes |
| Human-understandable | No | Limited | Yes | Yes | Yes |

There are some other XML-based formats defined in [Ver11], which are used in the context of automation. Some examples are BatchML, CANOpenXML, FDCML, FDI, GSDML, PROLIST ... etc. They are not included in this comparison, as they have distant fields of application.

5.1.3. Discussion and decision

As seen from the Table 2, STEP and JT do not fulfill most of the criteria. They are not easily human-understandable, not royalty-free (Siemens gives anyone copyright permission for JT but with some conditions) and does not support kinematics and physics aspects, which is a deficit for physics-based simulation applications. JT is also not vendor-independent, as it is a trademark of Siemens AG.

CMSSD is a data exchange format useful for handling discrete event simulation data, for example, between data processing applications and simulation applications. However, there is a lack of evidence for supporting a wider range of engineering tools and 3D applications [Sko11].

SysML(XMI) is used for tracing and relating domain-specific models, but do not cover necessarily the information contained in them. It is a general modeling language, which does not provide the semantic context. It has been used for reverse engineering and simulation applications of mechatronic systems [Bar15]. Although it is possible to define geometrical content with this format, there would be no tool support to visualize. However, visualization of the 3D content is a mandatory aspect of a 3D layout planning application, which results in the decline of the SysML(XMI) format.

AML format serves as a container for mature data formats in their fields and can be used for connecting the information inside the models from different domains. COLLADA format, which supports geometry, kinematics, and rigid body physics aspects, covers some of the essential requirements as a part of the AML format. A deficit for AML is the support of physical behavior of the components is limited to the control behavior through PLCOpenXML. Nevertheless, complex differential and algebraic equation systems might also be necessary for some applications. The support of co-simulation bindings is the second issue for the AML format.

Fortunately, it is possible to extend the AML format with any XML-based standard format. MathML format, which can define continuous functions, can be utilized to expand the AutomationML format for complex control behavior [Ber10]. Another possibility is the Functional Mock-up Interface (FMI) standard, which could be used to define complex behavior and allow co-simulation connection. FMI can be referenced in AML as an external reference for physics solvers like COLLADA for geometry and PLCOpenXML for control logic [Gra13].

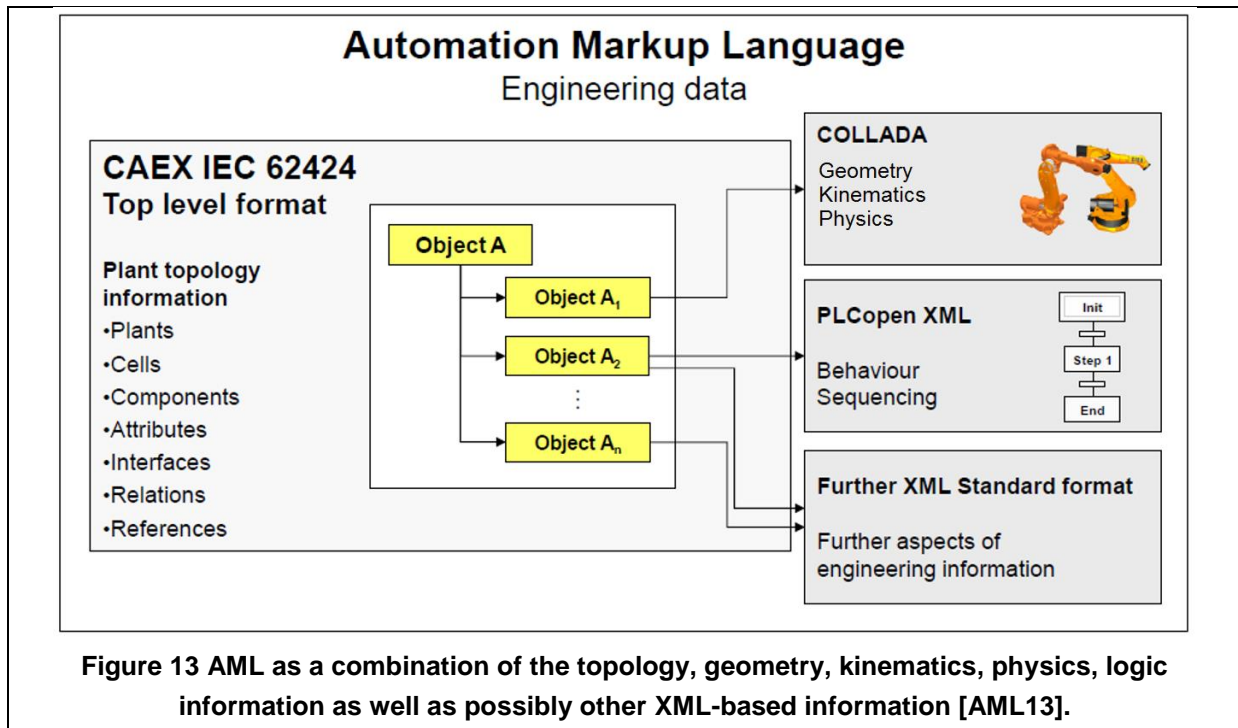
As a result, the decision is for the AML format extended with FMI external reference. Further details will be given in the next section.

5.2. Overview of the selected data exchange formats

5.2.1. AutomationML

Automation Markup Language (AutomationML/AML) is an XML-based, open, neutral data format standard for the storage and exchange of plant engineering information. The aim of

AML is to interconnect engineering tools with different specializations in the heterogeneous tool landscape. AML describes plant components as objects encapsulating different aspects like topology, geometry, kinematics, physics, and logic (see Figure 13) [Dra08].



5.2.1.1. Architecture

The AML format uses the object-orientation concept. All engineering parameters are modeled as objects or included as attributes of an object. A class is a predefined object type. An AutomationML object can contain other child objects or can be a part of a bigger composition, which results in a hierarchical object tree. CAEX-Format is used in AML for modeling and storage of such object trees.

Data modeling with CAEX is carried out based on an XML schema, which is in the form of the XSD file "CAEX_ClassModel.xsd" and defines all required data types. The CAEX modules in the schema specify the syntactical construction of classes, interfaces, attributes, instances, and all other elements. The CAEX schema acts as a "Blueprint" for CAEX documents and allows an automatic check of conformity if a present CAEX document is valid [DrSc10].

In Figure 14, the fundamental CAEX objects can be seen in a tree-structure. The <CAEXFile> element describes the root element of the CAEX document. The role class elements of <RoleClassLib> describe an abstract functionality without a concrete implementation. An example of the role is "Resource" or "Conveyor". The <SystemUnitClassLib> with its classes from type <SystemUnitClass> defines a library with predefined production system entries. <InstanceHierarchy> is the root node of the object tree, which contains an arbitrarily nested hierarchy from objects of type <InternalElement> (e.g. a specific conveyor in the plant). <InternalElement> objects can inherit a role as role requirement or supported role. The

relationships between <InternalElement> objects can be created with <InterfaceClass> objects from the <InterfaceClassLib>. Details about these libraries will be given in the following Chapter 5.2.1.2.

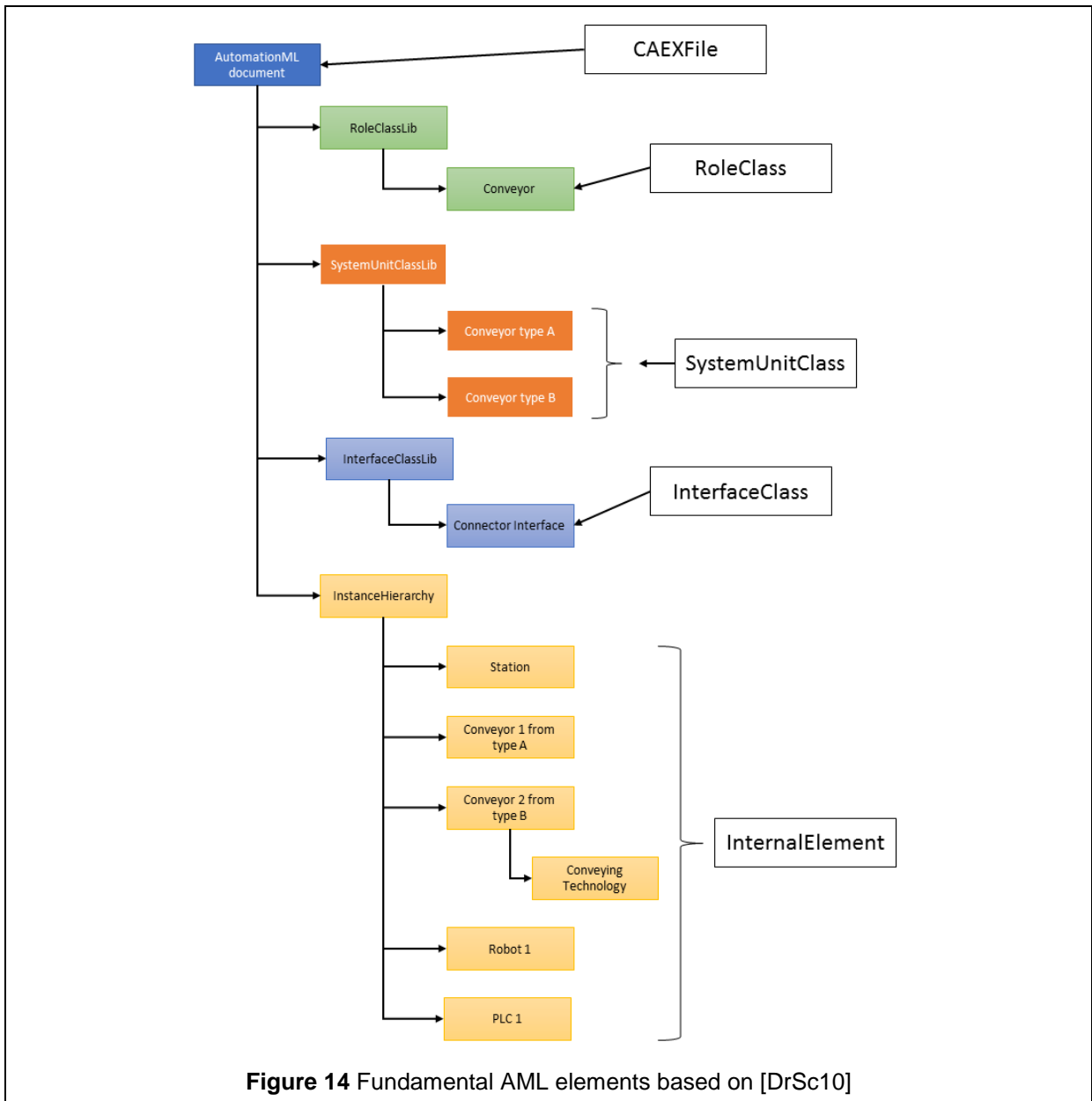


Figure 14 Fundamental AML elements based on [DrSc10]

The links between interfaces are described as part of the object hierarchy (e.g. links between connection points of conveyors). The position of an object in relation to other objects is defined by the <Frame> attribute, which contains the translation and rotation in x/y/z coordinates relative to the parent element. Geometry information can be added into the <InternalElement> as an external COLLADA document reference [AML13].

5.2.1.2. Base libraries

Instance hierarchies, which consists of AML object instances, store topology data of actual projects and are therefore the core of AML. However, there are more semantic levels in AML

such as base libraries with base classes, which are needed for the modeling of basic AML concepts [AML13]. They allow the reuse of class models. There are also relations within the class libraries, for example through inheritance or aggregation. The CAEX library concept distinguishes three types of library: the <SystemUnitClassLib>, <RoleClassLib> and the <InterfaceClassLib> with their respective classes. CAEX can contain as many of these libraries, where each class can be represented in a hierarchical structure within their library, for example, a user-defined tree structure with the father-child relationship between nodes. The father-child relationship between classes has, however, no further semantics. Classes of the same type can inherit therefore from each other and make class hierarchies that way – across libraries or CAEX documents. The various library types have the following meaning [DrSc10]:

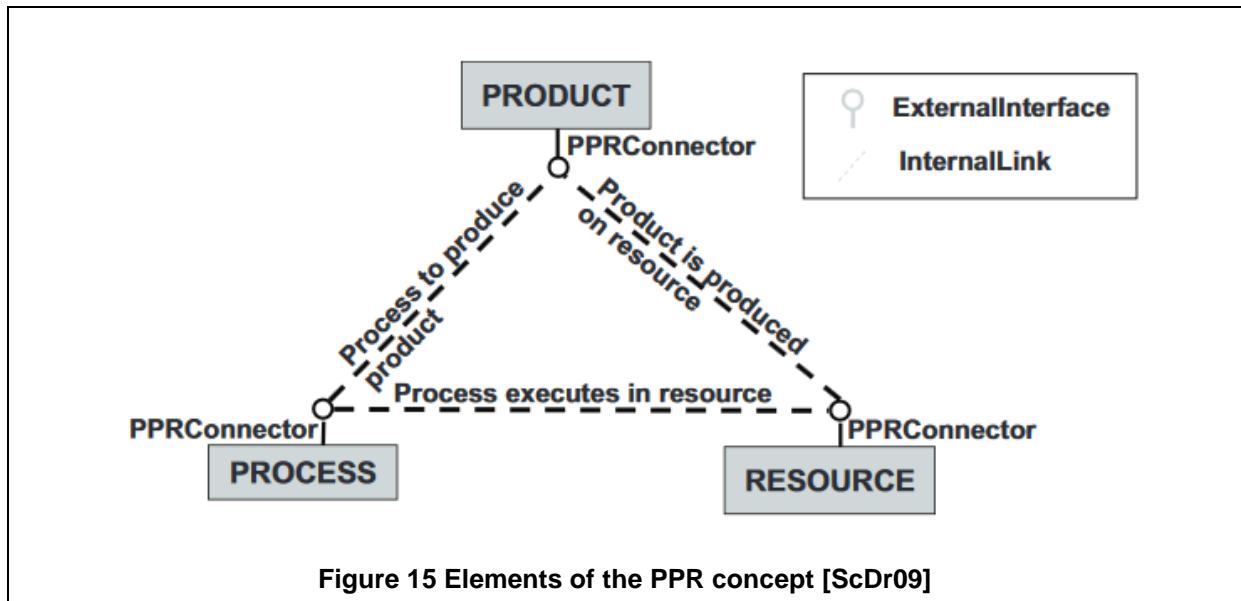
1. Classes of type <SystemUnitClass> describe concrete physical or logical system objects or their combination (called units). A unit can be a specific conveyor system, robot, or a valve from a vendor. They describe not only the object but also its technical realization including of its internal architecture, which includes attributes, interfaces, nested internal elements, and connections between them. An internal element can contain other internal elements - this allows the description of any complex object tree hierarchies. The association to an abstract <RoleClass> defines the role and function of <SystemUnitClass> or its instances. <SystemUnit> classes are grouped together in libraries of type <SystemUnitClassLib>.
2. A class of type <RoleClass> also describes physical or logical system objects, but at an abstract level and regardless of the actual technical realization. Roles are the semantic counterpart of the system units. While using <SystemUnitClass> specific technical solutions are shown, on the contrary, abstract roles are used to demonstrate the diversity of technical realizations and depict only the basic functionality of components. Roles can be defined as wildcards in the plant design and support an implementation-independent planning. Examples of roles include resource, robot, or conveyor. Role classes are summarized in libraries of type <RoleClassLib>. Each AML role must be associated directly or indirectly with a standard AML role, which gives information on the meaning of the object in the data exchange.
3. A class of type <InterfaceClass> describes a shared boundary, across which two separate components can exchange information, i.e. a flange, a signal, or a product node. Interfaces are used to make relations between objects. They are summarized in libraries of type <InterfaceClassLib>.

AML also defines extended concepts for the modeling of specific engineering aspects by using these basic structures. Important concepts for this work are AML Port concept and Process-Product-Resource concept [AML13]:

AML Port: AML-Port is used for grouping a number of interfaces. An AML object can contain a port component, which describes complex interfaces of the object. The interfaces are used to define connections to other objects. Example: “*MaterialHandlingConnectionPoint*” element,

which has an interface with the name of “*MaterialHandlingConnectionPointInterface*”. This interface defines the linking of two material handling components in the material flow. More details about material handling components will be provided in Chapter 6.4.

Product-Process-Resource: The need to describe all planning related data involved in the production process induced a threefold division into the process, product, and resource data. The relationship between these aspects is: A product is processed with resources, which in turn run processes (see Figure 15) [ScDr09].



A *resource* is an entity, which is engaged in the production. These include plants, robots, tools, machines, devices, software as well as their status, and possible messages. Resources can be understood both as the hardware components of a production system, for example, machines or conveyor systems, or also as the associated software systems such as, for instance, process visualization.

A *product* can be understood as a partial, intermediate, or final product or raw materials in the EP. An example of this is car parts like body, wheels, transmissions, engine, etc. Products include testing and test results, as well as product data and corresponding documentation.

A *process* represents the engineering actions on a product, which are performed by resources. This data includes process parameters, the process flow, and process planning.

Product-Process-Resource concept can be applied easily to material handling systems. Resources such as conveyors, turntables, loading stations and discharging stations, which are connected to each other, can transport goods from one place to another. Resource: Conveyor, Product: Good, Process: Transporting [DrSc10].

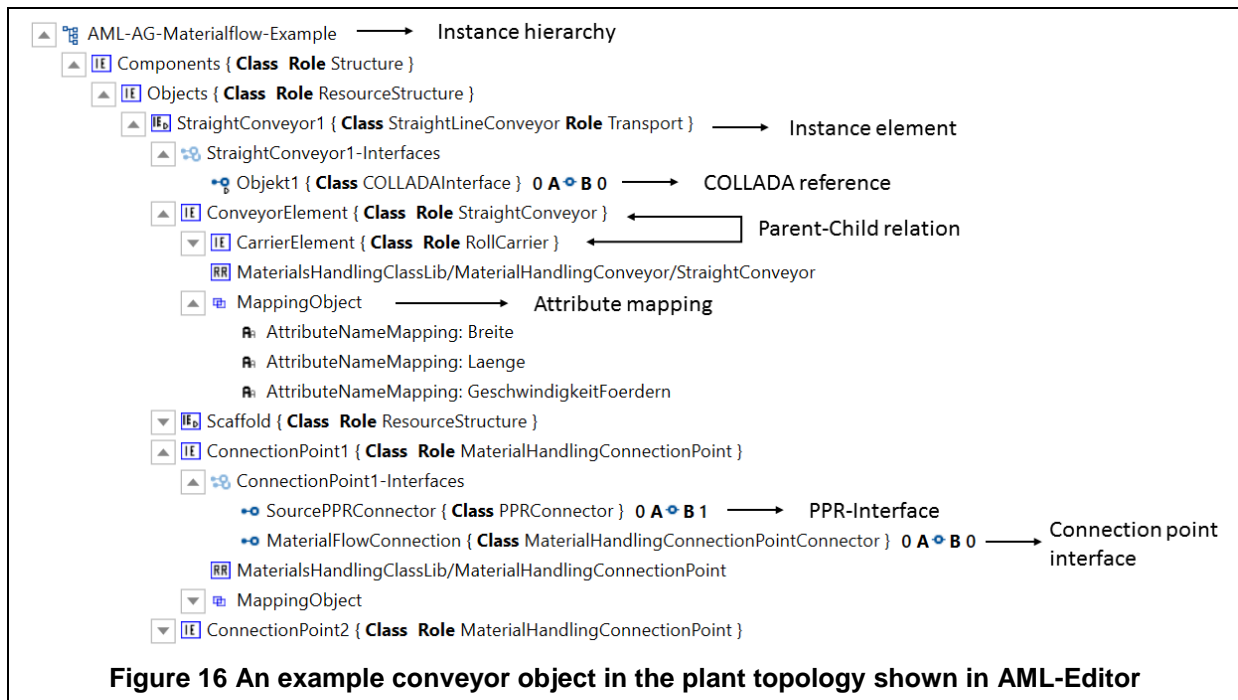


Figure 16, which is a screenshot from AML-Editor³, introduces the above-explained fundamental concepts on an example conveyor object in the plant topology. An internal element “*StraightConveyor1*” is created from the “*StraightLineConveyor*” system unit class, which is assigned to the role “*Transport*”. This element has a COLLADA reference “*Objekt1*”. The child element “*ConveyorElement*” is assigned to the role “*StraightConveyor*” and a <AttributeNameMappingObject> defines which attribute of the internal element corresponds to which attribute of the role class. The child element “*ConnectionPoint1*” contains two interfaces, one for the connection with the other conveyor elements and one for the PPR-concept.

5.2.1.3. Data exchange with AutomationML and COLLADA

The most important issue during the data exchange between two engineering tools is the correct transmission of the data structures from the source tool into the target tool. It must be possible to find a matching data object in the target tool for each transferred data object from the source tool. When engineering tools are coupled with AML, each system must have a corresponding exporter or importer, which reflects its specific data objects on the standardized AML objects. [Hun13]. The syntax and semantics are split from each other in CAEX structure. CAEX prescribes no object model; it provides mechanisms for modeling arbitrary user-defined data models. These are syntactically neutralized and some of the fundamental concepts has been explained above. On the other side, the semantics is user-defined through the content

³ AML-Editor is a software to visualize AML documents. It is provided by AutomationML e.V. and distributed under the AutomationML Public License, Version 2.1.

of role classes, system unit classes and instance elements. In this way, AML can handle the semantic diversity for a wide range of engineering applications [Dra13].

The first step of the data exchange with AML is the preparation of the internal proprietary data model in conformity with the AML architecture. In other words, semantics should be prepared using the AML syntax. Then comes the next phase, which consists of these four steps [Dra13]:

1. The exporter reads the information from the internal data model of the source software tool.
2. The exporter transforms the information into the exchange format.
3. The importer of the target software reads the information from the exchange format.
4. The importer transforms the information into the data model of the target software.

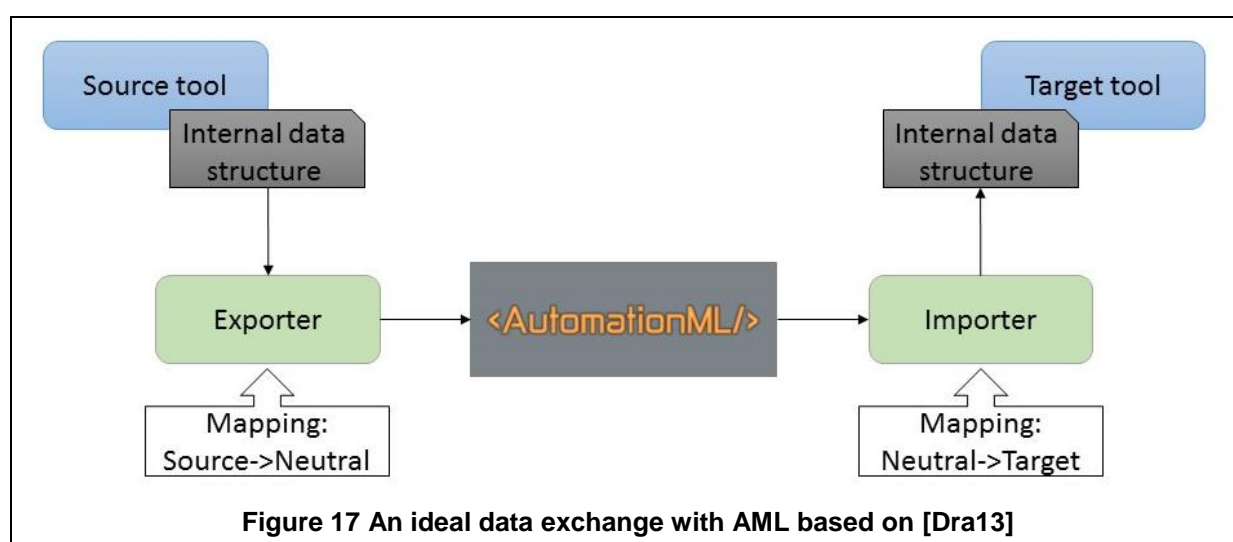


Figure 17 An ideal data exchange with AML based on [Dra13]

Although exporting the proprietary internal model into AML gives no guarantee that the target tool can interpret the model description, the data model is defined for the first time in a neutral format and externally visible. The following information is recognizable through components defined in standard:

1. Metadata of the exporter such as name, version, creation date, etc. (<WriterHeader> element)
2. Topology (<InstanceHierarchy> object)
3. Position (*Frame* attribute)
4. Geometry (external interface for COLLADA reference)
5. Connection points of system components (<InternalLink> object)

The semantic context, which is not defined in the standard, should be transformed using mapping strategies. The data exchange with AML can be carried out using different mapping strategies like mapping according to roles or mapping according to system unit classes. The mapping task of the importer is usually more complex than the exporters since the importer must find the matching objects/attributes in its data structure [Hun13]. It is possible to ease the semantic mapping through the systematic standardization of the models as described in

[Dra13], or using existing widely accepted standards to describe the object models. The correct way of handling the standardization problem can vary for project/software requirements.

5.2.2. COLLADA

An AML internal element object can reference a COLLADA-file to make an association to geometry, kinematics and rigid body physics. For this purpose, the standard AML-Interface “*COLLADAInterface*” is defined, which is derived from “*ExternalDataConnector*” base interface class [Aut13].

The geometrical representation and the kinematics information, which describes the physical connections of 3D solids and the dependencies among objects, can be stored using the file format “COLLADA” (COLLABorative Design Activity). Additionally, the COLLADA format includes the coupling of geometry, kinematics, and physics information. COLLADA defines a royalty-free, open, XML-based schema language to allow 3D authoring applications to exchange digital assets without loss of information. The primary goal of the format is to combine the software tools with 3D content into tool chains. [Bar08]. In Figure 18, an example COLLADA document from taraVRBuilder-Export can be seen.



Documents, which contain XML elements that are valid according to the COLLADA schema, are called COLLADA instance documents. The file extension of these documents is *.dae* (an acronym for **D**igital **A**sset **E**xchange). The COLLADA document also has a hierarchy of parent-child relationship like in CAEX. The parent element contains the child element, which results in a composition of information.

Other elements can be referenced by document elements using an addressing scheme. It is also possible to reference an object from an external COLLADA document. There are two

addressing schemes in COLLADA: Uniform Resource Identifier (URI) and Scoped Identifier (SID) [Bar08] [Arna06]:

- 1) **URI addressing:** COLLADA elements, which have an *id* attribute, can be referenced with this type of addressing. Mostly used in *url* and *source* attributes. The identifier begins with a hash character (#).
- 2) **SID addressing:** COLLADA elements, which have a *sid* attribute, can be referenced with SID addressing. Mostly used in *target* and *ref* attributes, <SIDREF> and <SIDREF_array> elements. There are also some other elements or attributes which reference to *sid* using this type of addressing.

For more information about addressing, *Chapter 3/Address syntax* from COLLADA specification can be read [Bar08].

The root of the COLLADA document is the <COLLADA> element. One <asset> element must exist as a child element. The other child elements are optional; there can be one or more library elements, zero or one <scene> element and zero or more <extra> elements. The <COLLADA> element contains both child elements and attributes. The order of the elements (sequence) must be respected.

COLLADA elements can be categorized into three main groups: 1) Assets, 2) Element Libraries, 3) Techniques, Profiles, and Extras [Bar08] [Arna06]:

- 1) **Asset:** The <asset> element contains the Meta information for a COLLADA document or element. The root element <COLLADA> needs at least one <asset> element to provide the creation and modification time for the document. Assets are hierarchical, and can be found as a child element of another element. It has no attributes, but many possible child elements.
- 2) **Element libraries:** A COLLADA Document is organized as a sequence of libraries, and can contain several library elements. Library elements can have one optional <asset> element and zero or more <extra> elements. Each library element contains only one type of child element, whose content is in accordance with the type of the library. For example, <library_geometries> element contains <geometry> elements. More information about the content can be found in the following chapters.
- 3) **Techniques, profiles, extras:** COLLADA format enables several representations of the same piece of content under one element to create a flexibility between various platforms or programs. Each piece of alternative content is represented by an element of <technique_common> or <technique>. The platform or program is specified with the *profile* attribute. Each technique is matched with an associated profile. The profile and the parent element in the instance document define the context for <technique>. The <technique> or <technique_common> element may also contain any well-formed XML data. It is also possible to specify another XML-Schema for validating the data. An example is MathML elements under <technique_common> inside the <formula> element. Profiles (e.g. <profile_common>, <profile_bridge>) represent the several sets

of functionalities for a specific platform. They encapsulate all the platform-specific values and declarations. The <extra> element is used to represent additional real data or semantic (Meta) data to the application. Extra information is represented as techniques containing arbitrary XML elements and data.

Understanding the content of the COLLADA document, in other words realizing how the information types represented in COLLADA format, is important to implement a data exchange software component for engineering tools using 3D content, as it is the case for the target tool set of this work. COLLADA content will be discussed in four chapters with the explanation of corresponding element libraries: COLLADA Geometry, COLLADA Topology, COLLADA Kinematics, and COLLADA Physics.

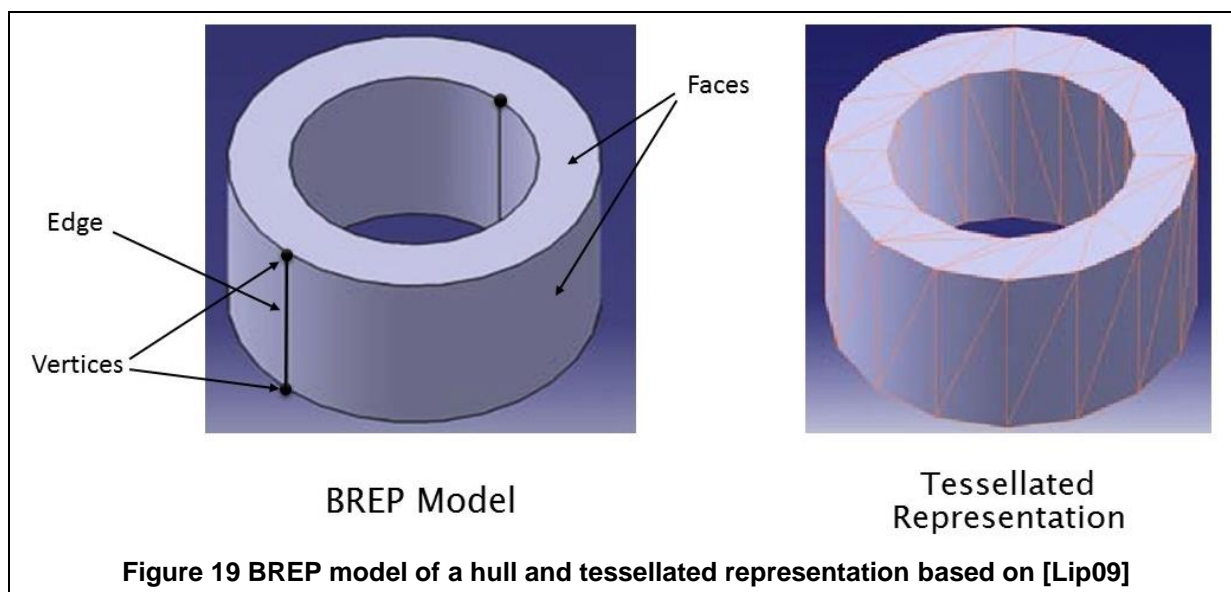
5.2.2.1. COLLADA Geometry

The <geometry> element is the container of the information that describes a geometric shape. There are possible representations of 3D objects when creating digital assets. Some common forms are B-Spline, Bézier curve, polygonal mesh, NURBS, patch and subdivision surfaces. COLLADA contains only polygonal meshes and splines. With the version of COLLADA 1.5 the boundary representation (BREP) is supported for the exact representation of the geometrical shapes. The actual type and complexity of the data are represented in detail by the child elements of <geometry>.

BREP represents the data structure of an object by giving information about each of the object's faces, edges, and vertices and how they are joined. The description of the object can be divided into two parts: Topology and geometry. Topology shows the connectivity of the faces, edges, and vertices in the data structure. Geometry describes the exact shape and position of each of the edges, faces, and vertices. The geometry of a vertex is given by its position in space with x/y/z coordinates. Edges can be i.e. straight lines, circular arcs. A face is represented by the description of its surface.

BREP models describe forms of any dimension by defining forms in the next lower dimension and the boundaries. The limiting elements of a shape are also the basis for the shapes of the next dimension elements. An edge is therefore limited by vertices (points), a wire is limited by edges, faces are made of wires, solids are defined by faces.

Figure 19 shows a BREP model of a hull. The given form is a limiting of three-dimensional space. It describes a solid, or simply said, it describes where matter is and where not. For the representation, the solid is restricted by two cylinders (inner and outer) and two planes (top and bottom). Both planes are limited by two circles, meaning one-dimensional elements [Lip09].



BREP models are represented by the element `<brep>`. The child elements of `<brep>` element contains the geometric elements to represent a BREP geometry. BREP cannot be visualized directly; they must first be "tesselated". Tesselation means that the surfaces should be converted into geometry primitives, which are suitable for graphics cards [MCT16].

COLLADA represents the tesselation as a comprehensive set of polygon-based geometry descriptions. Tesselated geometries are included as child elements in `<mesh>` element. These are `<lines>`, `<linestrips>`, `<polygons>`, `<polylists>`, `<triangles>`, `<trifans>` and `<tristrips>`. Figure 20 introduces the basic structure of a triangle-based geometry in COLLADA.

The `<source>` elements are used for raw data streams, which are accessed through the `<input>` elements. They also provide access methods to the information. The raw data can be included in the `<source>` element as the content of an array element or a reference to an external resource in any other format.

The majority of geometrical data in COLLADA is represented by array values without semantics. There are five types of array [Arna06]:

1. `<float_array>` for floating-point numbers
2. `<int_array>` for signed integers
3. `<bool_array>` for Boolean
4. `<IDREF_array>` for references to Ids
5. `<Name_array>` for names and SIDs

```

<geometry id="geo0" name="GEOM_0">
  <mesh>
    <source id="geo0.positions">
      <float_array id="geo0.positions-array" count="60">-3000 -3000 0 -3000 -3000 0 -3000 -3000 0 -3000 3000 0 -3000
      15000 1500 0 15000 3000 -3000 0 3000 -3000 0 3000 -3000 0 3000 -3000 0 3000 3000 0 3000 3000 0 3000 3000 0</float_array>
    <technique_common>
      <accessor count="20" source="#geo0.positions-array" stride="3">
        <param name="X" type="float"/>
        <param name="Y" type="float"/>
        <param name="Z" type="float"/>
      </accessor>
    </technique_common>
  </source>
  <source id="geo0.normals">
    <vertices id="geo0.vertices">
      <input semantic="POSITION" source="#geo0.positions"/>
    </vertices>
    <triangles count="2" material="mat0">
      <input offset="0" semantic="VERTEX" source="#geo0.vertices"/>
      <input offset="1" semantic="NORMAL" source="#geo0.normals"/>
      <p>17 17 15 15 2 2 2 2 4 4 17 17</p>
    </triangles>
    <triangles count="3" material="mat1">
    <triangles count="3" material="mat2">
    <triangles count="3" material="mat3">
    <triangles count="3" material="mat4">

```

Figure 20 Structure of a triangle based geometry in COLLADA

The <vertices> element is a special element to group all the vertex attribute data, which are included in the <source> element. It contains all the vertex information that is invariant to primitive winding and assembly such as points and 3D geometries. Other source data can also be normal arrays, texture coordinates, color coordinates, etc. The source element can be referenced by the geometric elements.

The geometry elements have child elements like the <triangles> element in Figure 18. The <input> element is used to associate a semantic to a data source and declaring an input to the collation element. The <p> element references the indices for the inputs. This combination describes the attributes of all the edges in the geometry primitive.

Materials are associated with a geometry when they are instantiated. The definition of the material is not stored in the geometry primitive. Otherwise, they would be fixed to the geometry, but it is helpful to provide different colors or textures to the same geometry element. <library_materials> library contains the <material> elements, which has the definition of a material.

An <effect> element describes the visual properties of a material, which is inside the COLLADA library <library_effects>. The effect portrays the light behavior of a surface such as reflection or transparency [Bar08].

5.2.2.2. COLLADA Topology

A single geometry is not enough for the representation of a complete system (e.g. a robot). Several geometries should be combined in a hierarchical structure for such a component. The same geometry definition can be used several times as different instances, can be transformed in the 3D space, or scaled to have different sizes of an identical shape. Some

components can even be set invisible. These functionalities can be applied in a so-called product tree, in which the individual geometry and partial products are assembled to a complete product. The <node> element is used to describe such a product tree. This element can have itself as a child element. The geometric position or scale of a component can be changed through transformation elements. These transformation elements are <matrix>, <rotate>, <translate>, <scale> and <screw>. The instantiation of the desired geometry is done by the <instance_geometry> element. With the element <bind_material> under <instance_geometry>, the material definition is bonded to an instantiated geometry with the *symbol* and *target* attributes of <instance_material>. These *symbol* values assign materials with ids defined in *target* to the *material* attribute of primitive geometries in the <library_geometries> [Lip09]. For example, <triangles> element has the *material* attribute with the value "mat1". The <instance_material> has the *symbol* value "mat1" and *target* value "material4278190335" that links <material> with id "material4278190335" with the <triangles> element.

In Figure 21, an example node structure is represented. The <node> element can be used under <visual_scene> element or defined separately under <library_nodes> element.

```

<library_visual_scenes id="libvisualscenes">
  <visual_scene id="libvisualscenes.scene">
    <node id="root" name="Product1">
      <node id="Product.1_0" name="Product">
        <instance_geometry url="#geo0">
          <bind_material>
            <technique_common>
              <instance_material symbol="mat0" target="#material4278190335"/>
              <instance_material symbol="mat1" target="#material4278190335"/>
              <instance_material symbol="mat2" target="#material4278190335"/>
              <instance_material symbol="mat3" target="#material4278190335"/>
              <instance_material symbol="mat4" target="#material4278190335"/>
            </technique_common>
          </bind_material>
        </instance_geometry>
      </node>
      <node id="Product2.1_1" name="Product2">
        <matrix>1 0 0 0 0 1 0 0 0 0 1 30000 0 0 0 1</matrix>
        <matrix>1 -0 0 -0 -0 1 -0 0 0 -0 1 -30000 -0 0 -0 1</matrix>
        <matrix>1 0 0 0 0 1 0 0 0 0 1 30000 0 0 0 1</matrix>
        <rotate sid="joint_1_axis0">-1 -0 -0 0</rotate>
        <matrix>1 -0 0 -0 -0 1 -0 0 0 -0 1 -30000 -0 0 -0 1</matrix>
        <matrix>1 0 0 0 0 1 0 0 0 0 1 30000 0 0 0 1</matrix>
        <instance_geometry url="#geo1">
      </node>
      <node id="Product3.1_2" name="Product3">
    </node>
  </visual_scene>

```

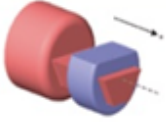

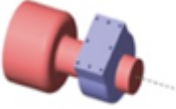
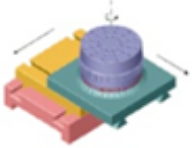
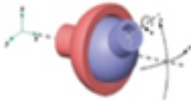

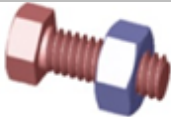
Figure 21 A simple product tree example

<scene> embodies the entire set of information that can be visualized from the contents of a COLLADA resource. Each COLLADA document can contain, at most, one <scene> element. A scene contains an instance of the visual scene to display the product tree. More about <scene> element will be explained in the following chapter 5.2.2.4.

5.2.2.3. COLLADA Kinematics

The kinematics of system components comprehends the physical relationships between the elements, like joint connections. There is a strict separation of geometry and kinematics in COLLADA. The task of a kinematic model is to provide all necessary information required by an application to solve an inverse kinematic problem. It should be possible to model a simple kinematics efficiently without overloading the required information, for example, the information for programming an industrial robot [Lip09].

Table 3 Joint types and definitions based on [Lip09]

| Joint Type | Degree of Freedom (DOF) | Example |
|-----------------|----------------------------------|---|
| Prismatic | 1x transformation |  |
| Revolute | 1x rotation |  |
| Cylindrical | 1x transformation 1x rotation |  |
| Planar | 2x transformation 1x rotation |  |
| Ball and socket | 3x rotation |  |
| Universal | 2x rotation |  |
| Screw | 1x transformation 1x rotation |  |

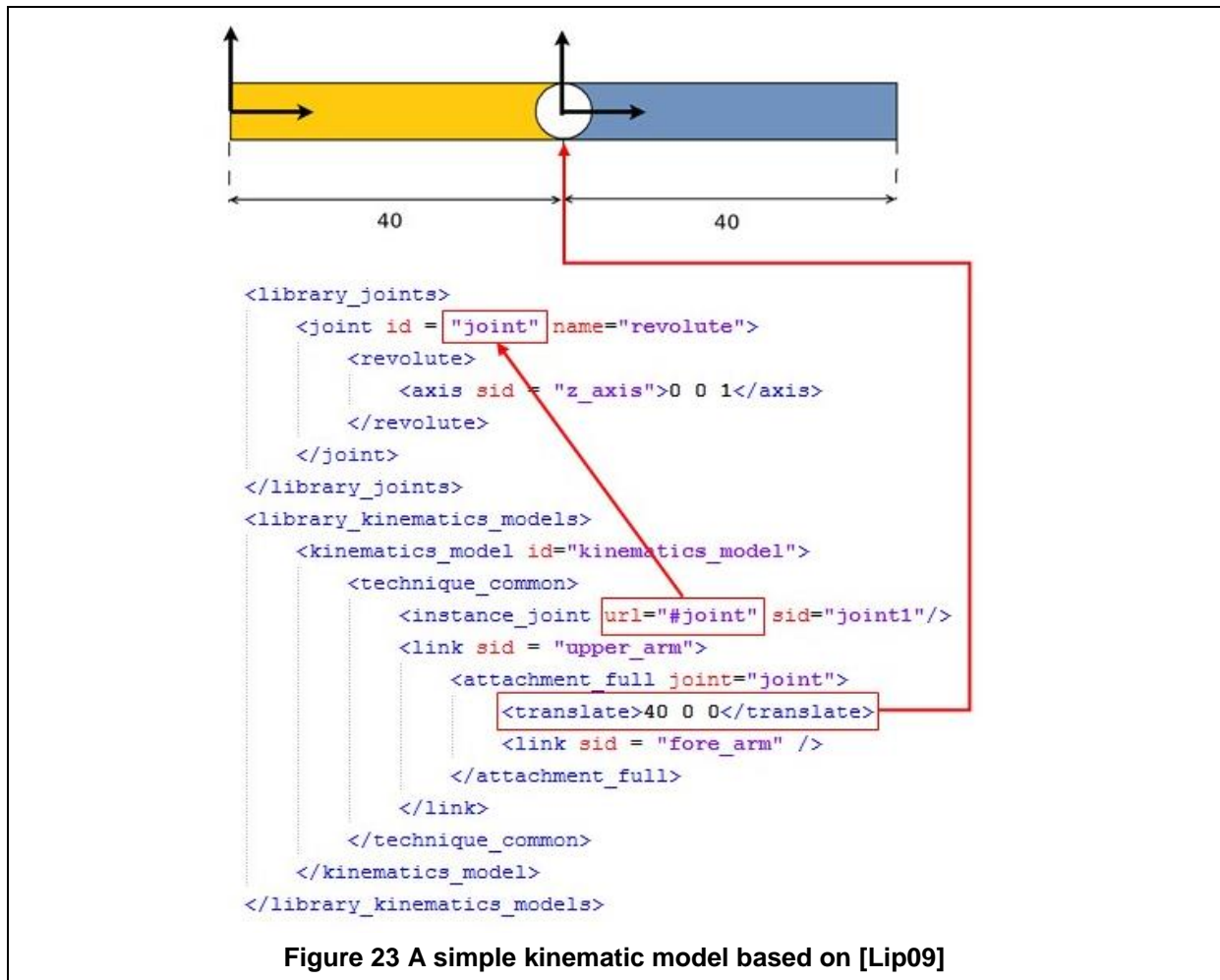
COLLADA 1.5 provides two joint primitives: The rotational joint defines the degree of freedom as a rotation about an axis. The translational joint defines the degree of freedom as a displacement along one axis. All other kinds of joint types can be derived from these two basic joint types. Some examples of joint types can be seen in Table 3.

A joint in COLLADA is represented by the <joint> element. This element includes child elements of type <prismatic> and <revolute> for the basic joint types. These joint type elements consist of the element <axis> and <limits>. The <axis> element defines the axis from the degree of freedom. Optionally, the range of a joint can be limited with the <limits> element [Lip09]. Figure 22 shows the definition of a prismatic joint and a universal joint.

```
<library_joints>
  <joint name="prismatic">
    <prismatic>
      <axis sid = "x_axis">1 0 0</axis>
      <limits>
        <min>-30</min>
        <max>50</max>
      </limits>
    </prismatic>
  </joint>
  <joint name="universal">
    <revolute sid="revolute1">
      <axis sid = "y_axis">0 1 0</axis>
      <limits>
        <min>-180</min>
        <max>180</max>
      </limits>
    </revolute>
    <revolute sid="revolute2">
      <axis sid = "z_axis">0 0 1</axis>
    </revolute>
  </joint>
</library_joints>
```

Figure 22 Joint definition in COLLADA [Lip09]

The joint definitions are used in kinematic models, which are the basis for calculating and solving kinematic problems, e.g. like the path planning of a welding process. Kinematic models or kinematic chains are systems of rigid bodies (links) which are connected by joints. They are defined by the COLLADA element <kinematics_model>. The joints, which were previously described in <library_joints>, can be instantiated by the element <instance_joint>. Likewise, the definition of new joints is possible. After that, the kinematic chain is defined, which starts with the <link> element. The link can be attached to the other links through the element <attachment_full>. Through the transformation elements <rotate> and <translate>, the following link can be positioned accordingly. Figure 23 shows how a simple one-joint kinematics is set up [Lip09].



Kinematic model elements describe only the degrees of freedom. The model must be enriched with more information to create an actual simulation of a system. COLLADA 1.5 differs the enrichment of purely kinematic aspects that form the basis for solving an inverse problem and purely dynamic aspects that determine the behavior of the kinematics while following a track. "Articulated systems" are used to enrich a kinematic model with such information.

An articulated system is defined by COLLADA element `<articulated_system>`. It is followed by the definition of which aspect of the model is described. This is done by the child elements `<kinematics>` or `<motion>`.

A model is first instantiated in the element `<kinematics>` to describe kinematic aspects. Within the instance, the parameters of the model are defined by the `<newparam>` element for further use. These elements are used for the reference to the instantiated model, references to the joints of the model, and the establishment of an axis value of the joint.

The kinematic aspects of a joint are determined in `<axis_info>` element, for example, the activity or the index of the joint vector. The properties, which are defined here, can also be determined by a parameter element [Lip09].

5.2.2.4. Combining Kinematics and Geometry

The results of kinematics should be combined with defined geometric shapes to represent the movement of a component visually in a software tool. For this purpose, a kinematics scene should be created analogous to a visual scene, which is done by the <kinematics_scene> element. Any number of kinematic models <instance_kinematics_model> or articulated systems <instance_articulated_system> can be instantiated within the scene.

The element <scene> defines what is visualized later on the display. And that is where the combination between the visual and kinematics scene is made. Both scenes need to be instantiated by the elements <instance_visual_scene> and <instance_kinematics_scene>. Which kinematic model is coupled to which component in the product tree is determined within the instance of the kinematic scene by the <bind_kinematics_model> element. Then the joints are coupled into the corresponding transformations in the product tree through the element <bind_joint_axis>, which is referencing a transformation element (<translate>, <rotate>, <scale>, <matrix>, <skew>, <lookat>) of a node in the visual scene (see Figure 24).

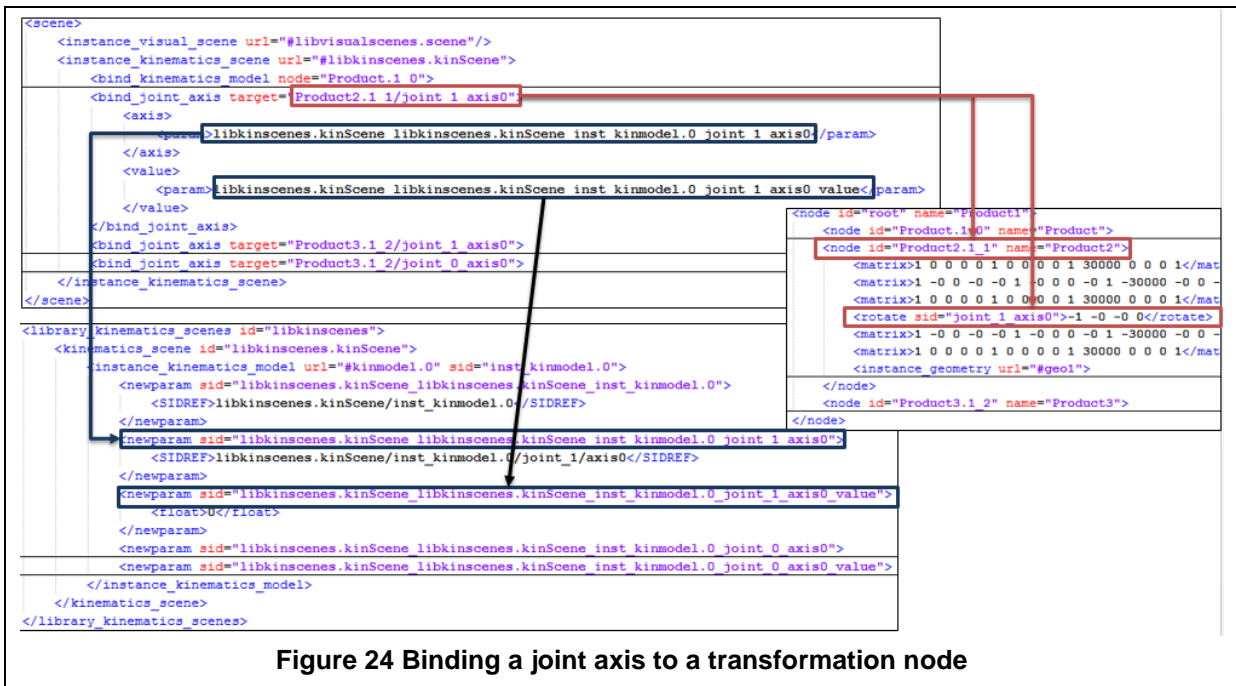


Figure 25 shows the connection of the kinematics scene with geometry. The element <axis> references the joints, and the element <value> defines the joint value, which should be included in this scene.

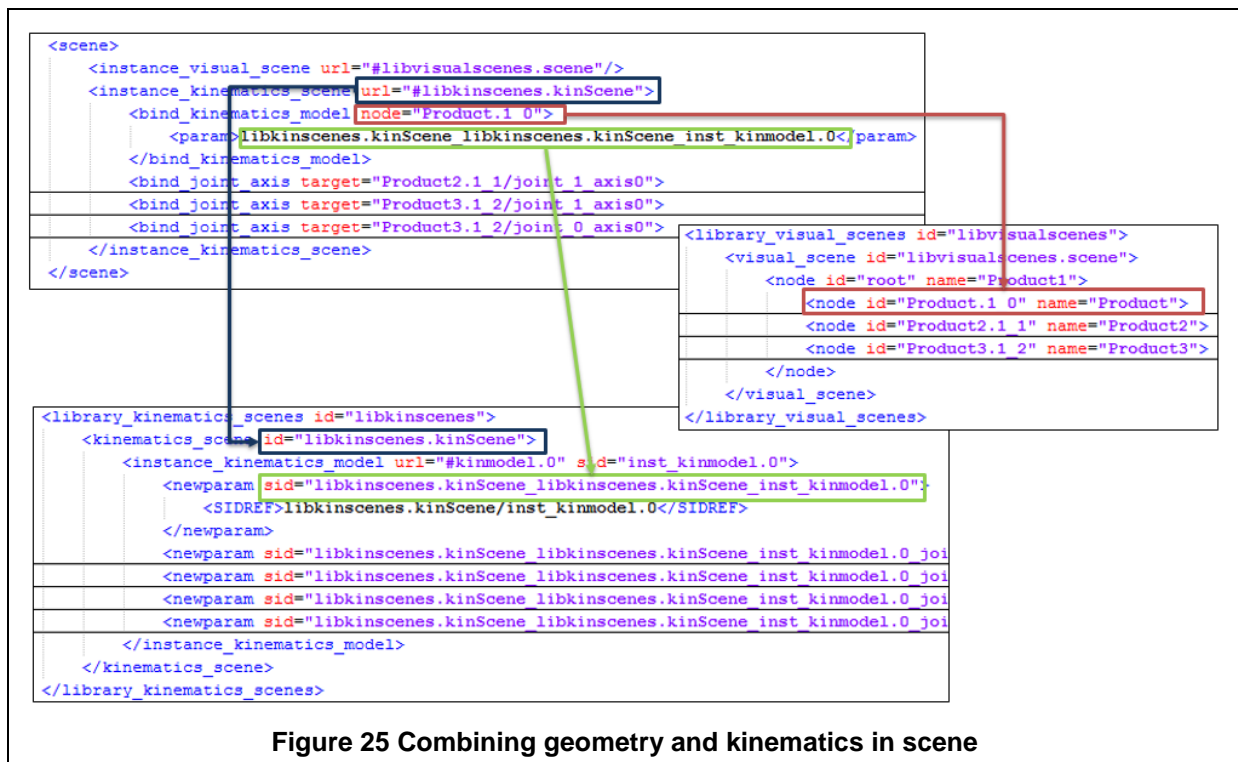


Figure 25 Combining geometry and kinematics in scene

5.2.2.5. COLLADA Physics

COLLADA physics libraries support basic rigid body dynamics. A rigid body is a non-deformable object that interacts with other rigid bodies according to Newton's basic laws of physics. A rigid body is represented by the `<rigid_body>` element in COLLADA, which consists of parameters and a collection of shapes for collision detection [Bar08].

The `<rigid_constraint>` element specifies how one body can move in relation to the other. It can limit the various angular and linear degrees of freedom. It is determined by two attachment frames and degrees of freedom (DOF).

The `<physics_model>` element is a logical grouping mechanism for rigid bodies and constraints. Physics models might be defined as a single rigid body or with rigid bodies which have constraints linking them. Physics models are gathered under `<library_physics_models>` library.

Rigid-bodies may contain a single shape or a collection of shapes for collision detection. For example, the `<convex_mesh>` element generates the convex hull for a given mesh to ease the collision calculation. It is also possible to define analytical shapes (boxes, spheres, capsules) for collision volumes. Using analytical shapes is preferred for better representation of certain round surfaces, improved performance, and reduced memory usage.

A `<physics_material>` or `<instance_physics_material>` describes the surface properties (i.e. restitution, friction) for a rigid body.

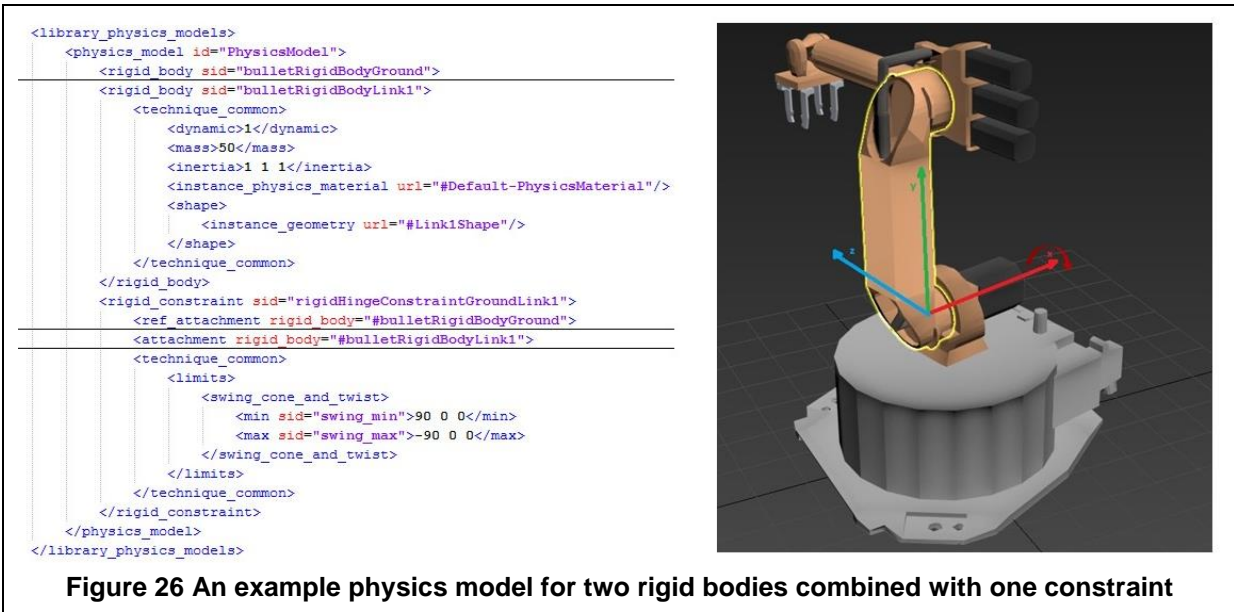
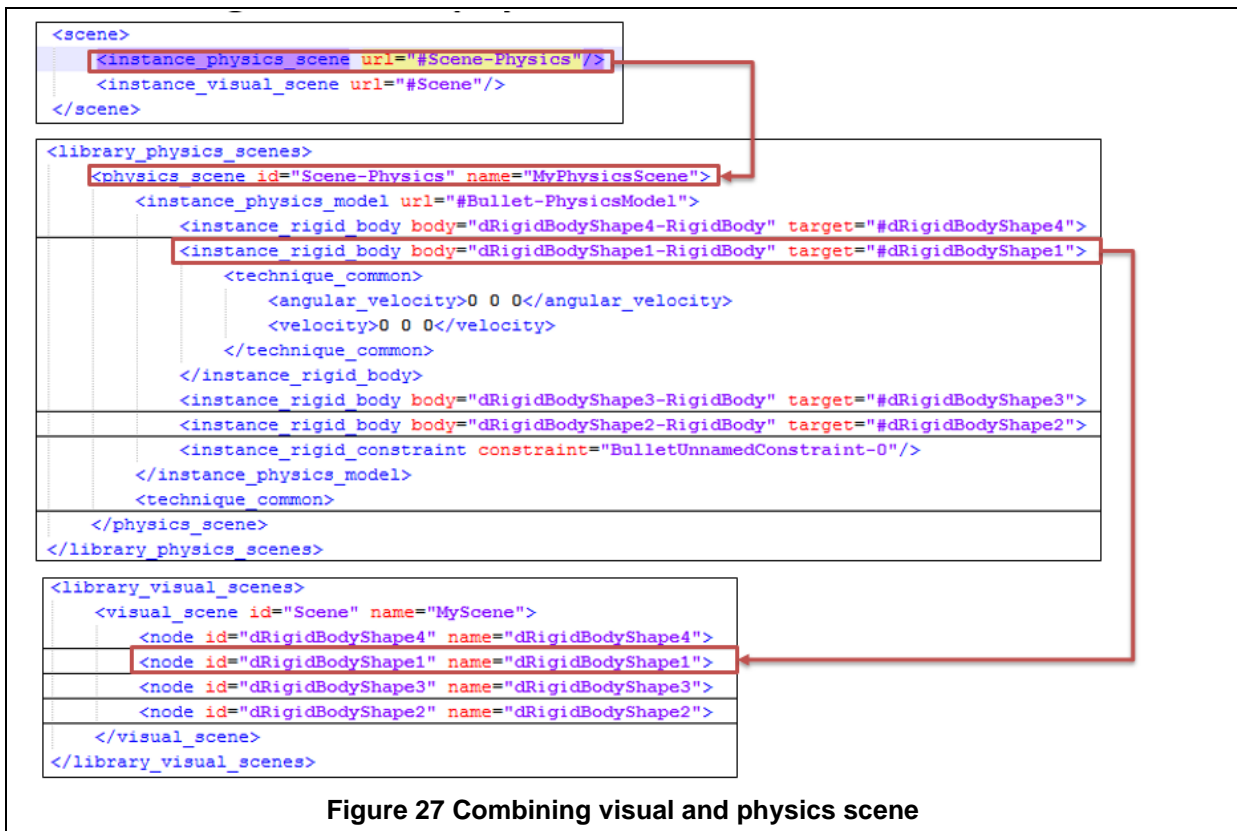


Figure 26 demonstrates a manipulator arm attached to a ground body. The rigid constraint is limited to be rotated +/- 90 degrees on its x-axis.



Physics-based systems do not have the same notion of hierarchy or parent-child relationships used by animated articulated models. Physics models are visualized by having their instantiated rigid bodies directly controlling the placement of nodes in the visual scene. The node could display different geometry than what is defined in the physics scene. This feature is used for optimizing the performance of collision calculation with complex geometrical

shapes. For example, a bottle can be represented visually with the real geometry but physically with a simple cylindrical shape. The Figure 27 shows how the physics scene can be combined with a visual scene.

5.2.3. Functional Mockup Interface

Functional Mockup Interface (FMI) is a tool independent standard, which covers the aspects of model exchange and co-simulation of dynamic physical models using a combination of interface description (an XML-file) and C-code (either compiled as DLL/shared libraries or as source code). These files (optionally together with some documentation) are packaged and distributed in an FMU (Functional Mockup Unit) component, which is a zip file with the extension “.fmu”. FMU can be utilized either for model exchange, for co-Simulation or both. A simulation environment can create one or more instances of the FMU to call the FMI functions and to simulate the models. An FMU may either include its solvers or require an external environment to execute numerical integration. The latest version of the standard is the FMI 2.0. [MOD14].

The FMI 2.0 standard provides two main functionalities: 1) Model exchange, 2) Co-Simulation.

5.2.3.1. FMI for Model Exchange

The goal of the model exchange interface is to allow the simulation environment to solve a system of differential, algebraic and discrete equations numerically. The central concept is that a modeling tool can generate and export a C-Code of the system model in the form of an input/output block together with an XML-file containing the definition of all variables (see Figure 28). It is then possible to run the FMU on a target system (e.g. other modeling and simulation environments).

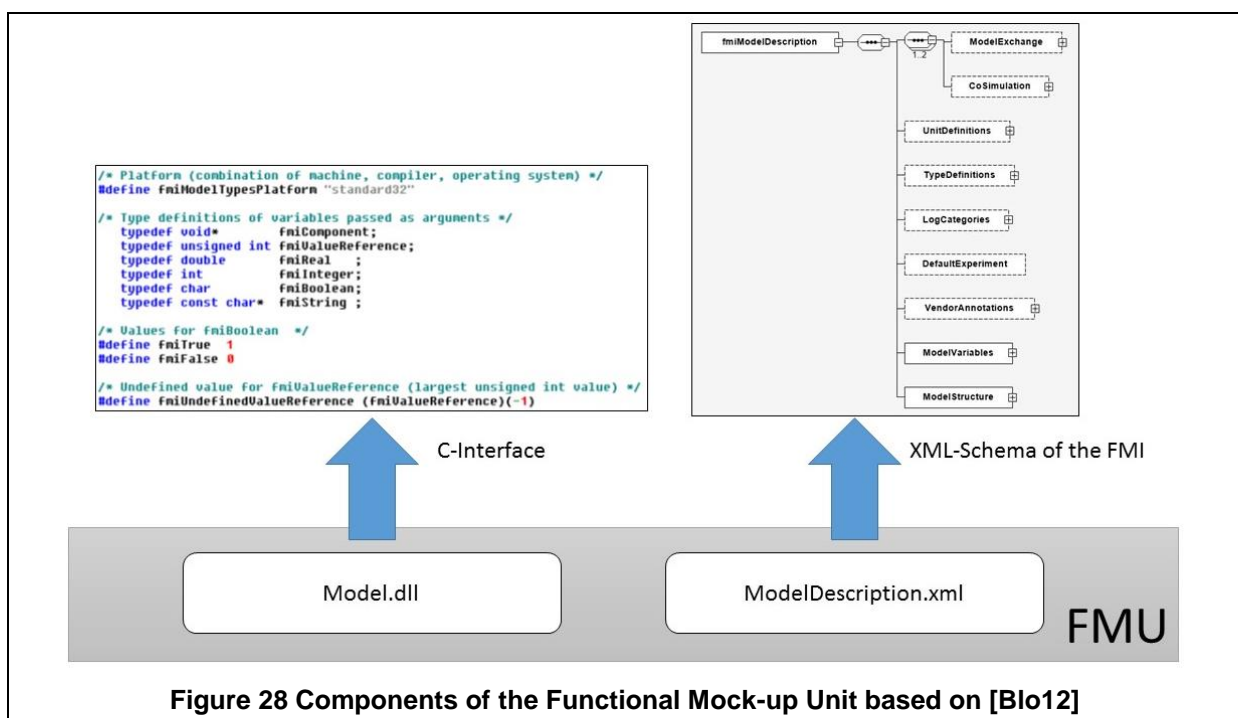


Figure 28 Components of the Functional Mock-up Unit based on [Blo12]

5.2.3.2. FMI for Co-Simulation

FMI can be used to couple two or more models with solvers in a co-simulation environment. In the time step between two states, the individual solvers of the subsystems are executed independently from each other. Intermediate results (variables, status information) are exchanged between FMUs and other possible solvers during simulation. A co-simulation master controls the data exchange, which is restricted to discrete communication points. It also manages the synchronization of all slave simulation solvers. The usage of variable communication step sizes, higher order signal extrapolation, and error control are defined by the interface [Blo11].

5.2.3.3. Integration of FMI into AML

The integration of behavior description with FMU into the AML is done by creating an external reference via an appropriate interface. For this purpose, a specific AML-Interface “*FMUInterface*” is defined, which is derived from base interface class “*ExternalDataConnector*” similar to other standard external reference interfaces like for COLLADA and PLCOpenXML [Gra13]. A role class “*FMU*”, which is derived from “*ExternalData*” role class, is also necessary to identify the referenced object as FMU and contain the interface which is referencing the corresponding *.fmu file. “*FMU*” role class contains a second interface “*VariableInterfaces*” to map the variables, which is defined in the “*ModelDescription.xml*” inside the FMU file (see Figure 29) [LNY16].

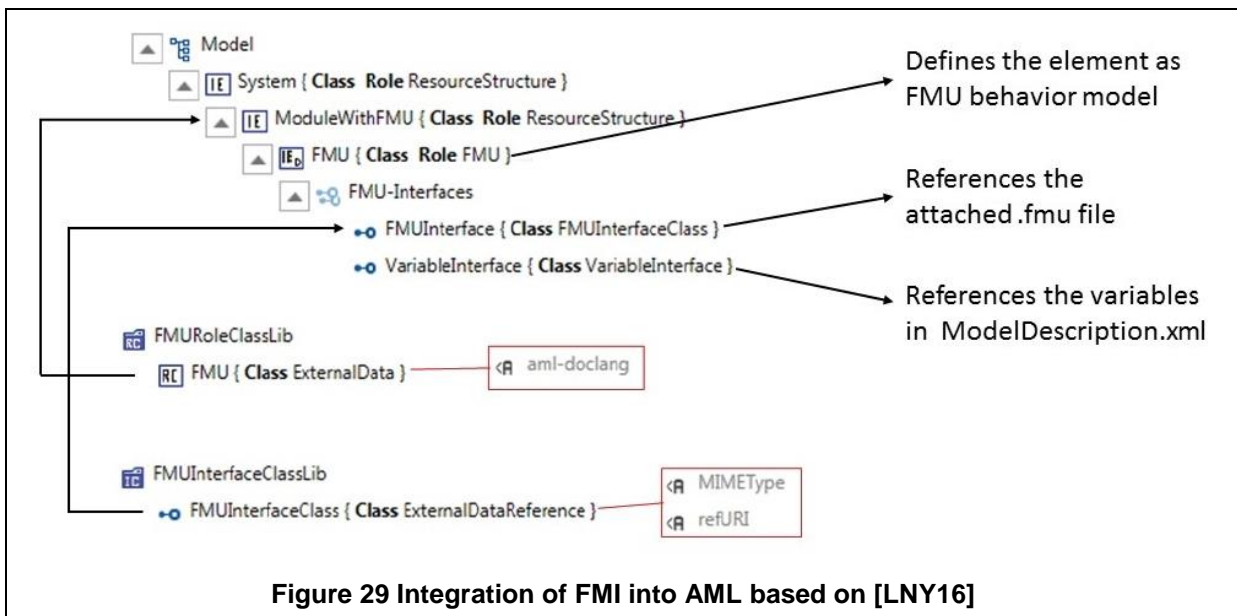


Figure 29 Integration of FMI into AML based on [LNY16]

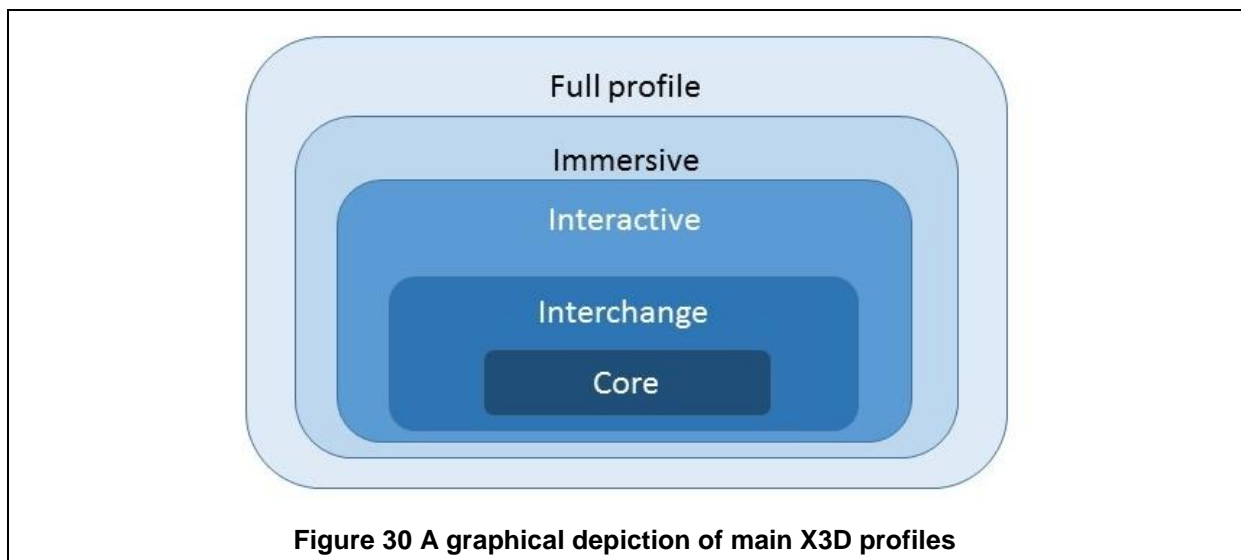
5.3. Overview of the native format (X3D/VRML)

tarakos software tools use X3D/VRML format to define their proprietary geometry and animation models. It is necessary to have an overview about them to understand the information structure of proprietary tools. The Virtual Reality Modeling Language (VRML) is a web-based standard file format (ISO/IEC 14772) for describing interactive 3D objects. VRML

is the predecessor and the subset of XML-based standard format Extensible 3D (abbr. X3D, ISO/IEC 19775). The X3D has features for 2D/3D graphics, animation, spatialized audio and video, user interaction, navigation, scripting, user-defined objects, layering, and others [Web13].

The scope of the X3D language is very broad so that it can be confusing without a systematic categorization. X3D framework is divided into manageable thematically related modules for a clear view of the standard. The modular structure is also useful for using only the part that is needed for the application and dropping out the other. As seen in Figure 30, the main profiles in X3D are [Web13]:

- The **core** defines the minimum requirements for X3D format. The nodes in this profile must be supported to satisfy the minimum standard requirements.
- **Interchange** is used for the exchange of geometry and animations between applications. Geometry, texturing, basic lighting, and animation are defined in this profile. Sufficient for the most of the static 3D layout applications.
- **Interactive** defines basic interaction through sensor nodes for user navigation enhanced timing, and some additional lighting.
- **Immersive** contains all 3D graphics and interaction nodes in the format including audio support, collision, fog, prototype, and scripting. Relevant if the format should be extended through prototype elements (see Chapter 5.3.4).
- **Full** profile covers all nodes including NURBS, H-Anim, and GeoSpatial components.



3D graphical components are the central part of the data exchange within the scope of this work, but some own defined nodes are also necessary for the sharing of physical information. X3D standard allows such extensions with the *Prototype* elements (<ProtoDeclare>, <ProtoInstance>), which is contained in the immersive profile. As a result, the range of the application reaches to immersive profile. More explanation will be given in the following chapters.

X3D is one of the scene description languages (SDL). It uses object oriented modeling aspects to define the constellations and relationships between the objects and their attributes in a scene. An X3D scene consists of an ordered list of objects, which are called “*nodes*” (similar to COLLADA). There are two types of node: Grouping nodes and child nodes. As the name implies, grouping node summarizes other nodes, starting from a central group node, so called *root node*. A hierarchical tree structure is created through the connection of these nodes, which is named as the *scene graph*. Every element in the scene graph can be identified with **DEF** attribute. A reference to an existing element is also possible with the help of **USE** attribute [Klo10].

The X3D scene graph consists of many object types, like Meta types, geometry primitives, animation, and scripting components.

5.3.1. Meta information

Meta information for X3D files can be provided in the header section. The header element is defined with the <head> tag. The individual metadata is defined within the element <meta>, which contains corresponding attributes with some values assigned [Klo10]. An example of header element can be seen in Figure 31.

```
<head>
  <meta name="title" content="Projekt_aml.dae" />
  <meta name="generator" content="tarakos COLLADAtoX3DConverter" />
  <meta name="creator" content="tarakos GmbH" />
  <meta name="author_email" content="ender.yemenicioglu@tarakos.com" />
  <meta name="author_website" content="www.tarakos.com" />
  <meta name="authoring_tool" content="tarakos Vrm1 To Collada Converter Version 1.1" />
  <meta name="copyright" content="tarakos GmbH" />
  <meta name="created" content="2015-08-28T10:28:13Z" />
  <meta name="modified" content="2015-08-28T10:28:42Z" />
  <meta name="description" content="Converted from Collada to X3D by tarakos COLLADAtoX3DConverter" />
  <meta name="unit_meter" content="1" />
  <meta name="unit_name" content="meter" />
</head>
```

Figure 31 Header element in X3D for the definition of Meta information

5.3.2. Geometry primitives

Geometry primitives are the visual components of an X3D scene. The most used geometry types are polygons and triangles. All geometry elements are derived from a parent geometry node <X3DGeometryNode>. The <Shape> node is used to bundle a geometry node with an appearance node. Appearance defines the color, material properties, or texture image. Materials state special effects such as transparency, shininess, or luminosity. All geometry types have one common field: **solid**. The solid field is a Boolean parameter, which is true if the geometry is viewable from both sides of the polygons, and false if it is visible from only one side. This parameter is used to reduce the number of polygons and thereof resulting rendering costs [Bru10].

A simple example for an X3D geometry could be a <Box>. It is defined by the width, height, and depth dimensions. An example shape with a box can be seen in Figure 32.

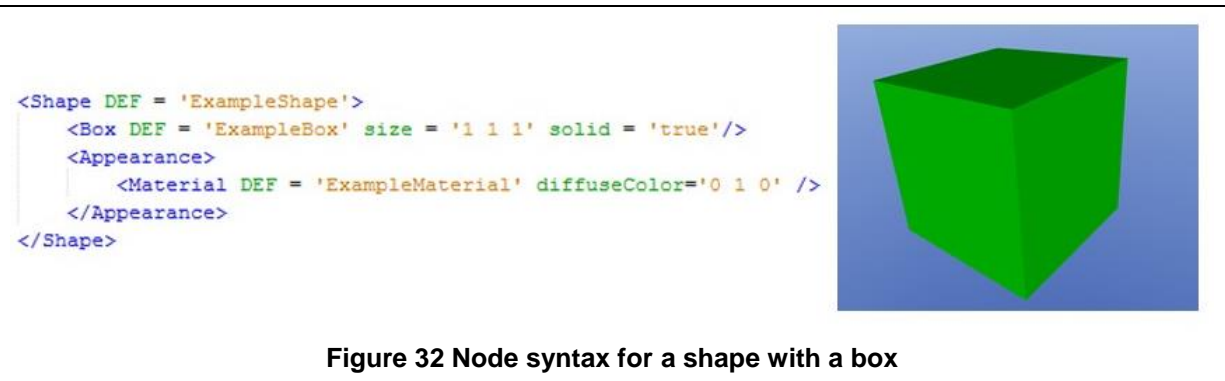


Figure 32 Node syntax for a shape with a box

Some other commonly used geometry primitives are <Cylinder>, <Cone>, <Sphere>, <IndexedTriangleSet> and <IndexedFaceSet>.

5.3.3. Grouping nodes

A large scene graph can be structured better by grouping the elements. There are grouping nodes for this purpose. The important ones for this work are <Transform>, <Group>, and <Inline>. These elements contain other nodes as children [Bru10].

The simplest of these type of objects is the <Group> node. It is used to collect related nodes into a single parent in the scene graph hierarchy.

The <Transform> node is a grouping node that defines the coordinate system for all its children relative to the parent coordinate system. Translation, rotation and scale are the transformation types, which can be defined in this node. The position is given in meters and angle in radians.

<Inline> is a grouping node, which can reference nodes from an external X3D scene. The exterior scene is retrieved via [URL](#) attribute field in the inline element.

5.3.4. Prototypes

A prototype element, which is defined by a <ProtoDeclare> statement, provides a way to create new nodes from other X3D standard nodes. A prototype can be used in the scene graph like any other node after the declaration. Prototypes as customizable objects can be used to define fields and even embed script code [Bru10].

Prototype declarations can be kept in a separate X3D file and reused when needed with the help of <ExternProtoDeclare> statement. External prototype is almost equivalent to a prototype, with two exceptions: 1) the implementation of the node type is stored externally, usually in an external X3D file, which is containing the corresponding prototype declaration; 2) default values for the fields are not defined because the implementation will provide the appropriate defaults [Web13].

They are used i.e. to define physics elements in X3D/VRML scene graph in this work and play an essential role for the communication between the physics engine and taraVRControl tool. An example usage of a physics transform prototype can be seen in Figure 33. The fields in the <ExternProtoDeclare> element declares the <PhysicsTransform> prototype and describe the necessary parameters for the physics model like mass, restitution, friction, etc. <PhysicsTransform> prototype contains a collision shape, which is defined as a standard X3D/VRML geometrical component and enhance the standard shape with physical attributes.

```

<Scene>
  <ExternProtoDeclare name="PhysicsTransform" url="templates/tvrcPhysicsTransform.wrl#PhysicsTransform">
    <field accessType="inputOutput" name="mass" type="SFFloat"/>
    <field accessType="inputOutput" name="restitution" type="SFFloat"/>
    <field accessType="inputOutput" name="translation" type="SFVec3f"/>
    <field accessType="inputOutput" name="rotation" type="SFRotation"/>
    <field accessType="inputOutput" name="collisionShape" type="SFNode"/>
    <field accessType="inputOutput" name="attachedObjects" type="MFNode"/>
    <field accessType="inputOutput" name="linearVelocity" type="SFVec3f"/>
    <field accessType="inputOutput" name="angularVelocity" type="SFVec3f"/>
    <field accessType="outputOnly" name="collisionShapeChanged" type="SFNode"/>
  </ExternProtoDeclare>
  <PhysicsTransform DEF="FallingBox" mass='1' restitution='0.1' translation='0 10 0' rotation='0 0 1 0'>
    <Shape containerField="attachedObjects" DEF="BoxShape">
      <Appearance>
        <Material diffuseColor='0.8 0.2 0.2'>
        </Material>
      </Appearance>
      <Box size='1 1 1'>
      </Box>
    </Shape>
  </PhysicsTransform>
  <PhysicsTransform DEF="GroundPlane" mass='0' restitution='0' translation='0 0 0' rotation='0 0 1 0'>
    <Shape containerField="attachedObjects" DEF="GroundShape">
      <Appearance>
        <Material diffuseColor='0.2 0.2 0.2'>
        </Material>
      </Appearance>
      <Box size='50 0.1 50'>
      </Box>
    </Shape>
  </PhysicsTransform>
</Scene>

```

Figure 33 Physics transform prototype declaration and instantiation

5.4. Summary

The selection of the data exchange format is the first step for modeling and implementing new data exchange components, because it is the “language” of the communication between the software tools. The boundaries of the data format influence also the capacity of the information transfer.

AML is selected as the standard data exchange format because it combines different aspects of engineering planning, like visual, physical, behavioral, etc. This format is also extensible with any XML-format for the specialized requirements like co-simulation capacity, as it is done with FMI. AML format uses the object-orientation concept to define a structure but does not

provide the content of the information. The content must be prepared and standardized, which will be explained in the next chapter 6.

A brief introduction to COLLADA as a part of AML and AML-FMI combination is also given, as it is necessary to understand how the graphical, kinematical, and physical content is transferred from one tool to another with the help of these formats.

The last section provides an overview of the native format X3D/VRML, to understand the structure of information in proprietary software tools. Especially the extension for the physical information with the help of prototyping concept is an important part of the data exchange, as it is used in the physics-based simulation.

6. Modeling of Material Handling System Elements with AutomationML

“Whether you can observe a thing or not depends on the theory which you use. It is the theory which decides what can be observed” – Albert Einstein in his 1926 Berlin lecture

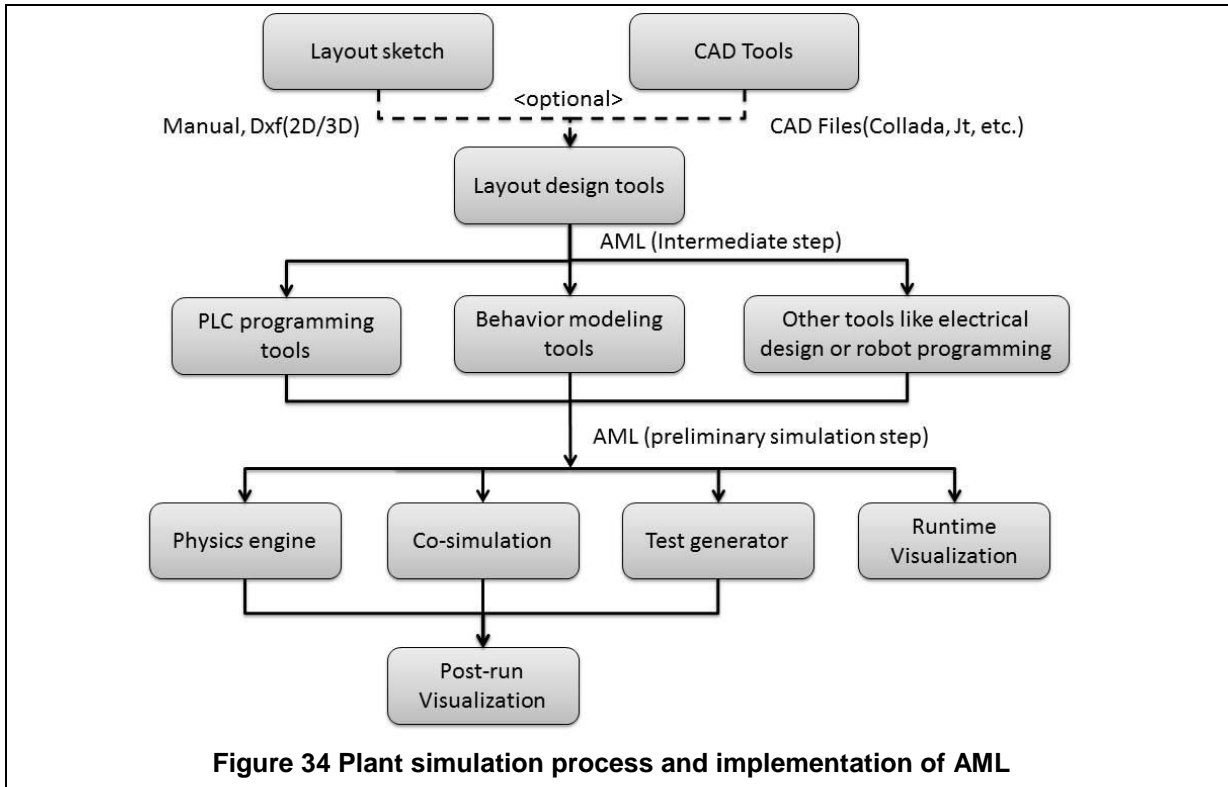
The first step for the data exchange is preparing the internal proprietary data model, as already mentioned in chapter 5.2.1.3. It is also needed as a preliminary work to understand the own model structure. The semantic content should be analyzed to create the semantic mapping between software tools, which are the subjects of the data exchange process.

6.1. Problem description

The steps of data exchange on the way to virtual commissioning should be clarified to define the task considered in this work. The need for a new product is determined at the beginning. After/During the product design, process engineers make a concept design for the corresponding manufacturing tools and workflow. If the geometry and kinematic information of manufacturing tools already exist, it can be transferred to layout planning tools. If not, it should be created with the help of 3D engineering tools. The layout plan can also be transmitted as manual sketches or 2D drawings to a layout planning tool. A 3D scene of the manufacturing plant can be created with the help of this information.

The outcome of the layout planning should be transferred to other engineering tools, which are specialized in particular fields: PLC programming tools, robot programming tools, electrical design tools, physical behavior definition tools, and others. Theoretically, most of these tasks, which are explained in detail before in chapter 2.2, can be fulfilled parallel, and the outcome of each process can be merged into one container file.

The resulting data can be used for creating a virtual scene, which can be connected with a physics engine for realistic visualization. With the help of co-simulation, other engineering solvers can be added to the simulation. An automatic test generation is also a possible task, which can be supplied with the result of co-simulation. Simulation results can be visualized at real-time (runtime visualization) or a post-run visualization could be necessary because of performance issues and to provide high fidelity of the outcome. Post-run visualization is more realistic than the runtime visualization because the simulation output is more accurate due to longer timing constraints.



As a neutral data exchange format, which includes the different facets of plant engineering, AML is a strong candidate to be the container of engineering-related data as explained in chapter 5. However, an implementation of import and export functions for the above-explained toolchain is essential (see Figure 34).

The export function requires the preparation of the internal proprietary data model. System component models contain different types of information, which should be converted/prepared into the object structure of the data exchange format. The significant aspects to be considered are topology, geometry, kinematics, physics, process definitions and other material handling relevant information types, mentioned in chapter 2.1.2.

Import function requires parsing the object models of the data exchange format and converting/mapping them into the internal proprietary data model. Although all aspects, which are mentioned in the export function, are also relevant for the import, the supported information types of the import functionality are restricted with topology, geometry, and physics currently.

6.2. Geometric model of the system components

The quality of the geometric models is significant for the simulation framework because the results of the physical calculations should be applied back into the visual scene to understand the bottlenecks and enhancement possibilities in the factory layout. An incorrect visual modeling of the system components can lead to false results like not recognized collisions or faulty collision warnings in the physical simulation.

Geometry models are gathered either from the component libraries of the proprietary software products or established from project partners in different geometry formats like .dxf, .jt, .stp, etc. The geometric model of the system components should be prepared for data exchange for further applications. Knowing that AML uses COLLADA as geometry format, the preparation of geometry models in COLLADA format is necessary. The native format of the tarakos software systems is X3D/VRML, so an implementation of a conversion method between these two formats is realized.

A brief comparison of the formats is needed to understand the similarities and differences of COLLADA and X3D/VRML. They are both XML-based file formats for representing structured 3D data, however with different design goals and intended use. COLLADA is an intermediate format, which focuses on the toolchain of various authoring tools, historically targeting the game industry. It is designed to represent data in multiple forms and to enable the transformation of assets. X3D focuses on the visualization of 3D assets and has principally been targeted for the web applications. X3D supports interactivity with applications, including a run-time model that enables picking, viewing, navigation, and scripting to manipulate the scene graph. However, COLLADA does not support interactivity in the standard specification [Arn07].

A comparison of the geometry primitives of these formats can be seen in Table 4. The details of the software component, which executes the conversion, can be found in chapter 7.2.3.

Geometry primitive types, which represent lines and triangular or polygonal meshes, can be converted to each other through defining the vertex list in the target format. However, some geometry types, which are defined only in native format X3D/VRML, should have to be triangulated to enable representation in COLLADA.

Table 4 Comparison of COLLADA and X3D geometry components

| COLLADA to X3D Conversion | |
|---------------------------------------|------------------------------------|
| COLLADA | X3D |
| Lines | PointSet, IndexedLineSet |
| LineStrips | IndexedLineSet |
| Triangles | IndexedTriangleSet, IndexedFaceSet |
| Tristrips, Trifans | IndexedFaceSet |
| Polygons, Polylists | IndexedFaceSet |
| X3D to COLLADA Conversion | |
| X3D | COLLADA |
| PointSet | Lines |
| IndexedLineSet | LineStrips |
| TriangleSet, IndexedTriangleSet | Triangles |
| TriangleFanSet, IndexedTriangleFanSet | Trifans |
| TriangleStripSet | Tristrips |
| IndexedFaceSet | Polygons |
| Cylinder, Cone, Sphere, Box | Triangles (Triangulation) |
| Extrusion | Polygons (Triangulation) |

Triangulation is the method for generating a mesh of triangles from a defined set of points. The triangles (faces) are created by edges (straight lines) between the points (vertices) of the point set. With the triangulation, it is targeted to have a mesh without overlapping edges and without an incorrect degenerate structure. The document from [Zim05] can be read to understand how the triangulation works and see different triangulation techniques.

Another problem by creating the geometry models is the representation of the transformations for nodes. A transform node can translate, rotate, or scale its children relative to the parent coordinate system, which helps to locate and orient objects correctly in relation to each other. There are different representations for transformations. It is necessary to change the

representations between rotation matrix and axis angle for the conversion between COLLADA und X3D/VRML.

A rotation matrix is a matrix that is used to perform a rotation in Euclidean space. Axis angle represents the rotation of a unit vector and an angle of revolution about that vector. The following three basic rotation matrices rotate vectors by an angle θ about the x, y, or z-axis, in three dimensions, using the right-hand rule:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Any rotation can be obtained from these three matrices using matrix multiplication. Yaw, pitch, and roll angles are α, β, γ :

$$R = R_z(\alpha)R_y(\beta)R_x(\gamma)$$

Using the approaches from [Bak16] to convert rotation matrix to axis angle with a rotation matrix R , a vector \mathbf{v} parallel to the rotation axis; the following equation should be satisfied:

$$R\mathbf{v} = \mathbf{v}$$

As the rotation of \mathbf{v} around the rotation axis must result in \mathbf{v} , the equation can be solved for \mathbf{v} which is unique up to a scalar factor unless $R = I$ (Identity matrix). So, by rewriting the equation:

$$R\mathbf{v} = I\mathbf{v} \rightarrow (R - I)\mathbf{v} = 0$$

That means \mathbf{v} is in the null space of $R - I$. \mathbf{v} is an eigenvector of R corresponding to the eigenvalue $\lambda = 1$. Every rotation matrix has this eigenvalue, the other two eigenvalues being complex conjugates of each other, which shows a general rotation matrix in three dimensions has, up to a multiplicative constant, only one real eigenvector.

The angle can be calculated through the sum of the diagonal elements of the rotation matrix. If rotation matrix represented like:

$$R = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix}$$

The angle θ can be found with the formula:

$$\theta = \arccos\left(\frac{m_{00} + m_{11} + m_{22} - 1}{2}\right)$$

However, there are two singularities at $\theta = 0^\circ$ and $\theta = 180^\circ$, which should be taken care of.

From axis angle to rotation matrix, with unit vector $\mathbf{v} = (v_x, v_y, v_z)$ and $v_x^2 + v_y^2 + v_z^2 = 1$, the matrix for a rotation by an angle of θ about an axis in the direction of \mathbf{v} is:

$$R = \cos(\theta)I + \sin(\theta) [v]_x + (1 - \cos(\theta))v \otimes v$$

It can also be formulated in the matrix form as:

$$R = \begin{bmatrix} \cos\theta + v_x^2(1 - \cos\theta) & v_x v_y(1 - \cos\theta) - v_z \sin\theta & v_x v_z(1 - \cos\theta) \\ v_y v_z(1 - \cos\theta) + v_z \sin\theta & \cos\theta + v_y^2(1 - \cos\theta) & v_y v_z(1 - \cos\theta) \\ v_z v_x(1 - \cos\theta) - v_z \sin\theta & v_z v_y(1 - \cos\theta) + v_x \sin\theta & \cos\theta + v_z^2(1 - \cos\theta) \end{bmatrix}$$

After the conversion of geometry models, a concept for transmitting the physical properties of the models should be considered, which is explained in the next chapter.

6.3. Physical model

There is a wide range of physical simulation types as mentioned in section 2.3.2, but this work is restricted to the rigid body simulation. The purpose of rigid body simulation is collision detection and collision response. It is used to check whether any of the rigid bodies are in contact, and if so what are the depth of the penetration, points of intersection, and the direction of the response force. It is also necessary to calculate impulses that keep the colliding objects from intersecting and cause them to move away from each other. External non-contact forces such as gravity should also be taken into consideration [Bab06]. Thereby it is possible to see i.e. if there is some unwanted collision between production equipment or goods or if the good falls to the ground because of a faulty operation.

Besides pure geometry models, it is also needed to parameterize the component models with physical properties. System components are divided into three main types to lighten the computational effort and increase the performance of calculations. The first type is the group of the objects, which are visually represented but are not taken into consideration in the calculations. They are important as a visual element to have a better understanding of the system but do not have any contact with critical system components, for example, conveyor's legs. The objects, which do not change their position in any way within the simulation scene, are classified as static bodies. They can collide with other dynamic bodies, but it is not necessary to calculate their collision response, for example, the scaffold of a conveyor. Objects, which have at least one degree of freedom (translation or rotation) in 3D coordinates, are defined as dynamic bodies. Objects, which have limited degrees of freedom (-also called "have constraints"-), are determined as kinematic dependent from other objects, which can be either static or dynamic, i.e. a robot arm attached to another one with a joint. All other dynamic bodies can be seen as free moving-dynamic bodies for calculation, for example, product good [Lac12]. In Figure 35, an example of such element types can be seen.

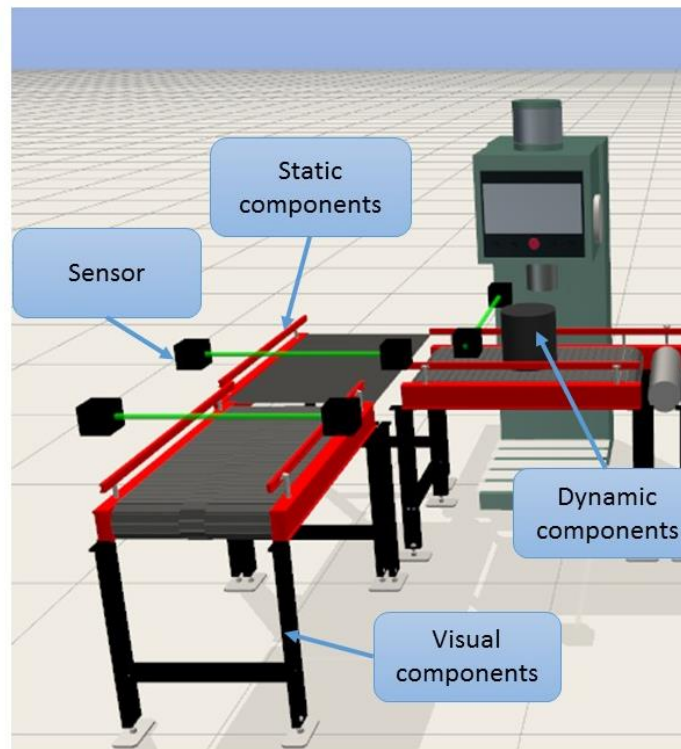


Figure 35 An example conveyor system for representation of visual, static, and dynamic components along with sensors

The amount of the necessary parameters in the model varies depending on which type of rigid body should be handled. A static body does not need mass information because it cannot be influenced by external forces or collision per definition. However, mass is a major factor for dynamic bodies to be able to calculate the collision responses. Mass can be gathered through the density and the volume of the object. Friction coefficient and restitution (also known as “bounciness” or “elasticity- is the proportion of the kinetic energy preserved in the impact) are also necessary parameters for static and dynamic bodies [Str12].

This information can be stored in COLLADA within the <rigid_body> element. The <dynamic> element, which can have the Boolean values *true* or *false*, shows if an object is a static or dynamic body. If no static or dynamic physics body is connected with geometry, then it is automatically only a visual component. The <mass> and <inertia> elements define the dynamic properties of a body. The <physical_material> element, which is a child of <rigid_body>, defines the dynamic friction, restitution, and static friction properties of the body [Bar08]. These elements and properties are represented as fields in X3D Prototype element <PhysicsTransform> after the conversion, as shown in Figure 33 in Chapter 5.3.4.

6.4. Topology and component description for material handling technology

The term *material flow* means the movement (transportation) of material through a production system. It is a set of actions in an assembly line in which a product is taken from a loading station, passes through intermediate steps like assembling or packaging, and arrives at the shipment. This set of operations is made by transportation systems such as carriers, conveyors, vehicles or sometimes robots. Material flow can be thought as a part of the general term *material handling*, which means not only the transportation of products but also their storage, control, coding, etc. [Din14].

The representation of the material flow is usually done in a quantitative form, i.e. the flow rate or the period between the arrivals of parts. The dynamic behavior of the material flow components can be modeled with knowledge of transport strategies and the corresponding static elements (like conveyors, branches, merges). By transferring this information into graphs and matrices, it is possible to get quantitative representations. In Figure 36, an example of a graph used in material flow can be seen. The details of the graph theory are not a part of this study, but it should be known that the main concepts of the graph theory are used to define the relationships between the components. More information about the graphs and their application with AML can be found in [LSH13].

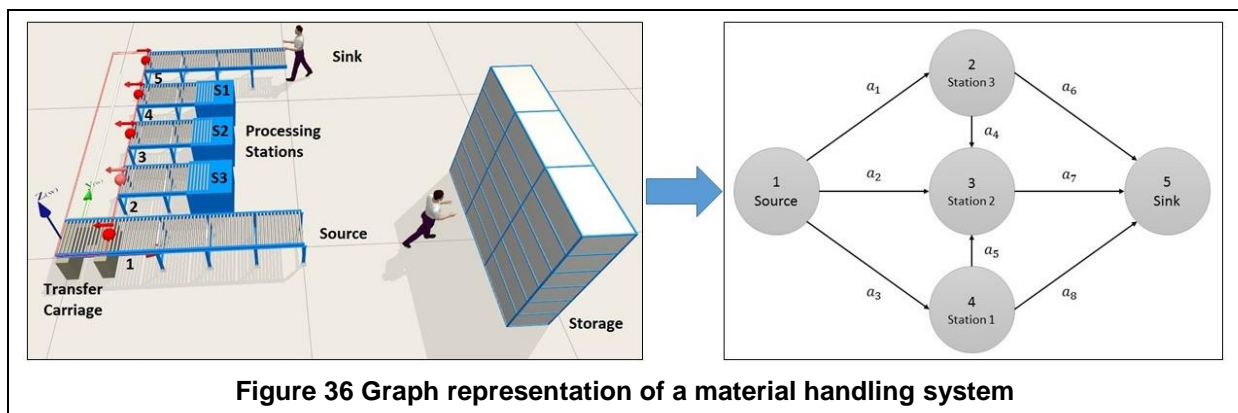


Figure 36 Graph representation of a material handling system

Analysis of a material handling system refers to qualitative and quantitative estimations. The quantitative estimations can be performed in the form of a simulation. In the simulation, the status changes in the material flow are calculated numerically gradually [Arn09]. The results can be displayed after the calculation (post runtime) or during the calculation (runtime).

So far, there is no common used standardization of the topology and component description for material handling technology. VDA automotive module box offers a specific solution for "Plant Simulation" tool, which is based on an XML-based format and is accessible only for VDA partners [May10]. Lack of standardization makes the further interpretation of the plant description difficult. Nevertheless, an existing AML export allows the systematic development of standardization [Dra13], which in principle is possible through the cooperation of other parties. Systematic approach creates first a mixture of standard and non-standard items in the

data model. The software vendors involved in the standardization process try to agree upon the functional roles and their attributes. This effort of creating a standardized data exchange solution for material handling will be explained in the following chapters.

6.4.1. Preparation of a standardized data exchange solution for material handling

As the solution should be implemented with AML format, an investigation of the standard definitions for material handling in AML whitepapers is a necessity. In AML whitepaper [AML13], primal default roles for material handling technology are described within the "extended role class library" as a proposed extension of base role library. Functional roles such as "Conveyor" or "Turntable", etc. are available in this context. However, these predefined roles are very abstract; they serve as a "marker" for components. They do not contain semantic information like component attributes or interfaces.

Such state of the role classes is not sufficient for the loss-free data exchange in the scope of material handling simulation. For example, attributes, which are common to many models, could be defined in the role class. A conveyor component has length, usable width, speed, and maximum load attributes in many different tools. An attribute, which is not included in the data model of a particular tool, may also be left blank. Thus, it is first created with a default value, and the real value can be entered by another tool later.

Resources in the material handling systems are defined in VDI 2411 [Ver701]. However, this standard guideline is extinguished without a successor. When creating a new role class library, the categorization in VDI 2411 was not used because the resources are classified according to mechanical properties in this document (e.g. Conveyors are categorized as belt conveyors, chain conveyors, band conveyors, etc.) This categorization is not adequate for early design phases because here only the transport functionality is considered not the technical implementation of it. Therefore, the approach of this work is to divide the elements according to their functionality. In modeling, e.g., the conveyor is generated as a functional role and load carrier is implemented as a second role with the name of "*ConveyorTechnology*". In the instance hierarchy, the conveyor object has a child object that takes over the role of the carrier.

In Figure 37, the role class library elements for material handling components can be seen. The first version of the material handling role class library has been created in cooperation with AML partners [Aut16] and is attached to the document in "Appendix A Material Handling Role Class and Interface Libraries in AutomationML". The application example of the library with explanation can be seen in chapter 7.1.

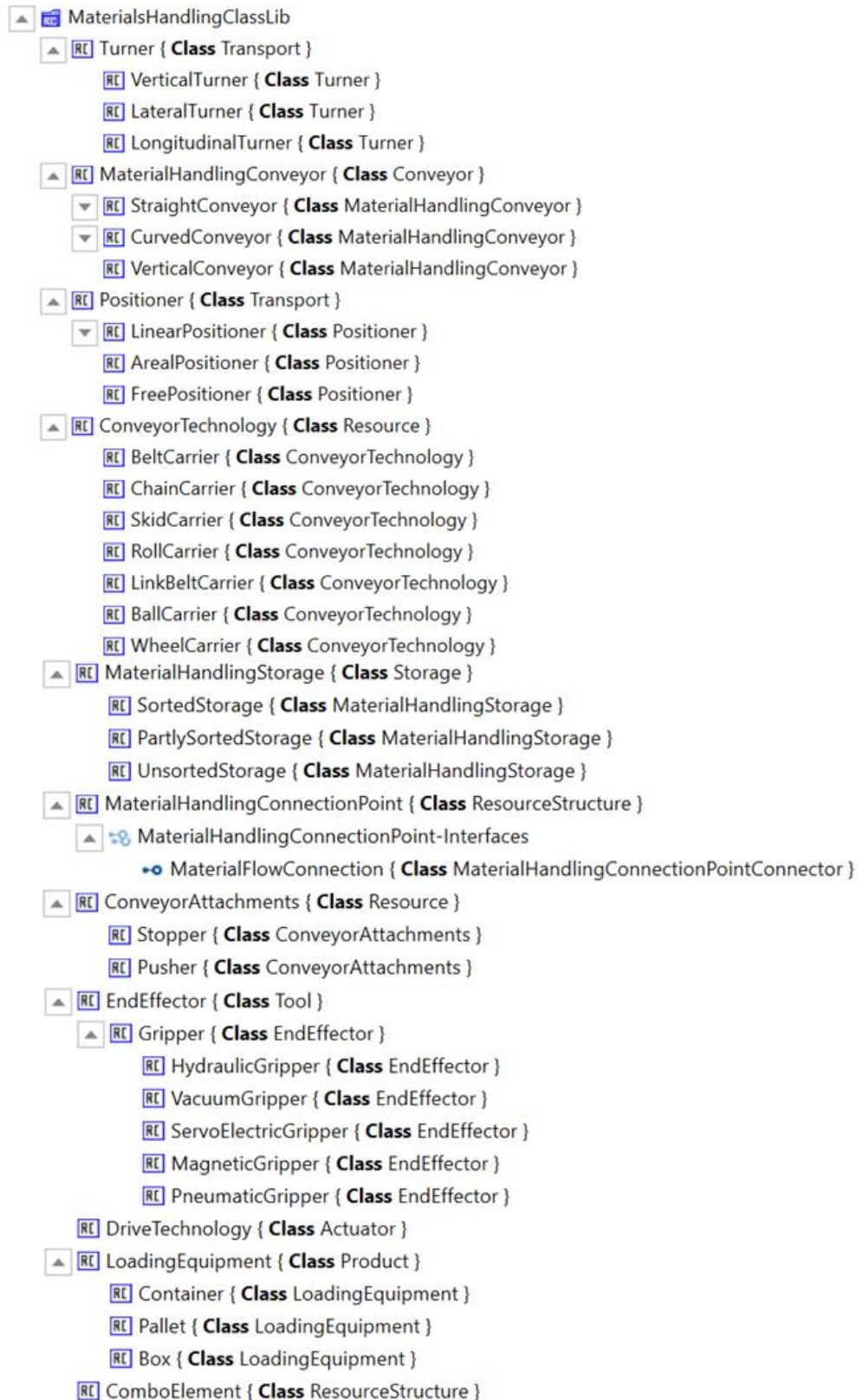
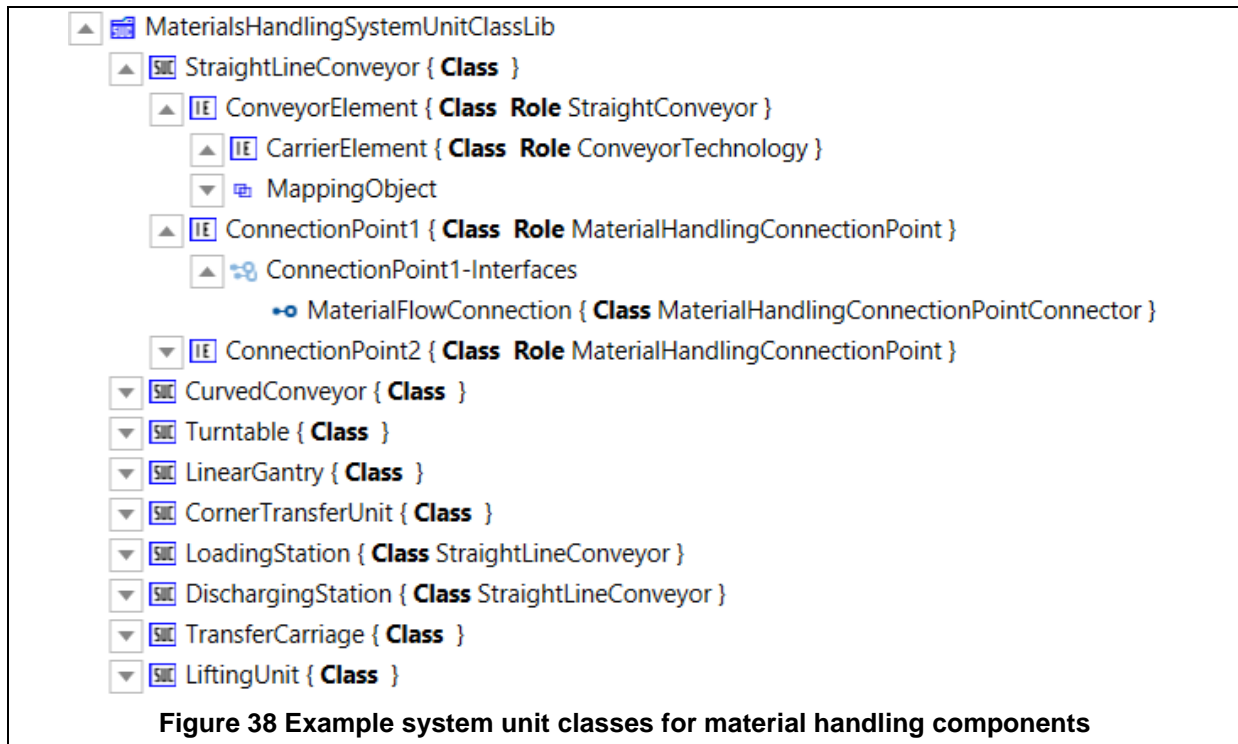


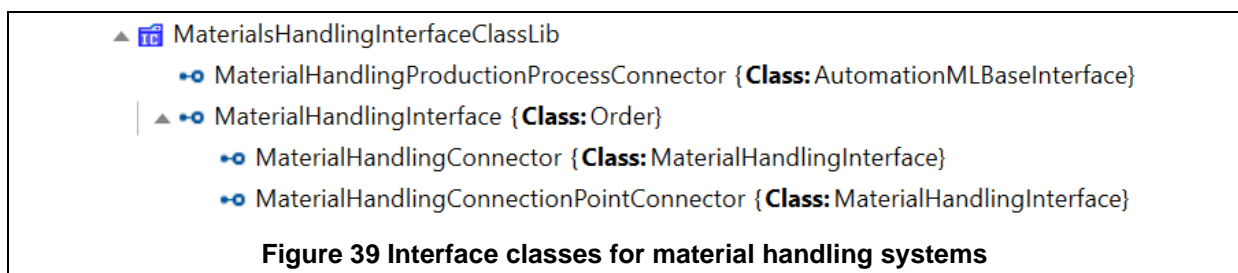
Figure 37 Role class library for material handling components

A System Unit Class library was created as an example approach, too. This library contains <SystemUnitClass>-objects that represent a pattern for an instance. A system unit class can consist of multiple internal elements, each of which plays a functional role. Attributes in the

system unit class are mapped by name (<MappingObject>) with the attributes in the role. Thus, different dialects can be adapted from the standard data model without having to make a name change. Examples of the system unit classes can be seen in Figure 38.

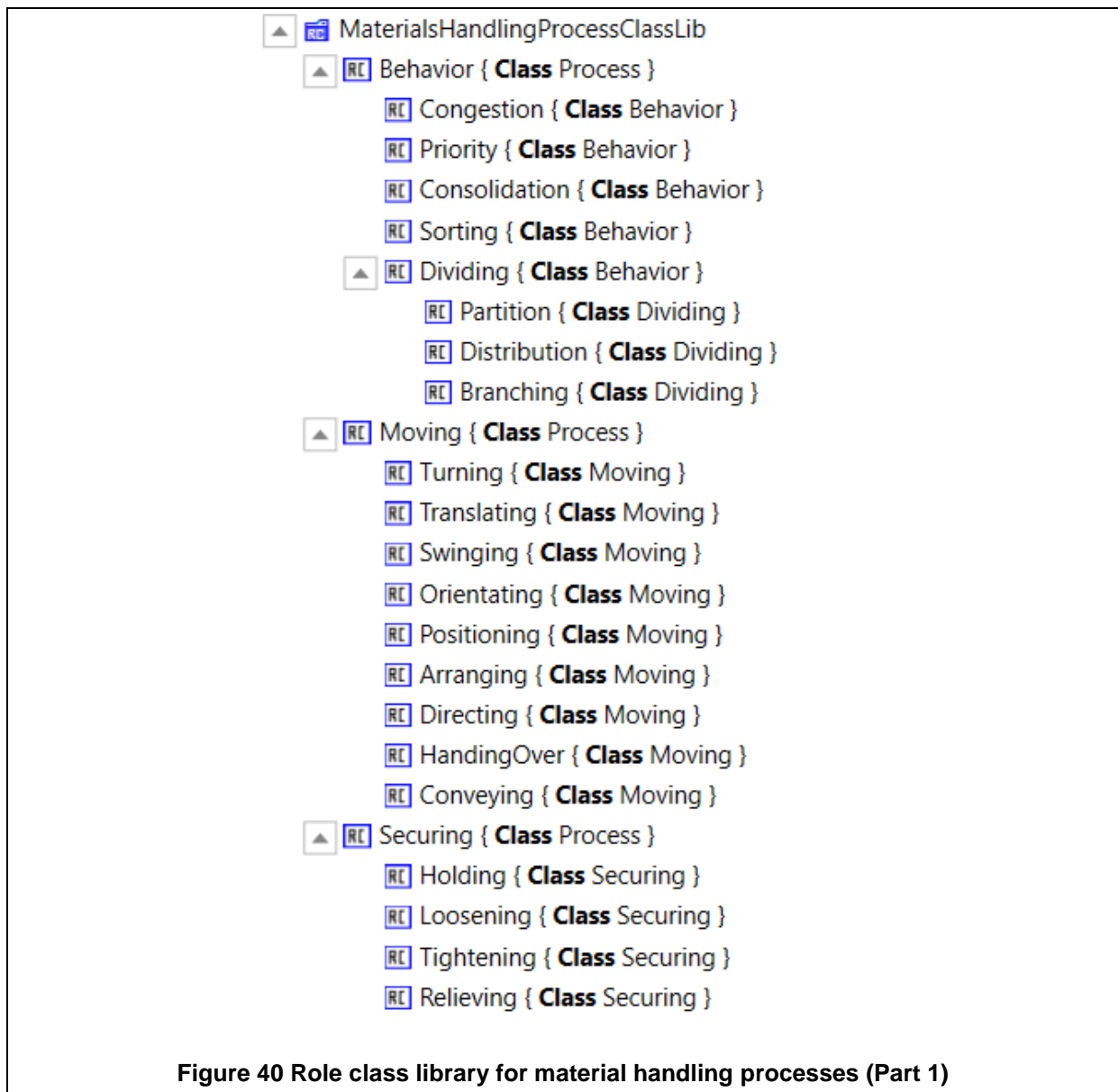


Finally, four new AML interfaces (<InterfaceClass>) were defined as seen in Figure 39. The first one is a base interface “*MaterialHandlingInterface*” for the linking of components in the material flow. It is derived from AML standard interface “*Order*” and inherits *Direction* attribute. *Direction* attribute defines if the connection point is used for input, output or could be both. There are two interfaces derived from this interface: “*MaterialHandlingConnector*” and “*MaterialHandlingConnectionPointConnector*”. “*MaterialHandlingConnector*” is designed to connect the components directly, without any port or connection point element. “*MaterialHandlingConnectionPointConnector*” connects the components through a connection point, as the name implies. The last interface in the library “*MaterialHandlingProductionProcessConnector*” is for the linking of material flow and manufacturing technology. So that, it is possible to combine these different aspects in the plant design.



6.4.2. Process description of material handling

Not only the resources but also processes are vital in the material handling technology. After VDI 2411 [Ver701] following function types are existing in a material flow system: machining, handling, conveying, storing, and placing. The guideline VDI 2860 [Ric90] defines only conveying, storage and handling as part of functions in material flow. It also defines and systematizes the handling to elementary and composite functions: saving, changing quantities, moving, securing and verification. In this work, a process role class library is also created according to the standard guideline VDI 2860. Nevertheless, some extra roles are also needed, e.g. roles for congestion behavior or roles such as source, sink, loading process, and discharging process. In Figure 40 and Figure 41, a list of created roles can be seen.



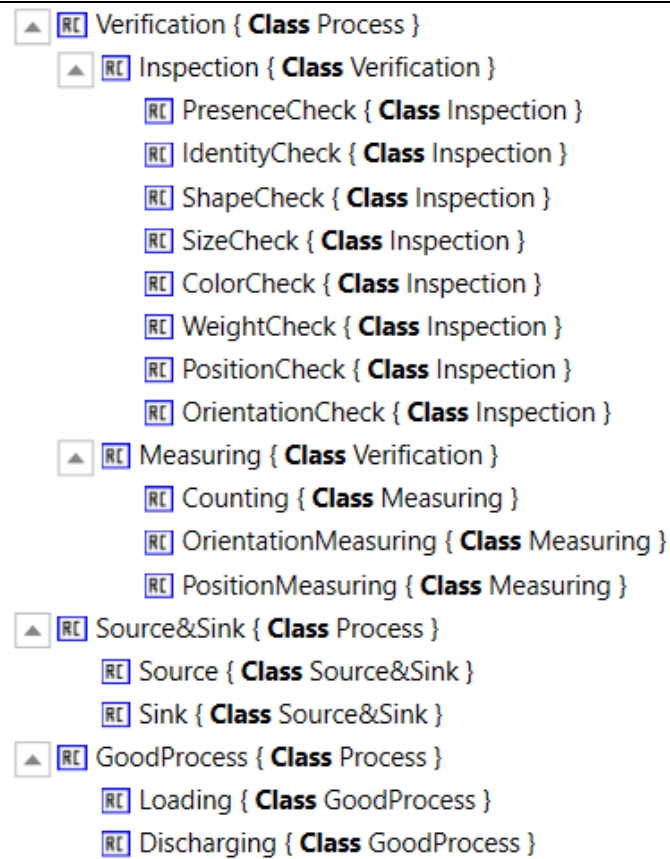
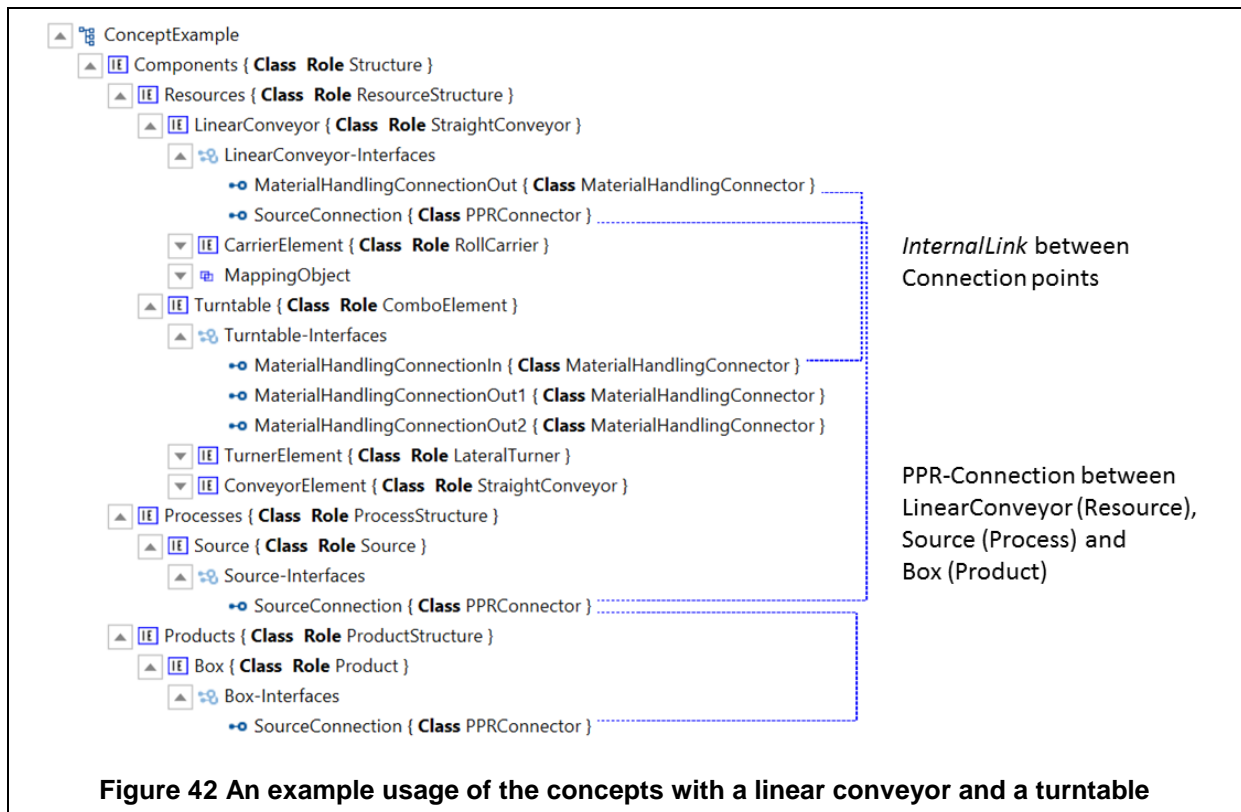


Figure 41 Role class library for material handling processes (Part 2)

6.4.3. An example usage of the concepts

The example in Figure 42 contains one linear conveyor and one turntable. The Linear conveyor is assigned to the material handling role class “*StraightConveyor*”. Turntable is a combination of a turner and conveyor element, which are assigned to the “*LateralTurner*” and “*StraightConveyor*” roles in the order. The role requirement “*ComboElement*” indicates that the turntable is a combination of multiple elements.



Conveyor elements contain a child element with the name “*CarrierElement*”, which is assigned to the material handling role “*RollCarrier*”. This role shows what kind of conveyor technology is used in the component.

The connection between the turntable and the conveyor is realized by the “*MaterialHandlingConnector*” interface class. The direction of the material flow can be identified through the *Direction* attribute of the “*MaterialHandlingConnector*”, which is inherited from base interface class “*Order*” (“*MaterialHandlingConnector*” is derived from “*MaterialHandlingInterface*”, which is derived from “*Order*”). The name of the interface class has been chosen correspondingly to show the material flow direction (In, Out).

The PPR interface from standard AutomationML interfaces is utilized to define the source for the good (“*Box*” in this example). In this case, the good (box) enters the system through the “*LinearConveyor*” element. A “*Source*” element, which is assigned to the role “*Source*” from material handling process class library, is used to define this process.

6.5. Summary

Implementation of a data exchange functionality for a layout planning tool is one of the main targets of this work. The preparation of the proprietary data models and exporting them in the standard neutral format AML is the first step to reach this goal.

3D visualization is an essential feature for the target tools, so it is first described how the geometry models converted between the native format X3D/VRML and exchange format

COLLADA. After that, important points for the modeling of physical properties are explained. The physical modeling is limited to rigid body physics.

The content of the material handling components in AML format is presented next. The standardization efforts play an essential role in this part. An introduction to role class and system unit class libraries, which are the results of the modeling and standardization, is given in this section.

Not only the components but also the material handling processes are a part of the modeling work. Standard guidelines and existing AML design concepts are of significant help for the description of the processes.

7. Implementation Concept

“Working software is the primary measure of progress.” –Agile Manifesto [Agi16]

This chapter introduces the implemented software solutions for the data exchange in material handling systems. There are two software tools, which are the subjects of the research. One of them is a layout planning tool taraVRBuilder, and the other one is a process visualization tool taraVRControl. They are both used to validate the AML and COLLADA import/export functionalities. taraVRControl is also the tool for visualizing the physics simulation and combining the physics simulation environment with the co-simulation. The implemented solutions in this work are in the form of plugin⁴ libraries, which enhance the existing software tools with AML data exchange functionalities.

7.1. Introduction to the software components of the digital factory

7.1.1. Layout planning tool

One of the application target of this work, the layout planning tool taraVRBuilder, is a 3D design software in the material handling, logistics and plant layout planning field. A facility layout can be configured from modules of a standard library. Besides the setting of production lines and constructional components for material handling systems, it is also possible to represent supportive elements, such as machines, vehicles, workers, etc. Animation of 3D scenes is the focus of taraVRBuilder, so it is possible to configure animation parameters of machines and facilities up to a medium complexity in a virtual 3D environment [tar15].

Although simulation is not a defined task for taraVRBuilder, there are existing functions to configure the modules for the strategy of distribution, accumulation capacity and measurement systems like distance measurement between selected points, throughput and counter clock recording, simple performance analysis and automatic creation of “Bill of Materials” (BOM). That means some of this information could be used as the basis for a simulation in the next phases of the toolchain, which makes data exchange an important function for the development of the tool.

7.1.2. Process visualization tool

The second target software tool, taraVRControl, is used to combine sensor data from real/virtual plants and installations with events of a 3D scene. In this way, it is possible to visualize a simulation of a virtual production system. Dynamic visualization components can also be set from an object library, and their properties can be coupled with process variables.

⁴ Plugin is a software component, which adds a specific functionality to an existing software [Cam16]. Also known as add-on.

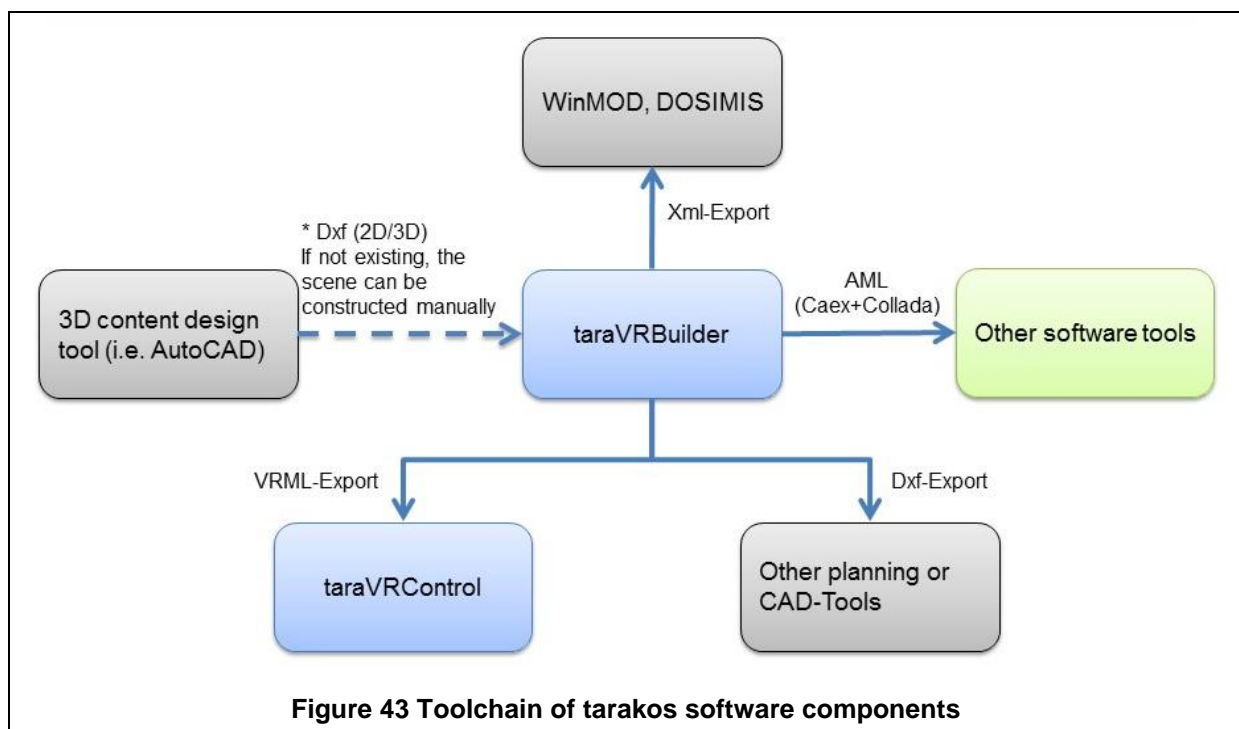
Information, which is relevant for the runtime component, is stored in an XML structure in a project file or can be directly written as a comment into the 3D file.

taraVRControl contains taraOPC2Control, an ActiveX component, to configure the 3D scene in an application which supports the integration of ActiveX components. This component enables the bidirectional communication between a 3D scene as an OPC client and an OPC server during runtime. Information, concerning the object characteristics, such as color, position, is read, and changes in the process variables are visualized in the 3D scene. There is also a plugin interface in taraVRControl, which enables integration of a user defined process connection or simulation [tar08].

7.1.3. Further development

tarakos software tools can import and export 3D geometry information from other tools. An interface to DOSIMIS also exists [Bös08]. Other than that, it was not possible to transport simulation relevant information into the other software tools from the later phases of engineering design.

Figure 43 shows the toolchain of tarakos software components, together with the new AML interface, which is the main subject and resulting product of this work.



The primary target is to transfer planning data of taraVRBuilder to other software tools like simulation tools, robot programming, PLC programming and let them enrich the information on the same format. This enriched data can be the basis of virtual commissioning or some other simulation-based analysis, which can be in turn visualized by taraVRControl as a runtime visualization component.

7.2. Software components for the data exchange

7.2.1. Selection of technology

Although it is not important which language to choose for programming, it is also apparent that some languages suit better to some applications. Nowadays, using several programming languages in a software system is also not seldom, which is called polyglot programming [Fje08]. An example of polyglot programming is combining a statically typed language like C/C++, C# or Java and a scripting language like Ruby or JavaScript. It is also possible to see many SQL, XML, or HTML as markup languages in modern software systems. The solution, which is introduced in this work, also consists of many programming languages and existing external library solutions, but the main structure is based on C#.NET, of which the reasons are introduced in this chapter.

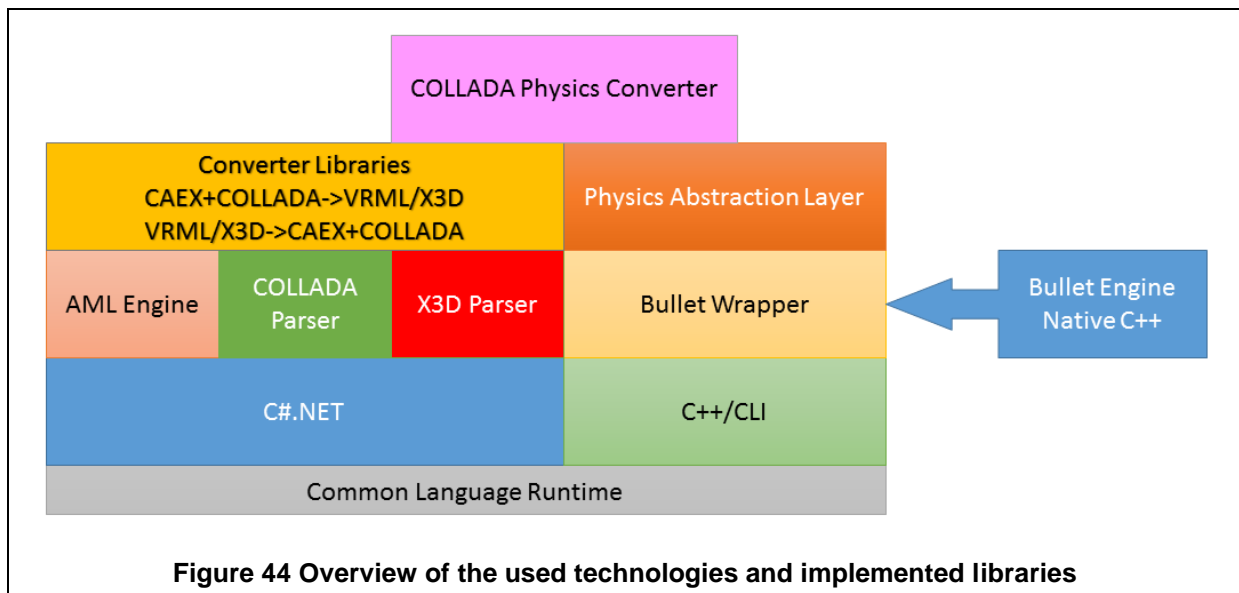


Figure 44 Overview of the used technologies and implemented libraries

Figure 44 shows how the technologies are utilized for the data exchange solution. The solutions are all based on .NET-Framework. Most of the libraries are created with C#.NET language, but the usage of C++/CLI was a necessity for the physics component. AML Engine and Bullet Engine are third-party libraries (developed and supplied by external communities). Other library components are the products of tarakos-internal development, using these technologies. The details of the used and developed technologies will be explained in the following chapters.

7.2.1.1. C# Language and .NET Framework

C# is a modern, object-oriented, and type-safe programming language. C# has its roots in the C family of languages. Microsoft's C# compiler (Visual Studio) for the .NET Framework is an implementation for both of these standards. Nowadays, software design increasingly relies on software components in the shape of self-contained and self-describing packages of functionality. These components present a programming model with properties, methods, and

events. The attributes supply declarative information about the component, and they can include documentation.

C# has a “**unified type system**”; that means all C# types, also primitive types such as `int` and `double`, inherit from a single root `object` type. Thus, all types have a set of common operations. Values of any type can be stored, transported, and operated upon consistently. C# supports both user-defined reference types and value types, which allows dynamic allocation of objects and inline storage of lightweight structures. Aspects of C#'s design includes the separate virtual and override modifiers, the rules for method overload resolution, and support for explicit interface member declarations [MC12].

The .NET Framework is an integral Windows component that is used to build the applications and XML services. According to [Mic16] the .NET Framework is designed to provide language interoperability across different programming languages, for example, class libraries written in C# can be used in VB.NET and vice versa. This framework consists of two main components: 1) The common language runtime, which is a language-neutral platform for application development and execution, including functions for memory management, thread execution, code execution, code safety verification, compilation, and other system services. 2) The .NET Framework class library, which is a collection of reusable types. They integrate with the common language runtime and have the function of helping application development ranging from traditional command-line or graphical user interface (GUI) to applications based on technologies like ASP.NET, Web Forms, and XML.

Visual Studio is a development environment to build ASP.NET Web applications, XML Web Services, desktop applications, and mobile applications. Visual C# language has access to the main technologies of .NET Framework through Visual Studio. For example, libraries that provide functionalities to parse and create XML-based documents, which are widely utilized for the data exchange implementation [Msc16].

7.2.1.2. AutomationML Engine

The AutomationML Association offers a free software library for developers - the AutomationML engine- under the terms of the GNU Library General Public License. AML-Engine reflects the CAEX data model in a C# class structure and contains utility classes and methods, to manipulate CAEX objects as well. The software developer is freed from validating the adherence to the CAEX schema by this library component. AML-Engine contains functions for importing data from other CAEX files, for cloning, replacing and copying CAEX data, specifying meta-data and advanced functions for splitting and merging of CAEX files. The integration of the engine to the proprietary software projects can be done by referencing the engine's .dll library files [Dra131].

7.2.1.3. Bullet Physics Engine

Bullet Physics is a professional open source C++ code for collision detection, rigid body, and soft body dynamics library. The library is free for any commercial use under the ZLib license. This engine supports following functions [Cou13]:

- Discrete and continuous collision detection. Collision shapes include concave and convex meshes and basic primitives.
- Rigid body dynamics constraint solver, vehicle dynamics, character controller and slider, hinge, generic 6 Degree of Freedom and cone twist constraint for ragdolls.
- Soft Body dynamics including constraint support

Bullet Engine is selected for the example application because it is open, well-documented and has already been used for many industrial/commercial projects. Soft body dynamics is not considered in the implementation, as the requirements defined within rigid body applications.

7.2.2. Architecture of the implemented components

In this chapter, the architecture of the data exchange libraries is presented. For the realization of AML data exchange functionality, several C#.NET libraries are used. The first library is an external library, the AutomationML-Engine, which is explained in the chapter 6.2.1.2. One self-implemented library is used to convert the inner structure of the proprietary software into CAEX instance hierarchy, and another one is to gather the inner structure back from the CAEX file. The other four libraries are used to convert the geometry between COLLADA and native geometry format (in this case X3D/VRML). The interactions between the libraries can be seen in Figure 45.

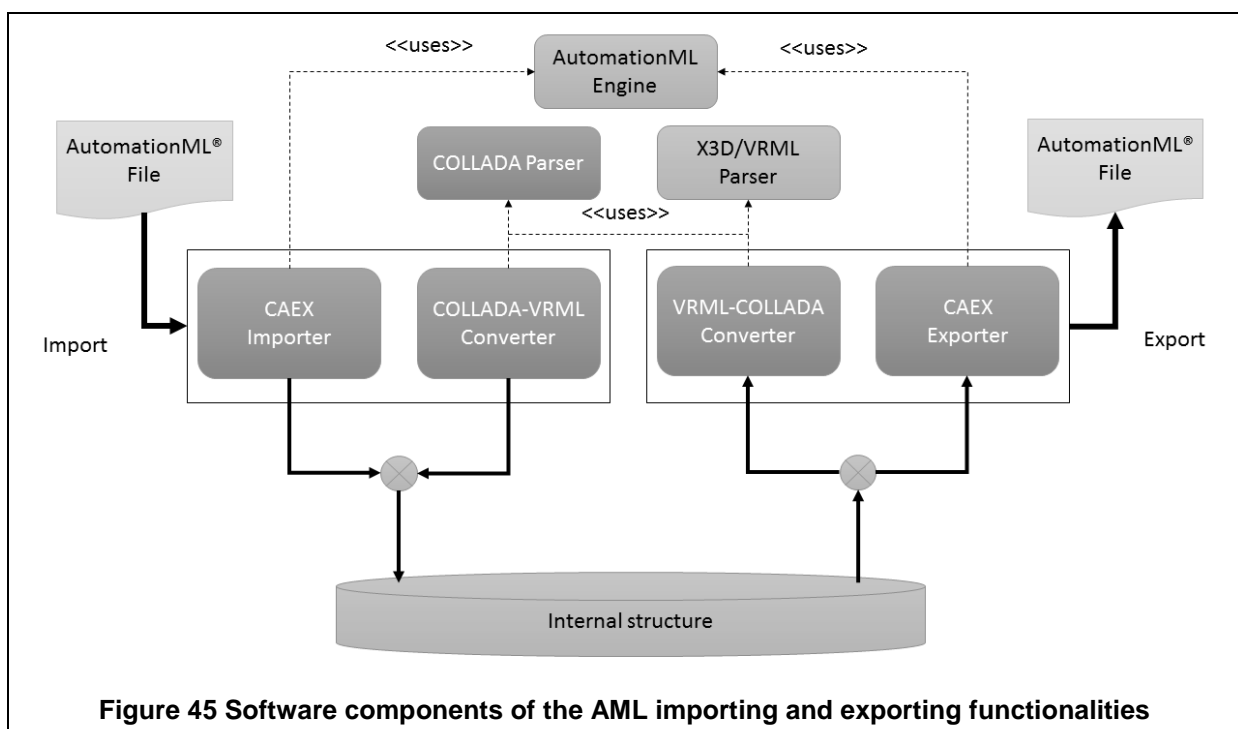


Figure 45 Software components of the AML importing and exporting functionalities

The import function parses the CAEX file with the help of AML-Engine, goes through the *Frame* attributes to position the objects. If an external COLLADA file is referenced, it is converted to the X3D/VRML format with the help of the implemented converter functionality.

The amount of the information outcome is more than importing by exporting. The hierarchy of objects is represented as instance hierarchy in CAEX format. A system unit class library, which contains the collection of proprietary object library components, is used for the classification of instance objects, which is referenced as an external file in the instance hierarchy file. The role classes are also referenced externally. The connection points are defined as CAEX <InternalLink> with the help of interface class "*MaterialHandlingConnectionPointConnector*". The position of the objects is written as *Frame* attribute in the instance hierarchy, and the geometry is converted to COLLADA via implemented software components and attached as an external reference in the file.

A challenging task in the implementation is the geometry conversion between COLLADA and native format X3D/VRML. COLLADA primitive geometry types, which are defined in the <mesh> element, can be transferred directly as indexed geometry types under <Shape> element in X3D. However, the indices of the coordinates, normal vectors, texture coordinates and color vectors should be calculated again because one to one convergence is not possible. Triangulation methods are used for X3D geometry types like <Box>, <Cone>, <Cylinder> and <Sphere> to represent them as triangle/polygon meshes in COLLADA. Effect properties as color, ambience and transparency are parallel in <Appearance> element in X3D and <library_materials>, <library_effects> elements in COLLADA. Even so, the differences in the representation should be regarded.

The sequence of the export function is:

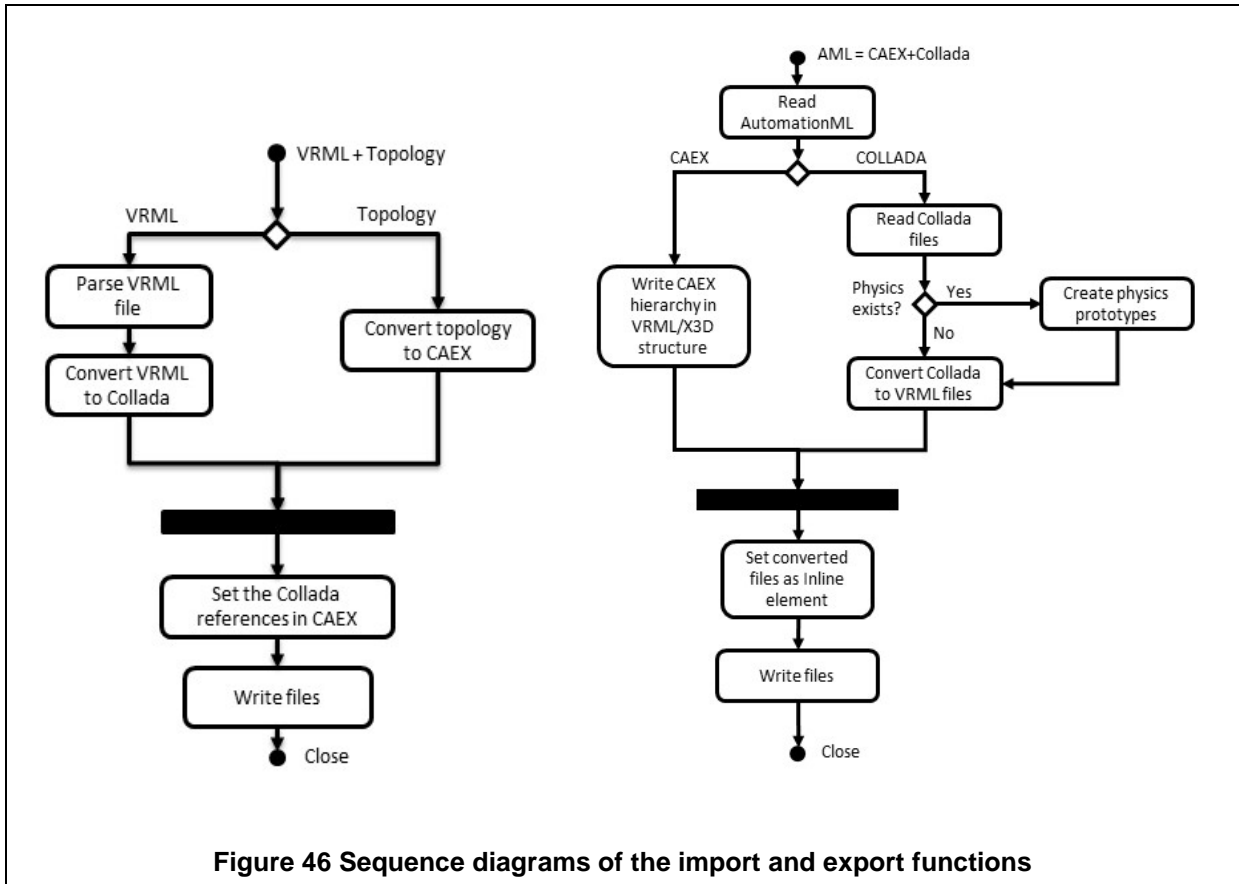
1. Exporter library reads the internal data structure of the taraVRBuilder database.
2. The internal data structure and process parameters are transformed into the CAEX elements with the help of the AML engine.
3. The geometry information is sent to the X3D/VRML parser.
4. The conversion of X3D/VRML to COLLADA is triggered.
5. Parallel to the COLLADA conversion, external references are added in CAEX.
6. The result is written as three AML files. A file for the instance hierarchy, another for the role class and system unit class libraries and the last one for the COLLADA geometry.

The sequence of the import function occurs similar but the other way around:

1. taraVRControl reads the AML file and sends it to the importer library.
2. The CAEX topology is transformed into the internal data structure with the help of the AML engine.
3. The COLLADA data is sent to the COLLADA parser.
4. If physics libraries in COLLADA exist, then <ExternProtoDeclare> statements (see Chapter 5.3.4) are created from the physics elements.

5. The conversion of COLLADA structure to X3D/VRML is triggered.
6. The results of conversion -X3D/VRML files- are added as <Inline> elements into the X3D/VRML file which contains the topology.

The sequence diagrams are shown in Figure 46.



7.2.3. Development of software libraries

The software libraries are implemented as C#.Net projects. A project contains the source code files, resource files such as external AML files, references to external libraries that the program depends on, and configuration data such as compiler settings. When a project is built, the C# compiler creates an executable assembly by using the files in the project [MC16]. This assembly can be used as a plugin for tarakos software kit if it implements a specific interface (*IX3DImporter* or *IX3DExporter* with respect to the implemented functionality).

7.2.3.1. COLLADA Parser

Parsing is the process of reading a document and providing an interface to the user application for accessing the content of the document [LiC09], in this context a COLLADA document, which is an XML document. The COLLADA parser is a software library that accomplishes these tasks and validates the COLLADA document with respect to XML Schema.

COLLADA parser library consists of three main functional blocks: COLLADA element catalog, which is a collection of POCOs (Plain Old C# Object), representing the COLLADA elements

in the specification. There are also the reader classes, which are responsible for parsing the XML document in a valid form. The writer classes save the actual status of the COLLADA elements as a COLLADA document after processing.

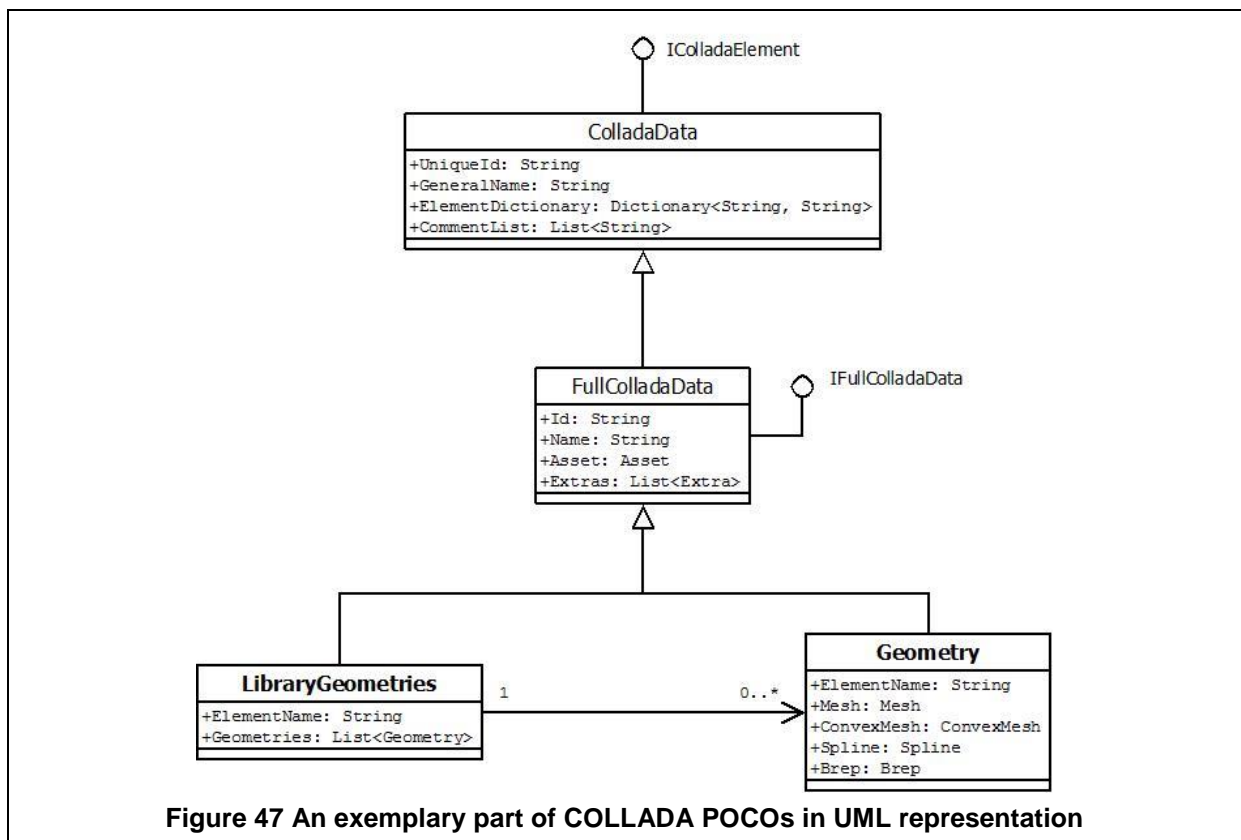


Figure 47 shows an exemplary implementation of COLLADA elements as UML class diagram. All COLLADA classes are derived from the abstract class *ColladaData*, which implements the *IColladaData* interface. This abstract class has some fundamental properties like the unique id of the instance, the name of the COLLADA element, a dictionary for child elements (if there is any) and a list of XML comments as they are not relevant for the process but contains possible useful semantic information. *FullColladaData* is also an abstract class, which is representing any COLLADA element which has an *id*, a *name* as attributes, an *<asset>* element and a list of *<extra>* as child elements. Both *<library_geometries>* and *<geometry>* elements satisfy these conditions, so the corresponding classes are derived from *FullColladaData* class. *LibraryGeometries* contains a list of *Geometry* instances parallel to the XML schema definition, which can be seen in the UML diagram as an association relationship. *Geometry* class contains the geometric element types, which are supported in COLLADA, e.g. Mesh, Convex-Mesh, Spline, BREP. Only one of the geometry elements can exist in a *Geometry* class instance as it is defined in the COLLADA specification [Bar08].

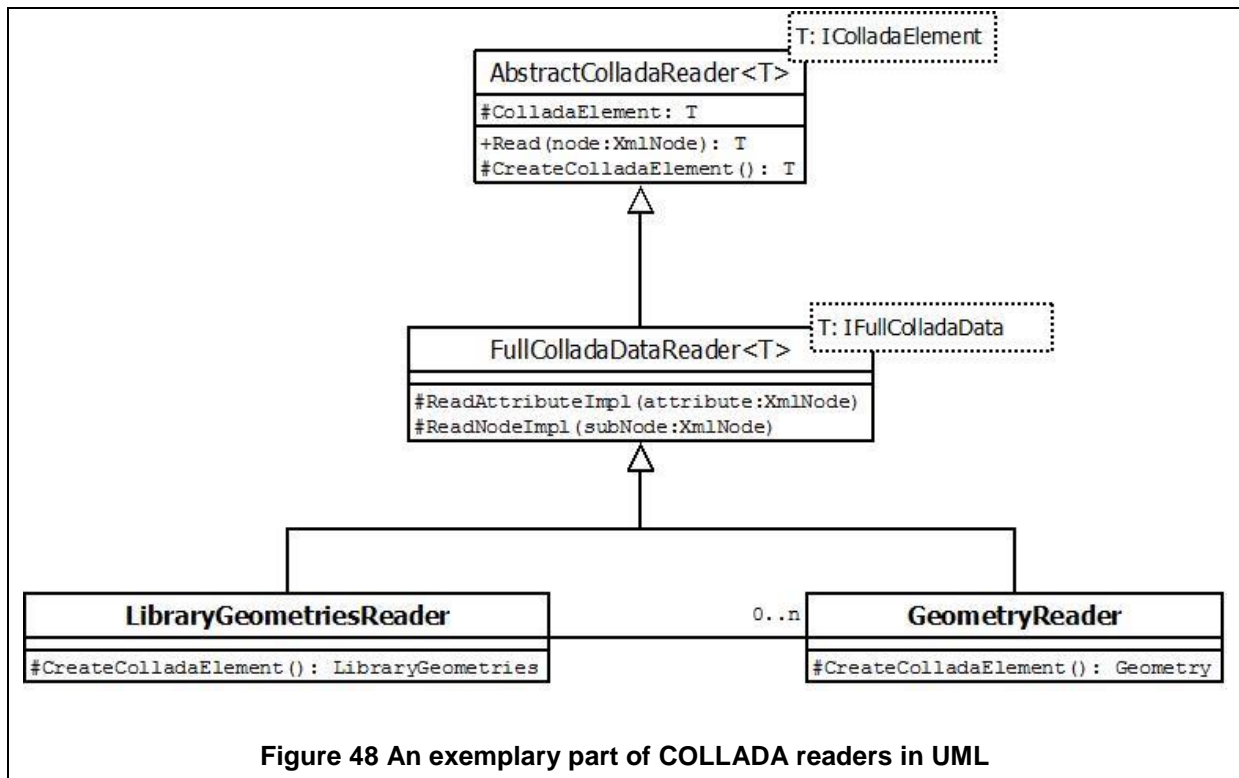
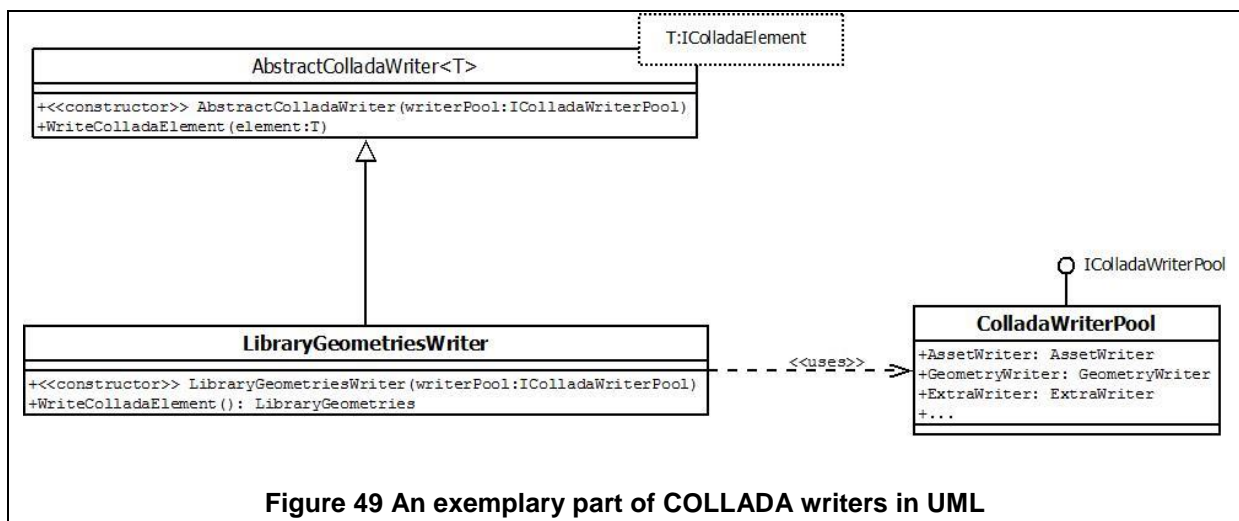


Figure 48 shows how the structure of COLLADA reader classes is constructed similarly to the COLLADA element structure. *AbstractColladaReader* is the parent of all reader classes, which has a generic parameter with the type of *IColladaElement*. *FullColladaDataReader* is an abstract class, which is providing methods for all *IFullColladaData* types. *LibraryGeometriesReader* is the reader class for <library_geometries> element, which creates instances of *GeometryReader* class to gather the <geometry> element information.



As seen in Figure 49, COLLADA writer classes have a little bit different structure as the readers. *AbstractColladaWriter* is the parent of all writer classes, which also have the generic parameter *IColladaElement*. However, other than the readers, all instances of writer classes are managed in a writer instance pool *ColladaWriterPool*. *ColladaWriterPool* ensures that no

writer class instance is created if it is not necessary. Every writer class gets the writer pool as a parameter in constructor during the creation. For example, when *LibraryGeometriesWriter* needs to write a <geometry> element, it fetches the *GeometryWriter* instance from the instance pool. After creating the whole document structure, a class named *ColladaDocumentCreator* gets the list of COLLADA elements and sends them into the method *WriteColladaElements()* to save the entire COLLADA document in a file.

7.2.3.2. X3D/VRML Parser

X3D/VRML parser is one of the core libraries of the tarakos software tools. This library plays a primary role for the conversion tasks. It is the software component, which is used to read, edit, and create the native format.

VRML and X3D have very similar structures, the most important difference is VRML is a plain text format, and X3D is an XML format. However, after parsing the data structure, individual elements can be handled uniformly. X3D has more elements than VRML, as VRML not supported anymore but X3D is still actual as the official successor of the VRML.

Actually, the X3D/VRML-parser library was not designed for the conversion tasks. Therefore, some enhancements were necessary to prepare the library for such operations.

The core of the parsing occurs in *SceneGraph* class, which has corresponding methods to read VRML or X3D files. *SceneExporter* class can write the data structure either in VRML or X3D format.

An important enhancement for the parser, which is developed during this work, is *NormalCalculator*. The *NormalCalculator* is used to calculate coordinates and indices of the normal vectors. A normal vector is a vector perpendicular to a surface. Normal vectors are necessary for lighting calculations and other rendering effects such as normal and bump mapping. It is possible that some objects do not have any normal or faulty defined normal vectors after conversion. For such cases, *NormalCalculator* calculates the normal vectors using polygon coordinate list and coordinate indices as a parameter. The functionality depends on these simple equations:

According to [Lea16] assuming a polygon has vertices:

$$P = \{(x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y_2, z_2)\}$$

Then the two vector lying on the same plane as the polygon can be found with:

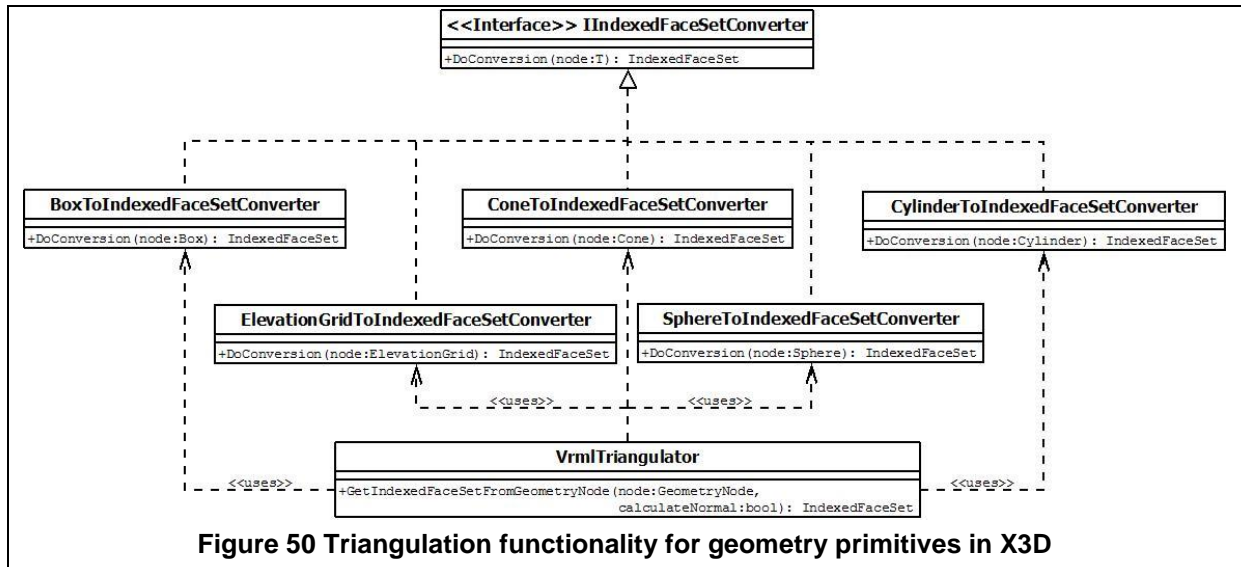
$$V1 = [x_1 - x_0, y_1 - y_0, z_1 - z_0]$$

$$V2 = [x_2 - x_0, y_2 - y_0, z_2 - z_0]$$

Polygon normal vector can be calculated as:

$$N = V1 \times V2$$

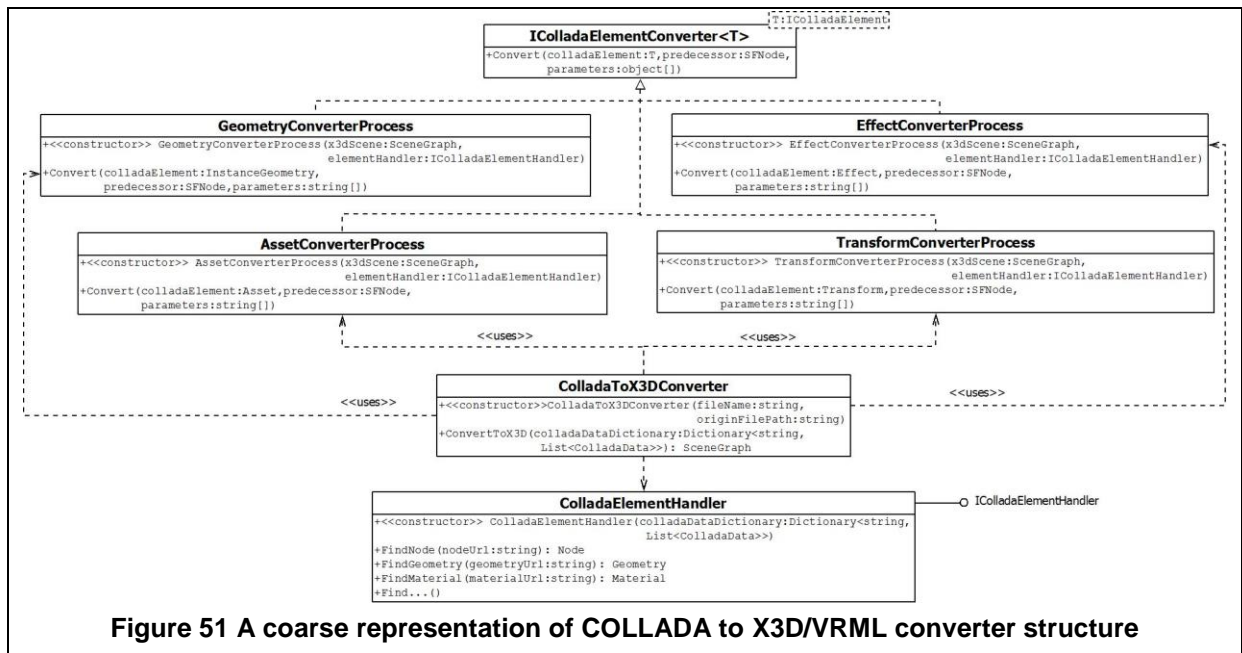
Second enhancement for the parser is the utility classes for the triangulation. As mentioned in section 5.2, triangulation of some geometry primitives is necessary for the conversion. `VrmlTriangulator` class can triangulate these geometry primitives to indexed meshes. The mathematics behind the triangulation methods will not be explained in this work, for more information about triangulation techniques [Zim05] can be read.



As seen in Figure 50, an interface `IIndexedFaceSetConverter` defines the main functionality for conversion between geometry primitives like box, cone, cylinder, elevation grid and sphere to `IndexedFaceSet` element. The `IndexedFaceSet` node is a 3D shape formed by polygons in X3D schema. It defines indices and coordinates of vertices to specify the polygonal faces [Web13]. `VrmlTriangulator` class chooses the convenient converter and executes the triangulation for the `GeometryNode`, which is a base class for geometry primitives.

7.2.3.3. COLLADA to X3D/VRML Converter

This library has the function to transmit the topology, geometry, effect, and all other kinds of information in COLLADA to X3D/VRML structure.

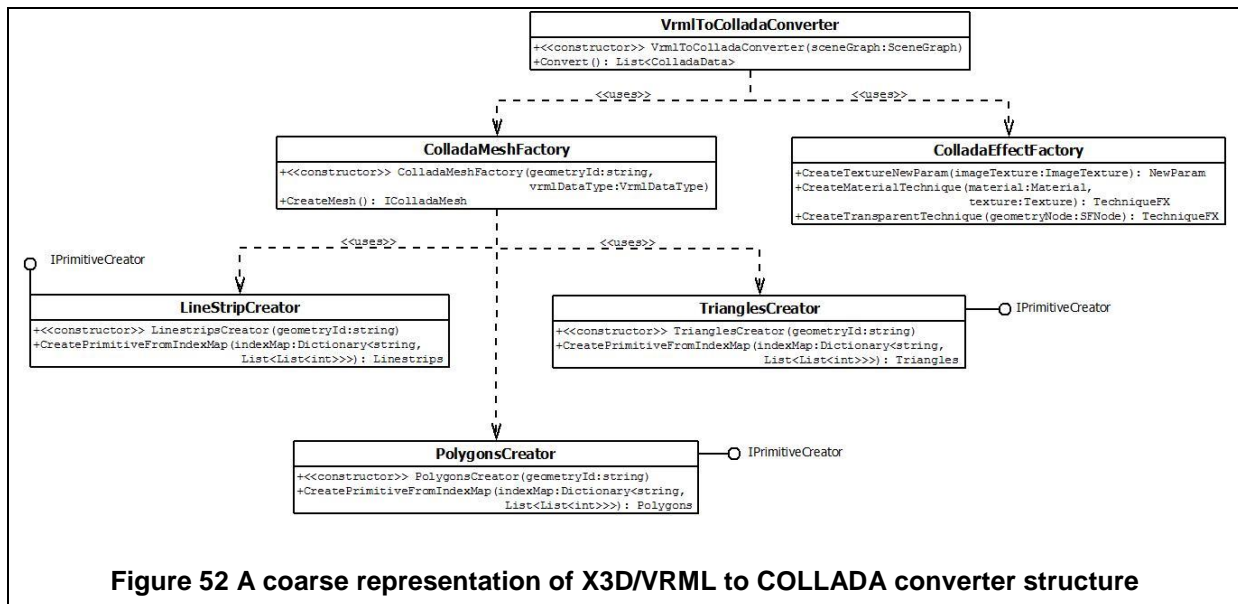


In Figure 51, a rough representation of the converter can be seen. The core of the operation is the *ColladaToX3DConverter* class. The file name of the new document and the file path of the origin document (to get the possible external references if there is any) are given as parameters to the constructor. *ConvertToX3D()* method gets a dictionary of COLLADA libraries and executes the conversion which is resulting the new X3D document with the given file name. *ColladaElementHandler* class handles the operation of finding the referenced COLLADA element. For example, a <geometry> element is referenced by its id in the topology tree, then *FindGeometry()* method finds the real *Geometry* instance with the given id. The conversion of COLLADA nodes to X3D nodes occurs in process classes, which implement the interface *IColladaElementConverter* with the generic parameter *IColladaElement*. There are more converter processes than seen in the diagram, but it is not necessary to show all of them, as they all function similarly. For example, the *GeometryConverterProcess* converts the <geometry> element of COLLADA to <Shape> element of X3D, and so on.

7.2.3.4. X3D/VRML to COLLADA Converter

This software component converts the X3D/VRML structure to COLLADA libraries. As X3D/VRML standard does not contain kinematics and physics like COLLADA⁵, it is used to transfer only topology and geometry information as COLLADA format. As a matter, it is theoretically possible to convert proprietary X3D physics prototypes back again in COLLADA. However, such a solution would be only available for intern-usage, as prototype elements are not standardized, and there is no use-case for such an operation in the scope of this work.

⁵ X3D has actually rigid body physics elements in the standard, but they are not used in our software tools and most of the visualizer has no support for them.



As seen in Figure 52, *VrmItoColladaConverter* is the core of the conversion functionality. In the constructor, a X3D/VRML *SceneGraph* instance is given as parameter, and through *Convert()* method, it converts the elements in X3D/VRML scene graph into COLLADA libraries. The *ColladaMeshFactory* class is responsible for the geometry conversion. It gets a geometry element as *VrmlDataType* instance and finds the suitable creator class. COLLADA primitive geometries are created in classes, which implement the *IPrimitiveCreator* interface. After the creation of COLLADA geometry, *ColladaEffectFactory* class is used to generate the COLLADA effects due to the effect parameters in X3D. For example, if converter finds an *<IndexedFaceSet>* element in the scene graph, it sends this element into *ColladaMeshFactory*. Mesh factory decides that the suitable creator for this element is *PolygonsCreator*. It gets/creates an instance of the appropriate creator and provides the COLLADA element (in this case *<polylists>*) back to the *VrmItoColladaConverter*. *VrmItoColladaConverter* uses *ColladaEffectFactory* to add the visual effect elements into the COLLADA geometry.

7.2.3.5. Physics Abstraction Layer

Physics abstraction layer is the connection between physics engine(s) and proprietary software libraries. The prototype implementation is a result of company-internal development for project AVANTI [AVA16] and has been done with the Bullet engine, which is written in C++ language. Bullet wrapper is a wrapper library, which makes it possible to use native C++ functions in .NET framework. A wrapper library can be defined as an additional software layer of abstraction, which translates an existing foreign interface into a compatible interface (mostly) without adding any new functionality [Lar08]. The technology used for wrapping is called CLI (Common Language Infrastructure), and C++/CLI is a .NET language that is similar to the native C++ and can be used as a reference library in C#. Additional to Bullet wrapper, a higher abstraction level as an intermediate layer has been implemented to allow other or multiple physics engine connections in the future (see Figure 53).

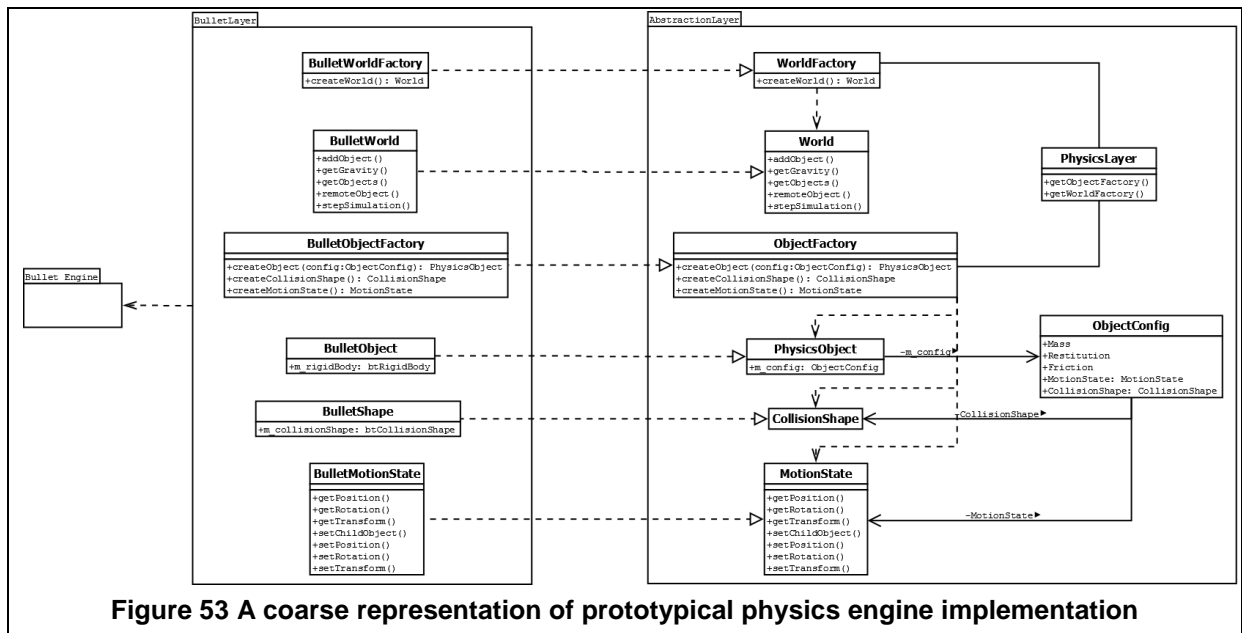


Figure 53 A coarse representation of prototypical physics engine implementation

This software component serves as a plugin for the taraVRControl software, but theoretically, it is possible to separate them and use it in external software tools, too.

7.2.3.6. COLLADA Physics Converter

COLLADA physics converter component is an implementation of *IColladaElementConverter* interface, which is already familiar from Chapter 7.2.3.3. The reason, why it is separate from COLLADA to X3D/VRML-converter library, is the dependency from physics abstraction layer, which is not necessary to have in the plain converter in all use-cases. The dependency structure of the library is illustrated in Figure 54.

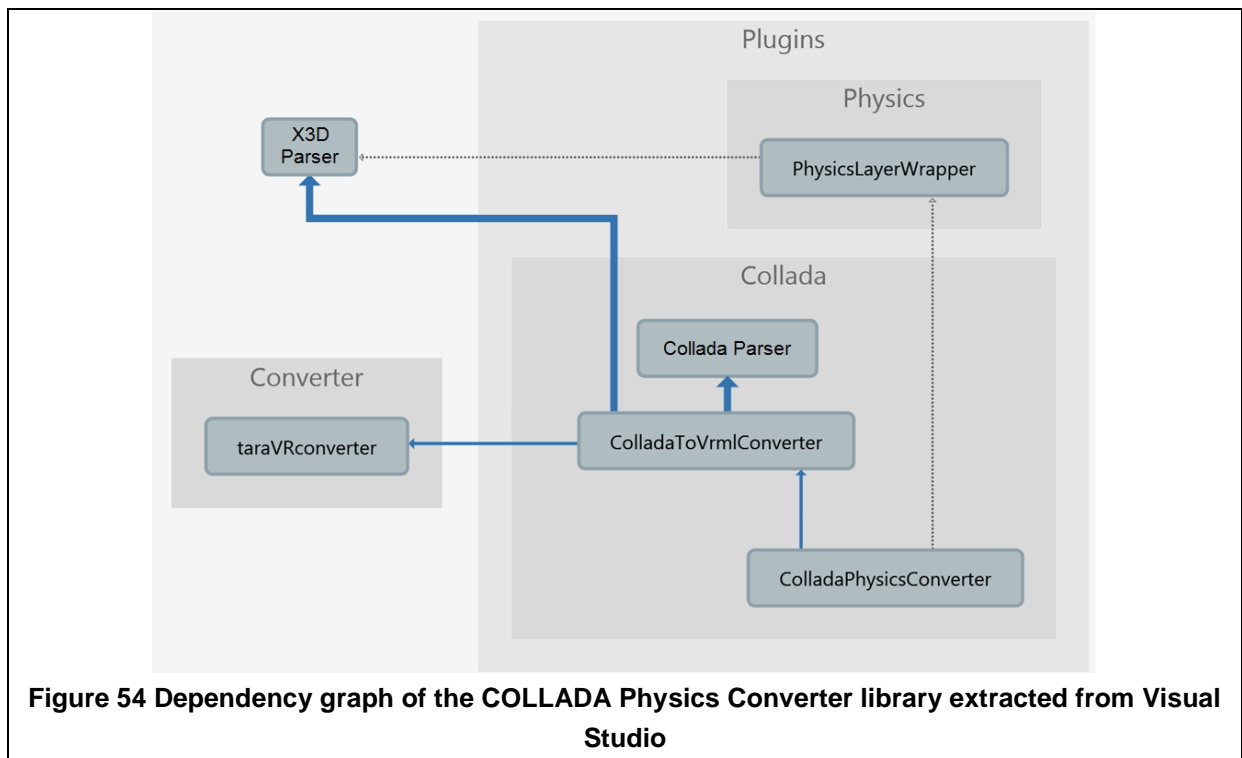


Figure 54 Dependency graph of the COLLADA Physics Converter library extracted from Visual Studio

The objects of the physics scene are implemented as <ExternPrototype> elements in the X3D/VRML scene graph, which allows taraVRControl software tool to handle the physics related functions like communication with the scene, saving/loading the status, and similar interactions. Abstract physics layer can load a scene into the physics engine, which has been defined as external prototypes in the scene graph. COLLADA physics converter library has the function to convert the COLLADA physics elements to these external prototypes.

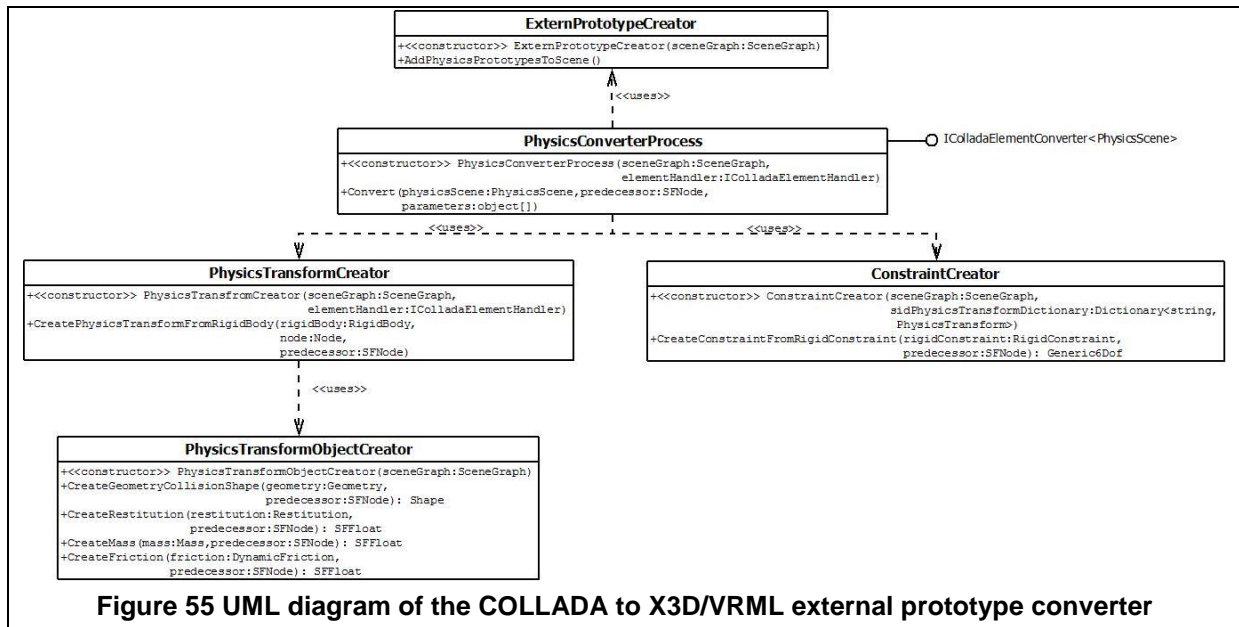


Figure 55 shows that the *PhysicsConverterProcess* class is the core element of the library. In *Convert()* method a <physics_scene> in COLLADA is converted into a X3D transform object. Each <rigid_body> object in the physics scene are transferred to <PhysicsTransform> prototype in *PhysicsTransformCreator* class, which uses an instance of *PhysicsTransformObjectCreator* class to gather necessary elements like collision shapes and properties like mass, friction, restitution parameters. After the objects in physics scene are created, the constraints are prepared through an instance of the *ConstraintCreator* class, which is triggered through *PhysicsConverterProcess*.

7.2.3.7. AutomationML Exporter

AutomationML exporter library uses X3D/VRML to COLLADA converter and external library AML-Engine to export an AML file with COLLADA references from taraVRBuilder. Geometry information in taraVRBuilder is saved as VRML file, and all other relevant information is transferred in object collections (i.e. *Dictionary* or *List* classes in C#) to AML Exporter. AML Exporter iterates through the all elements in these collections and creates the instance hierarchy in CAEX. When a geometry reference is present, it gets the VRML file, converts it to COLLADA with the help of X3D/VRML to COLLADA converter and attaches it as a reference into the internal element.

AML Exporter also has a semantic evaluation layer, which has the information of all possible internal elements. This semantic evaluation layer assigns the roles and system unit classes to the internal elements, creates the frame attributes from position information and makes the attribute mapping between internal element attributes and role attributes.

After the creation of internal element hierarchy, the AML Exporter assigns the internal links between the interfaces.

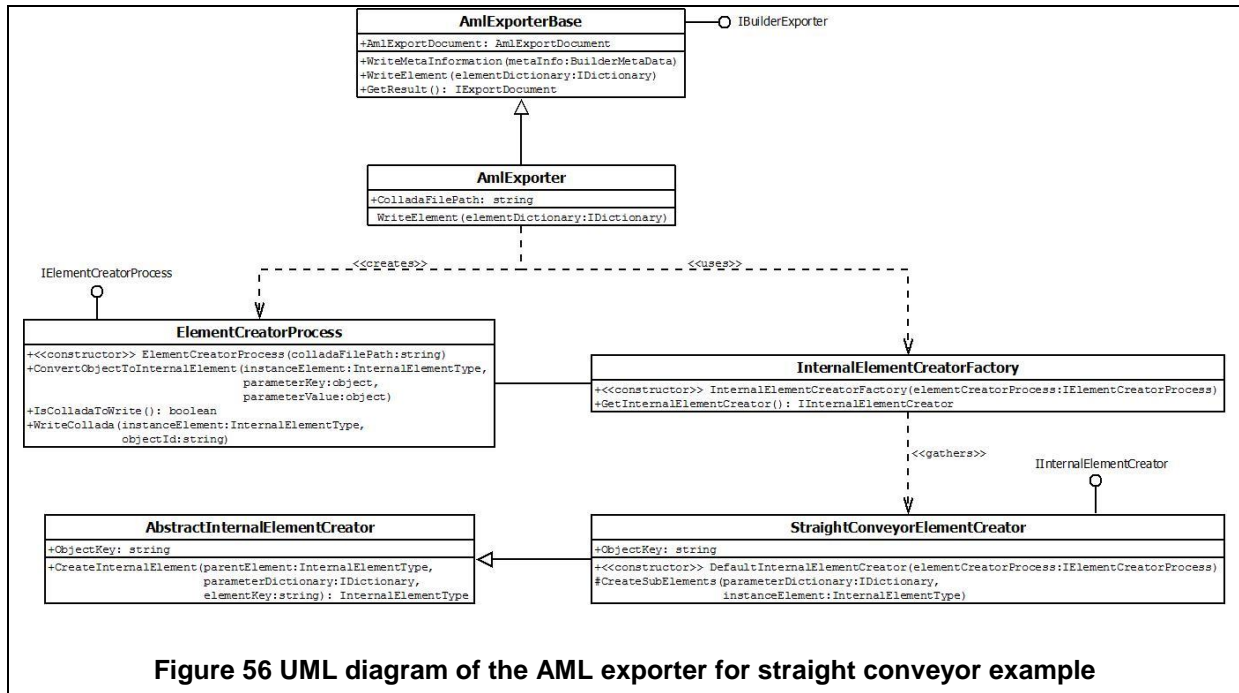
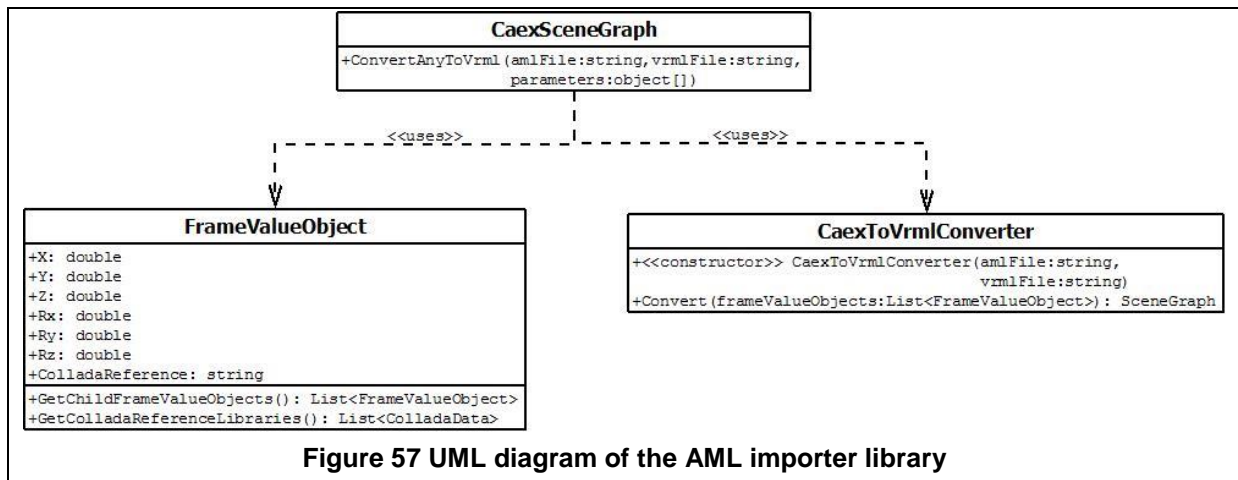


Figure 56 represents how the AML exporter functions. *AmlExporter* class gets a dictionary of proprietary data objects. It creates an instance of *ElementCreatorProcess* which contains some main functions like building the raw internal element or writing the COLLADA reference. This process class is not used by *AmlExporter*; it will be passed over to *InternalElementCreatorFactory*. *InternalElementCreatorFactory* contains a mapping of element creator classes which implement *IInternalElementCreator*. Every object type in taraVRBuilder has one corresponding element creator. In this example, a straight conveyor object has the type key i.e. “*Gerade_Multistau*”. Creator factory knows that for this type key the *StraightConveyorElementCreator* class is responsible, and triggers the associated methods from the class instance to create the internal element of this type with all assignments of system unit class, role class, and attribute mappings.

7.2.3.8. AutomationML Importer

AutomationML importer library is used to create a scene graph in the taraVRControl tool. It is capable of locating the object positions and convert the COLLADA references into X3D/VRML format with the help of COLLADA to X3D/VRML-converter library. A semantic evaluation of the AML content is not existent yet. Still, the main function of the taraVRControl tool is the

visualization and connection with physics-engine so the transfer of the position, geometry and physical properties of components is enough for the defined task.



In Figure 57, it can be seen that the *CaexSceneGraph* class has the main functionality to convert AML to X3D/VRML structure. *ConvertAnyToVrml()* iterates through the instance hierarchy and puts the position information and COLLADA references into *FrameValueObject* instances. *FrameValueObject* contains a list of its type, which allows representing a hierarchical structure. After the iteration through the instance hierarchy, all *FrameValueObject* instances are transmitted to *CaexToVrmlConverter*, which creates the scene in X3D/VRML structure. All COLLADA references are converted into VRML or X3D files, and the topology is established in a container file, which contains references to new created X3D/VRML files with <Inline> element.

7.3. Summary

This section explains the core functionalities of the software components, which execute the data exchange functions. An introduction to the target software tools, tarakos software tool suite, is provided at the beginning to understand how the data exchange components work together with the main software tool.

After that, the technology, which is useful for the development, is introduced, as it is a part of the implementation. The software solution consists of several software libraries, so the architecture shows how they combined with each other to realize the data exchange solution.

The software libraries can be categorized into three main functional blocks:

1. Converter libraries for geometry models, which are responsible for importing and exporting between the native format X3D/VRML and COLLADA
2. Libraries for physical models, one is to import COLLADA rigid body physics libraries to X3D/VRML prototype structure and the other one is to connect the visualization tool with the physics simulation.

3. AML converter libraries for the material handling components, which import/export the topology of the elements and makes the mapping of material flow/process defining elements, interfaces, and/or attributes.

All these libraries are implemented as add-ons for existing software tools, but some parts can be used independently for possible other requirements, as they are developed separated and inserted into the main tool through interfaces. AML libraries are related strongly to the semantic content of the proprietary tools, but COLLADA to X3D/VRML converter or physics-engine-layer are such products, which can be used flexibly for any other compatible tool.

8. Application

*“My friend, all theory is grey, and green
the golden tree of life.” - Johann Wolfgang von Goethe in ‘Faust’*

*“The philosophers have only interpreted the world, in various ways. The point, however, is to
change it.” – Karl Marx in ‘Eleven Theses on Feuerbach’*

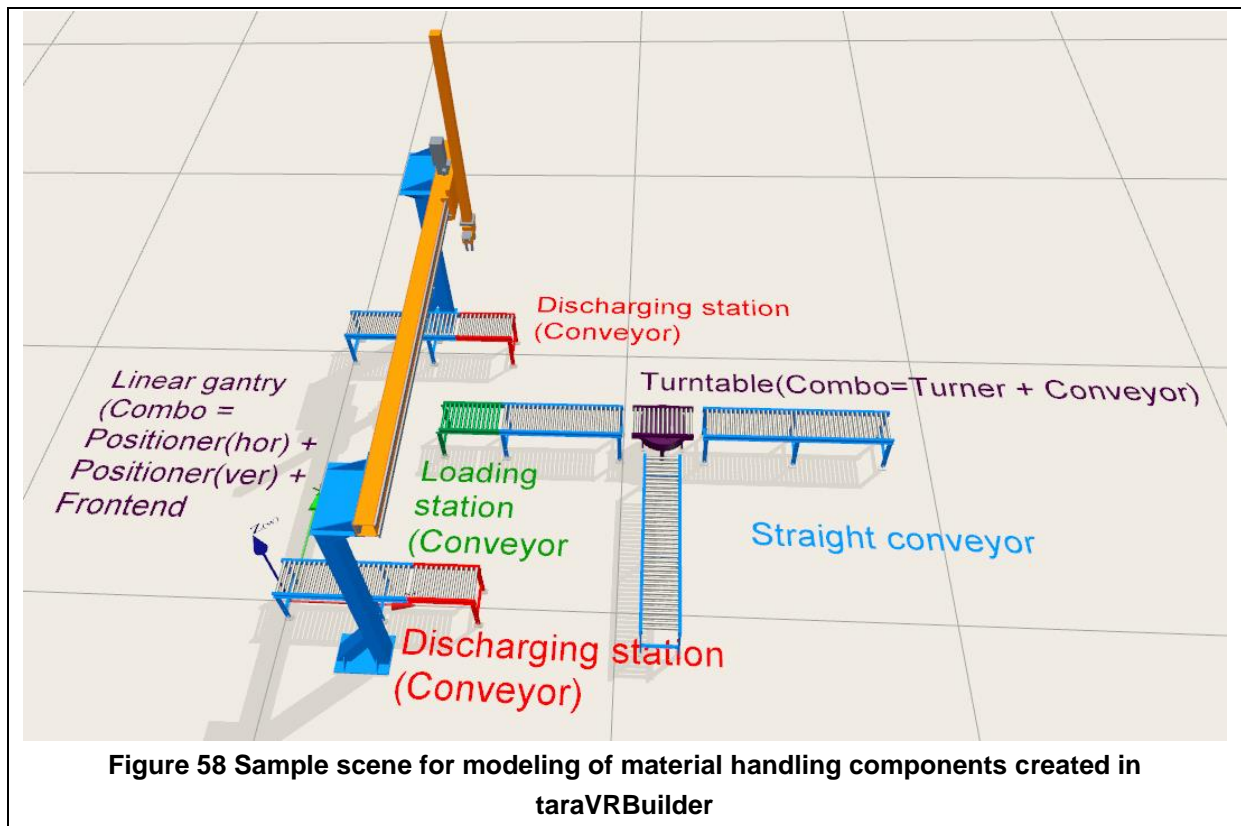
The application of the created models and the implemented software libraries is presented in this chapter. There are two main application fields: material handling simulation and virtual commissioning. The first example is a sample material handling system, which validates the functionality of implemented software libraries for the data exchange with AML in the layout planning tool taraVRBuilder. For the cooperation of tarakos tools with virtual commissioning tools, three AVANTI project demonstrators [AVA16] are introduced. Especially, the second demonstrator is an example of the physics simulation, which is using taraVRControl as the process visualization tool.

8.1. Sample material handling system

An example scene for a material handling system is introduced in this chapter. The application scenario is used to demonstrate the function of data exchange software components in the workflow. The application scene has plenty of elements to present the variety of modeled components. The modeling of a material handling system, exporting and importing functionalities together with the geometry information of the system and the assignment to standardized AML components are explained in the following section. All created and used files of the sample system can be found as digital copies in the attached digital storage medium.

8.1.1. Exporting into AutomationML

The application scene in Figure 58 is created in taraVRBuilder software tool. The material flow begins with two sources on the connection points of two conveyors in the scene. One of the sources produces blue boxes and the other red ones. There is a discharging station for each source, which is connected with a linear gantry. Linear gantry gets the good from both discharging stations and puts it into the loading station. Conveyors transport the good further to a turntable. Turntable lets the red boxes go straight and turns the blue ones to the right. There are ten component instances and five different resource types in the scene.



The first step of the export is transmitting the data structure as it is in the native database. AML-Exporter software component gathers this data structure, and converts it to AML internal elements, assigns the roles and system unit classes. Position parameters are transferred to *Frame* attribute in AML standard and units are converted from millimeters to meters.

Each central element in the instance hierarchy contains one or more internal element “*ConnectionPoint#*” to define its connection points. These connection points include also a *Frame* attribute to determine their position in comparison to the central element. They are assigned to the role class “*MaterialHandlingConnectionPoint*” and inherit the “*MaterialHandlingConnectionPointConnector*” interface, which is linked to the interfaces from other connection point elements to define the material flow (i.e. one connection point for the entrance of the good and the other one for the exit).

A central element in the internal hierarchy can contain one or more functional child elements. For example, the conveyor element named “*StraightConveyor1*” contains the “*ConveyorElement*”, which is defining the functional block for the conveying. This element has the standard role “*StraightConveyor*” and attributes mapped to the standard attributes of the role (*Length*, *Width*, *Velocity*, and *Tilt*). The “*CarrierElement*” defines the conveying technology of the conveyor. For this example, it is a roll carrier, as it is assigned to the standard role “*RollCarrier*”.

The central components, which contain more than one functional child elements, are marked with the standard role class “*ComboElement*”. An example of this concept is the turntable, which consists of a rotational part with the supported role “*LateralTurner*” and a transporting

part with the supported role “*StraightConveyor*”. Another example would be a linear portal with a lifting element with the supported role “*VerticalPositioner*” and a rail element with the supported role “*HorizontalPositioner*”. Thereby it is possible to define elements with complex functionalities by splitting them up to small functional blocks, using superposition principle.

In Figure 59, a screenshot of the exported document from AML-Editor can be seen.

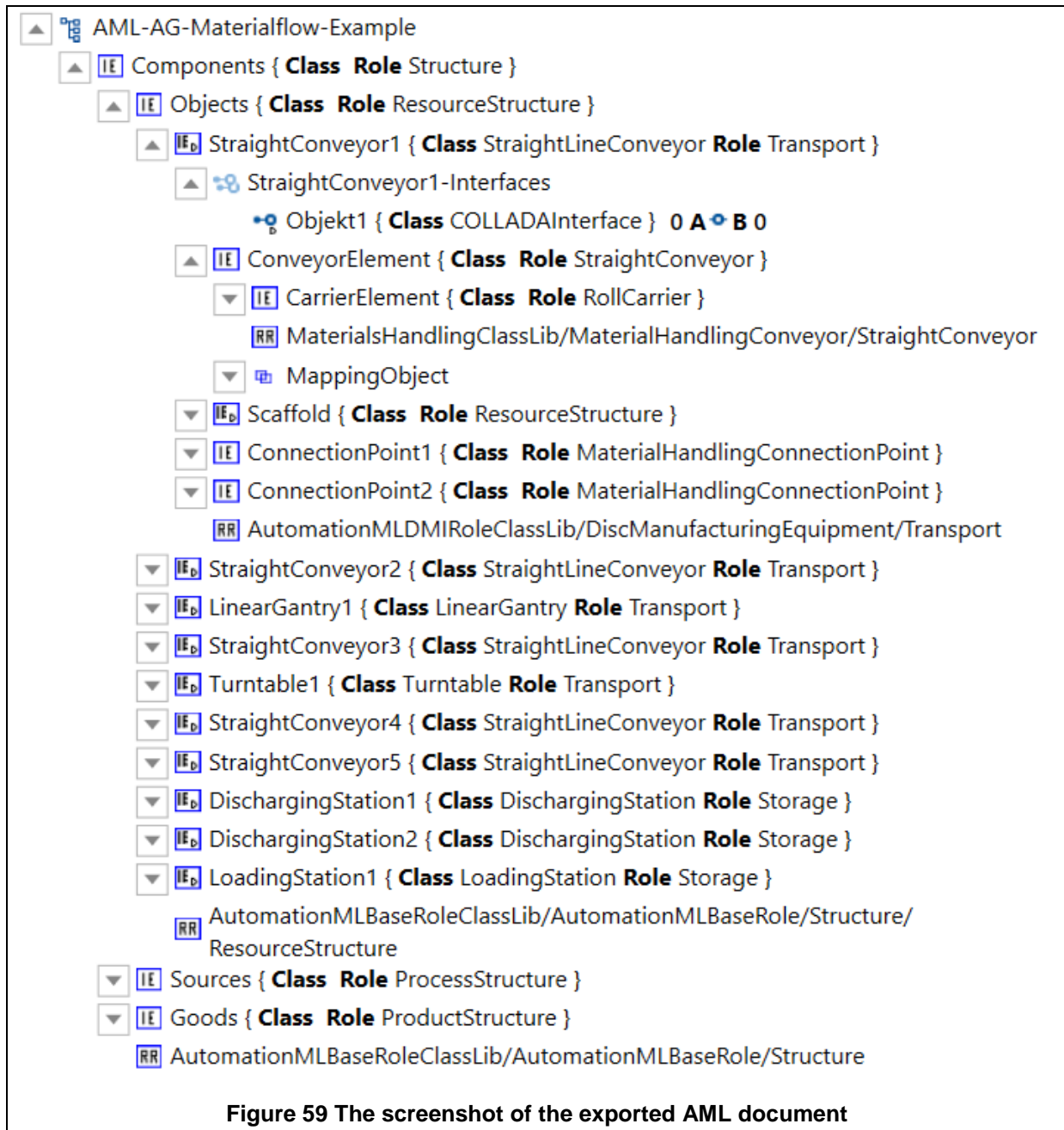


Figure 59 The screenshot of the exported AML document

Process-Product-Resource concept from AML-Whitepaper, which was explained in chapter 5.2.1.2, is applied to the source, good and connection point relationship. The source defines one or more jobs, which are executed to provide a product – the good- into the material handling system. The job has properties like good type, cycle time, and offset. The good enters the resource through the connection point.

Figure 60 explains how this happens in the sample system: The source defines a job to provide a good into the system, which is, in this case, the red box. The red box enters the resource – conveyor- through the connection point, which is highlighted with green color. The arrows show the direction of the good movement.

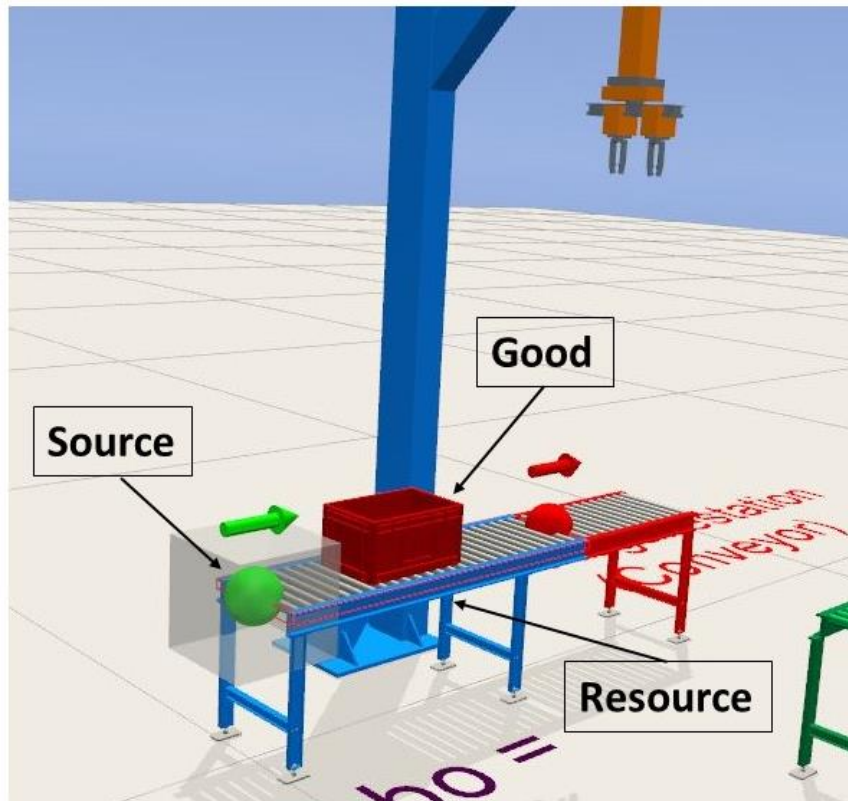
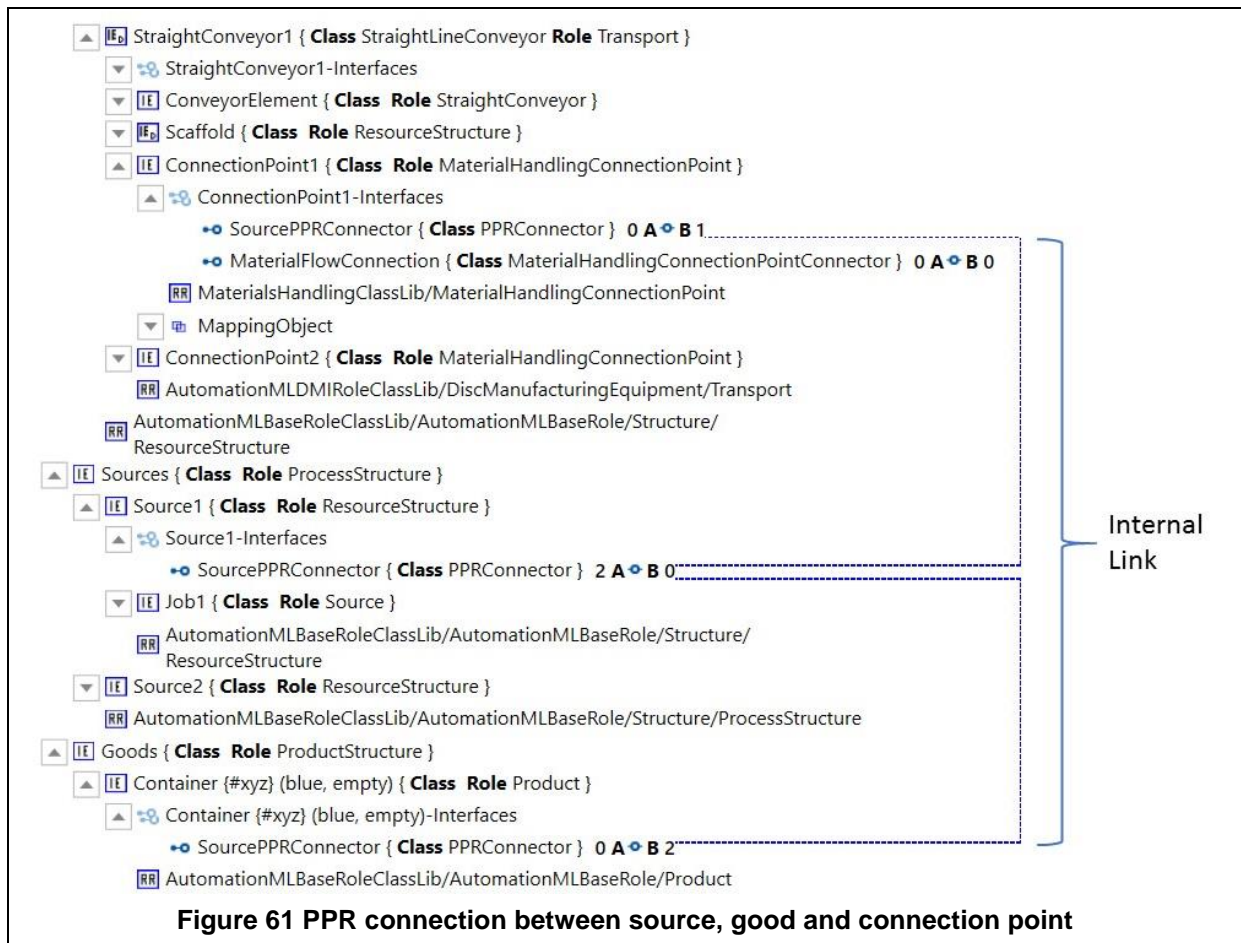


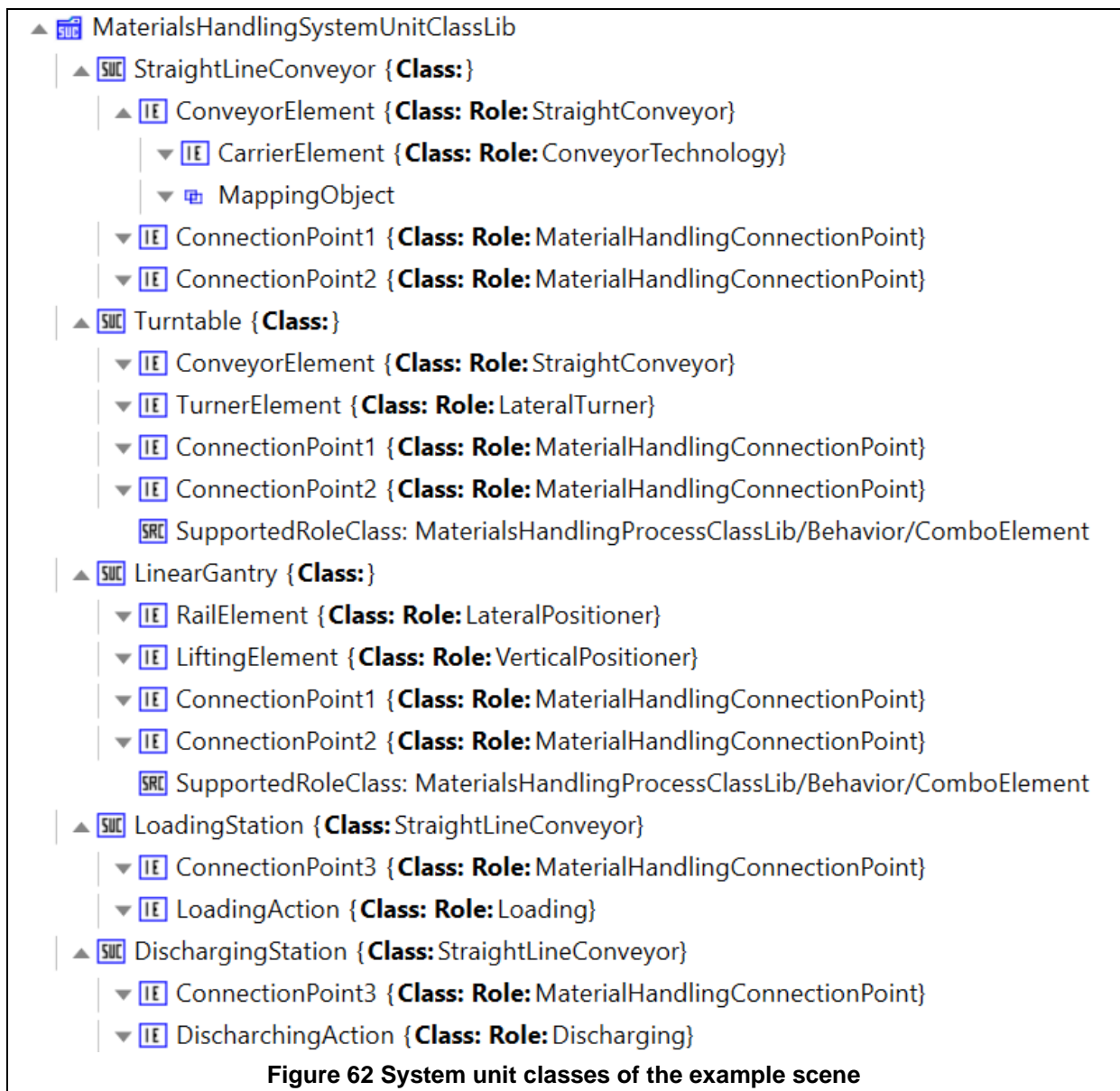
Figure 60 Example implementation for Product-Process-Resource (PPR) concept.

These elements are combined with each other with the help of "PPRConnector" interface class, which is a child of the "AutomationMLBaseInterface". Figure 61 shows how the concept is realized in AML-Export for the sample scene.



8.1.2. Assignment to standard components

The standardization process contains reorganizing the structure for the standard elements, assigning the internal elements to corresponding system unit classes and role classes and creating the attribute/interface mappings. System unit classes of the example scene and assigned roles can be seen in Figure 62. All internal elements in the sample system are created based on these system unit class objects. The role class assignments are also defined in the system unit class objects.



The *"StraightLineConveyor"* system unit class has two connection points. The *"ConveyorElement"* defines the actual transporting component. *"CarrierElement"* as a child of the *"ConveyorElement"* determines which conveyor technology is on the use. Conveyor technology can be a belt carrier, chain carrier, roll carrier, or any other carrier type. It is not yet defined in the system unit class.

The *"Turntable"* is a combination of two main child components: *"ConveyorElement"* and *"TurnerElement"*. The supported role class *"ComboElement"* flags the system unit class as a combination of more than one element. *"ConveyorElement"* is again responsible for transporting the component straight and *"TurnerElement"* changes the direction of the good. This system unit class has again two connection points defined.

The *"LinearGantry"* is also a combination of two elements. The *"LiftingElement"* represents the part, which picks up the good and lifts. The *"RailElement"* is the transporting part in the

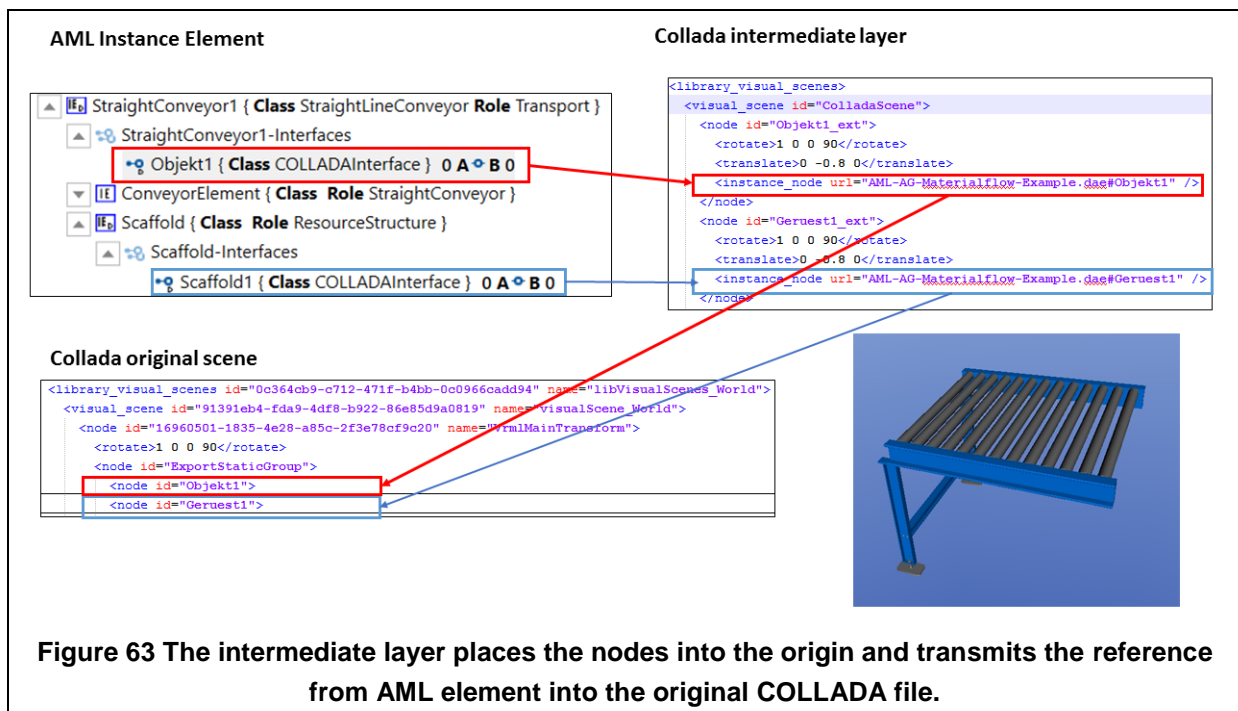
component. The "LinearGantry" is marked as "ComboElement" and has two connection points.

The "LoadingStation" contains all the properties, which the "StraightLineConveyor" has. In addition, it has one extra connection point. Therefore, it is derived from the "StraightLineConveyor" system unit class using the inheritance relation [AML13]. The "LoadingAction" element defines the process, which occurs in the "LoadingStation".

The "DischargingStation" is very similar to the "LoadingStation", except it contains a "DischargingAction" as a process-defining element.

8.1.3. Geometry conversion

Converting of the geometry from native format X3D/VRML to interchange geometry format COLLADA was also a part of the task. The scene graph is exported to COLLADA as a whole. There are references to the corresponding COLLADA nodes in AML instances. An important detail to mention: The transformation of the elements was defined in COLLADA scene and AML scene twice, which leads to confusions in the positions. A solution would be defining only the geometry nodes in COLLADA and let only AML to define the transformations. However, it is also useful to see the whole scene as COLLADA, as not all tools can read AML directly but COLLADA. The implemented solution is to create a layer between in AML and COLLADA in COLLADA file form, which sets the components into the origin. AML instances reference to the nodes in the intermediate layer, which transmits the reference to the original COLLADA node using COLLADA-internal referencing methods (see Figure 63).



The result of the geometry export is tested in different software tools to check the validity of the conversion algorithm. Some examples of these software tools are Autodesk 3dsMax with

OpenCollada, Google Sketchup, Microsoft 3D Builder, and Blender. Although many test tools do not support the COLLADA version 1.5, there were no significant complaints by showing the geometry. The result in Autodesk 3dsMax can be seen in Figure 64.

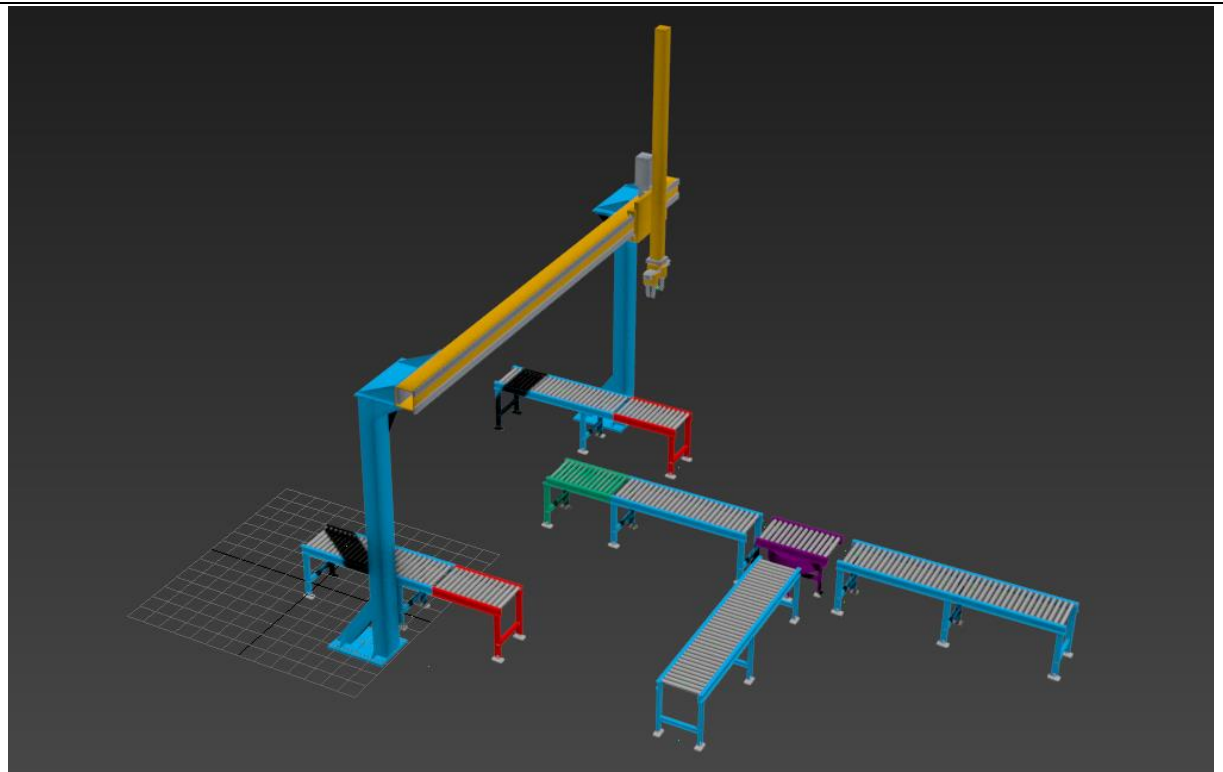


Figure 64 Import result of the example scene in Autodesk 3dsMax with OpenCollada plugin

8.1.4. Importing from AutomationML

The second step on the application was testing the proprietary importer with the export from taraVRBuilder. Therefore, the sample material handling scene is imported to taraVRControl software tool with the AML-Importer functionality. The sequence of the import process has already been explained in the section 7.2.2. The result of the import can be seen in Figure 65.

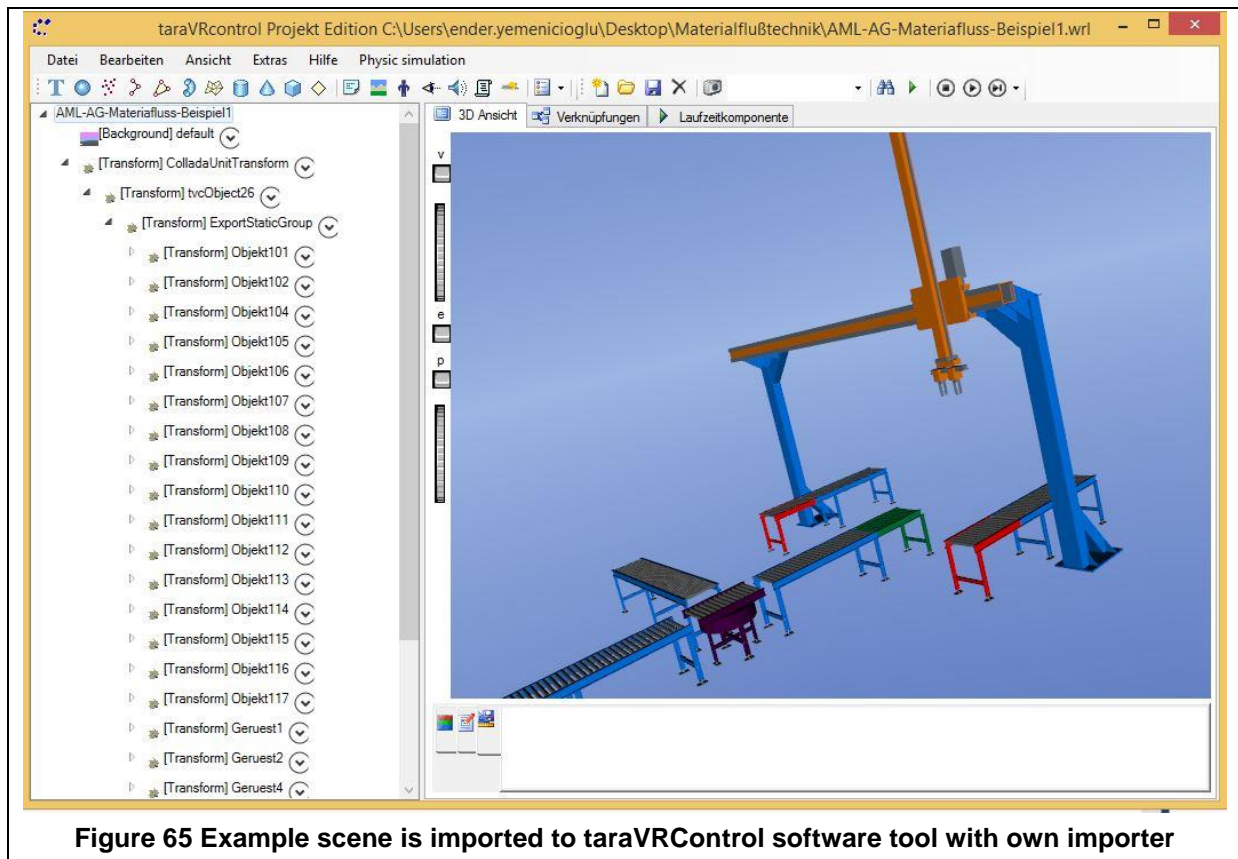


Figure 65 Example scene is imported to taraVRControl software tool with own importer

8.2. Application in virtual commissioning

The layout planning tool taraVRBuilder and visualization tool taraVRControl are equipped with the AML import and export functionalities to communicate with other software tools. The first application field of these software components has been the virtual commissioning. During the ITEA 2⁶ project AVANTI [AVA16], the implemented solutions have been applied rudimentary in a research project with industrial and academic partners. Thereby it can be seen how the data exchange solutions and are utilized in a virtual commissioning toolchain, enhanced with physics simulation and co-simulation framework.

8.2.1. Material handling systems in virtual commissioning toolchain

tarakos software tools joins the virtual commissioning work chain at the beginning with taraVRBuilder and at the end, as a visualization tool for co-simulation results with taraVRControl. In Figure 66, the targeted positioning of tarakos tools in virtual commissioning toolchain is displayed.

⁶ <https://itea3.org/project/avanti.html>

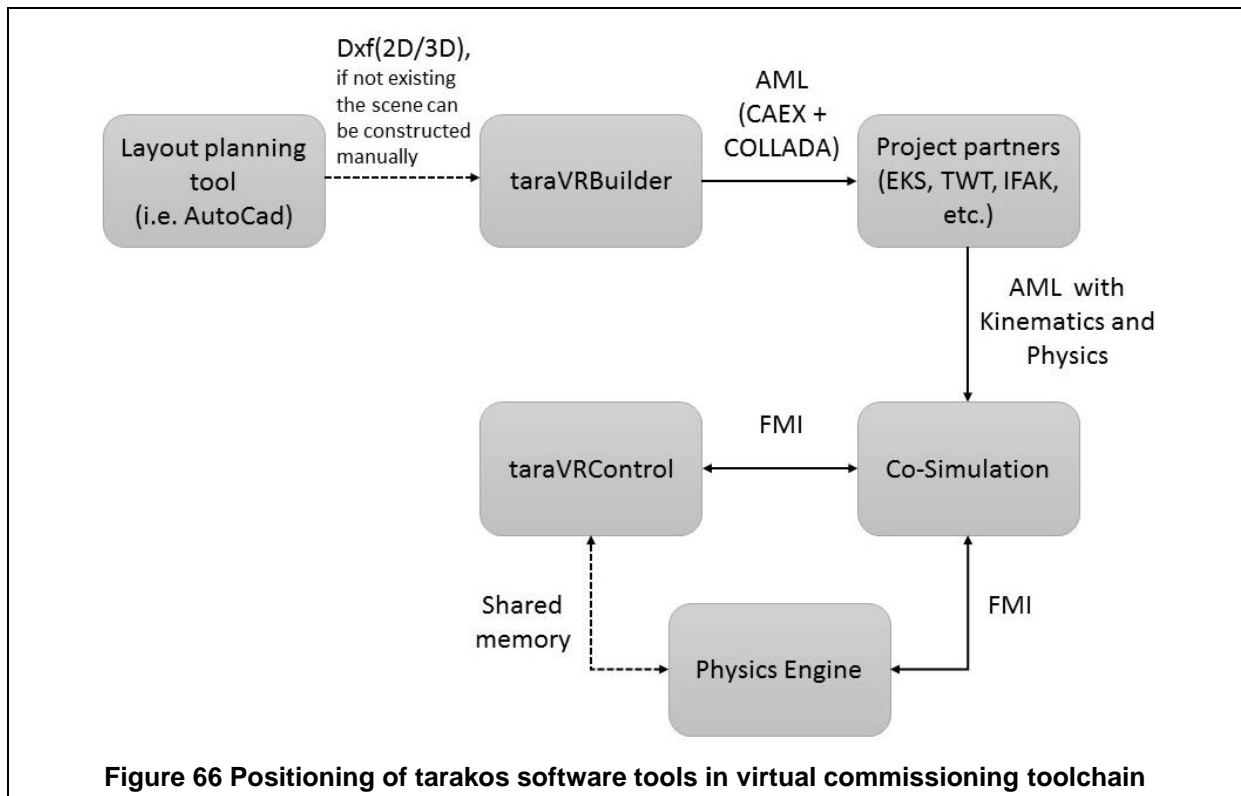


Figure 66 Positioning of tarakos software tools in virtual commissioning toolchain

The first step in the toolchain of tarakos is the importing of 2D/3D layout and component files to create a 3D scene in taraVRBuilder. This step is not essential because it is also possible to build a scene with standard library components of the tool. The position of the manufacturing objects and the logical structure of the material transfer like which element is connected to which one can be described in taraVRBuilder. This information together with the 3D geometry can be exported with the AML interface. After exporting, the content of the data can be enriched with other tools in the work chain. The kinematics and physics of the object models can be included in the AML file. As the next step, the physics scene should be created with the entire information and connected with co-simulation. taraVRControl can connect directly with physics engine as a visualization component or can connect with co-simulation.

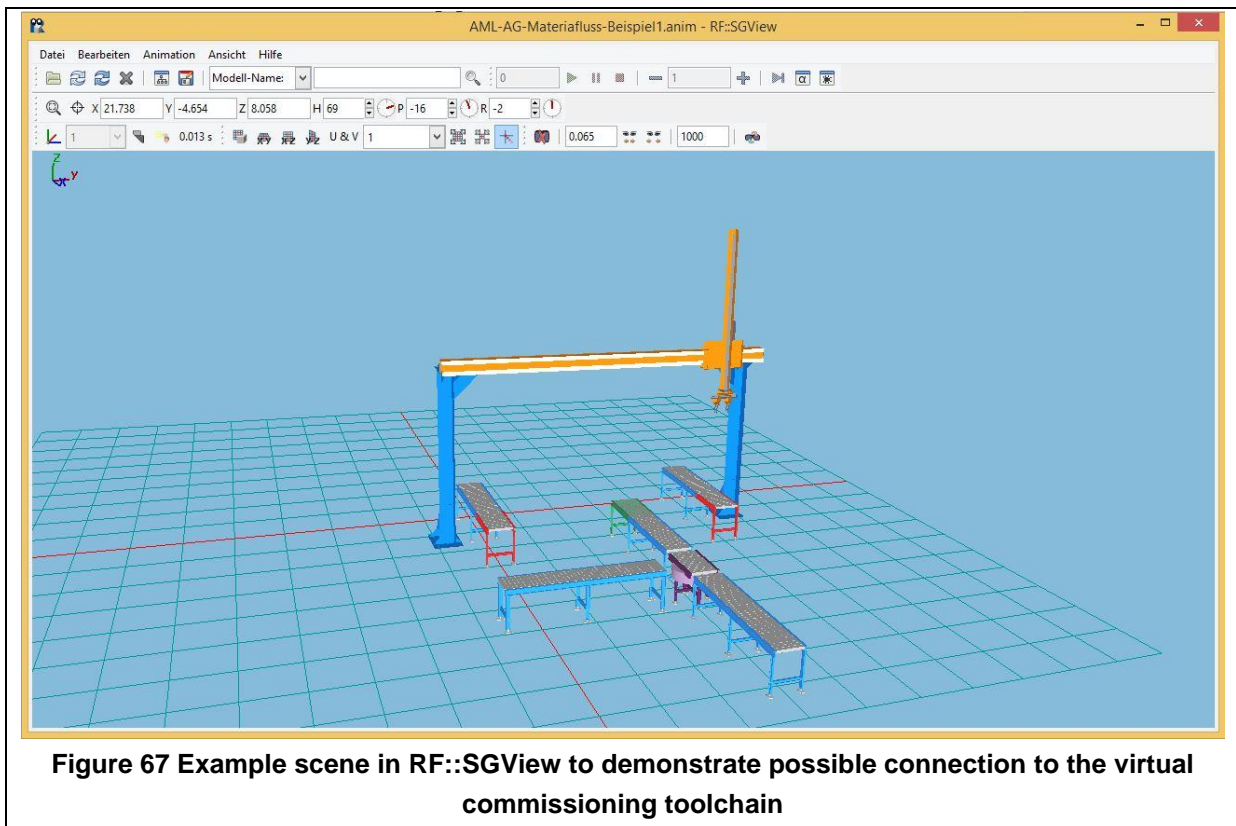


Figure 67 Example scene in RF::SGView to demonstrate possible connection to the virtual commissioning toolchain

In Figure 67, the sample material handling system from section 8.1 is imported into RF:SGView software tool. The RF:Suite software tools from the company EKS Intec are used for executing virtual commissioning for production lines in real time [EKS16]. EKS Intec is one of the project partners in AVANTI project. Thereby, a data exchange from taraVRBuilder into virtual commissioning toolchain is exemplified.

8.2.2. Demonstrators

A demonstrator presents a case that can be run in real-time to give the overview of the simulated process. There are three demonstrators in AVANTI project, in which tarakos software tools participate.

The first demonstrator is a pneumatically driven assembly, named pneumatic station. The original hardware was also established to validate the simulation results on the real platform. This demonstrator was used to test the AML import functionality in taraVRControl. After importing the model from AML, the pneumatic cylinders are combined with the behavior model directly with a plugin implementation in taraVRControl. The behavior model is written as C-Code in this use case, but it is also possible to exchange the C-Code with an FMU when it is present (see Figure 68).

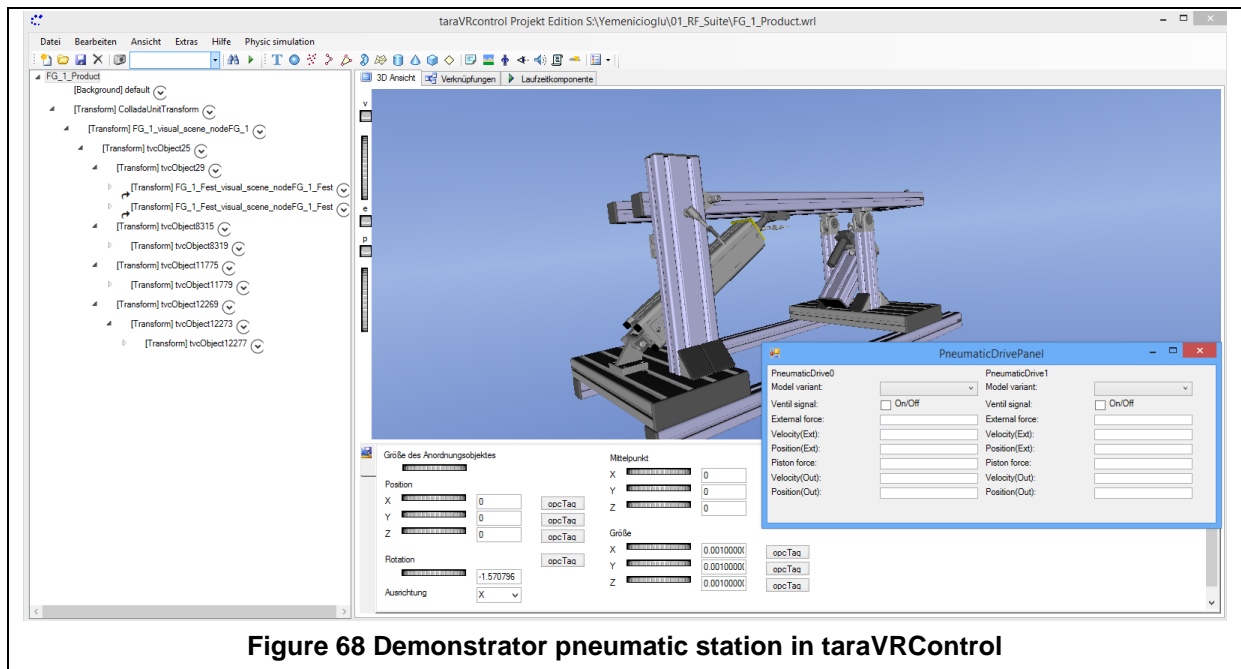


Figure 68 Demonstrator pneumatic station in taraVRControl

The second demonstrator is a material handling system, which is also present as hardware. This use case demonstrates the project results for physics simulation, co-simulation connection and testing system with automated test execution for PLC. The goals for the use case are:

- Realistic simulation of the virtual material handling system
- Systematic testing of the system
- Comparison of the results with the real system

A real material handling system from the project partner FESTO has been established to compare the simulation results of the virtual system with the real behavior. The geometry of the system has been imported into taraVRControl software, which provides the visualization for the physics simulation as seen in Figure 69. The behavior of the individual logical elements is defined in FMUs provided by the project partner IFAK (Institut f. Automation und Kommunikation). The test system for automated testing uses the sensor signals from simulation environment and signals from the logical components to execute the tests. Software components participating the simulation are exchanging signals with each other through the co-simulation framework.

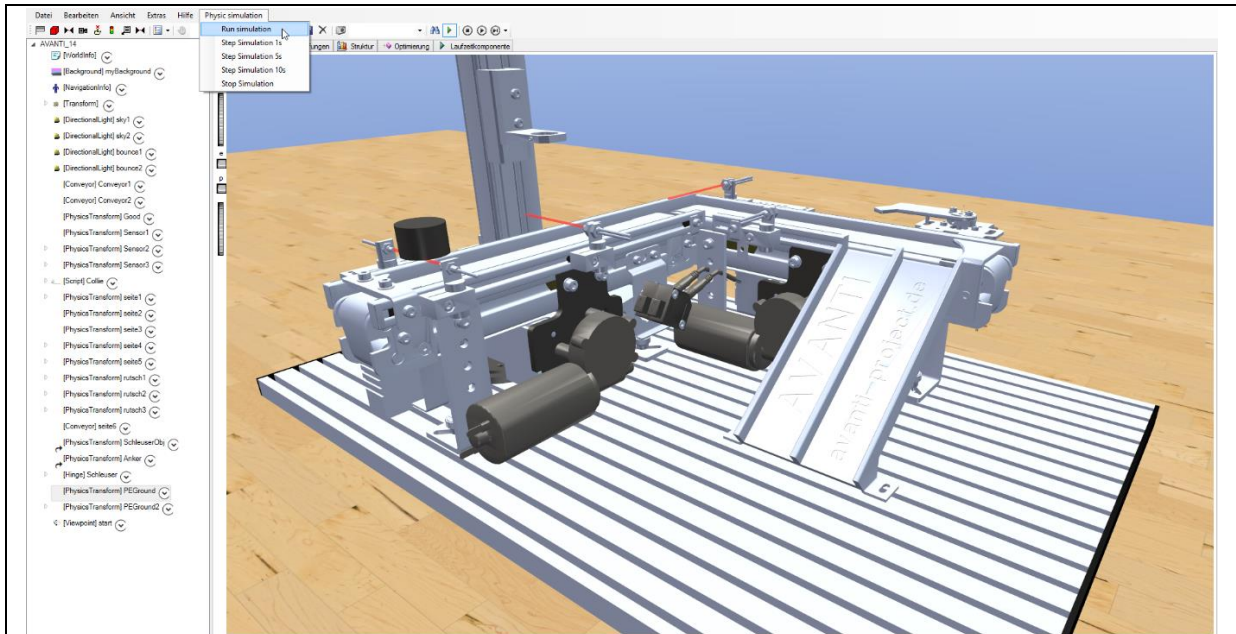


Figure 69 Material handling system use-case in taraVRControl

The third demonstrator is the marriage station, which is a complex assembly station in the automotive industry joining chassis (powertrain) and car-body with manifold pneumatic and electric components, four robots and complex geometries, processes sequences and control programs. Figure 70 shows the demonstrator scene in taraVRControl software, which was imported from AML with the help of COLLADA to X3D/VRML converter library.

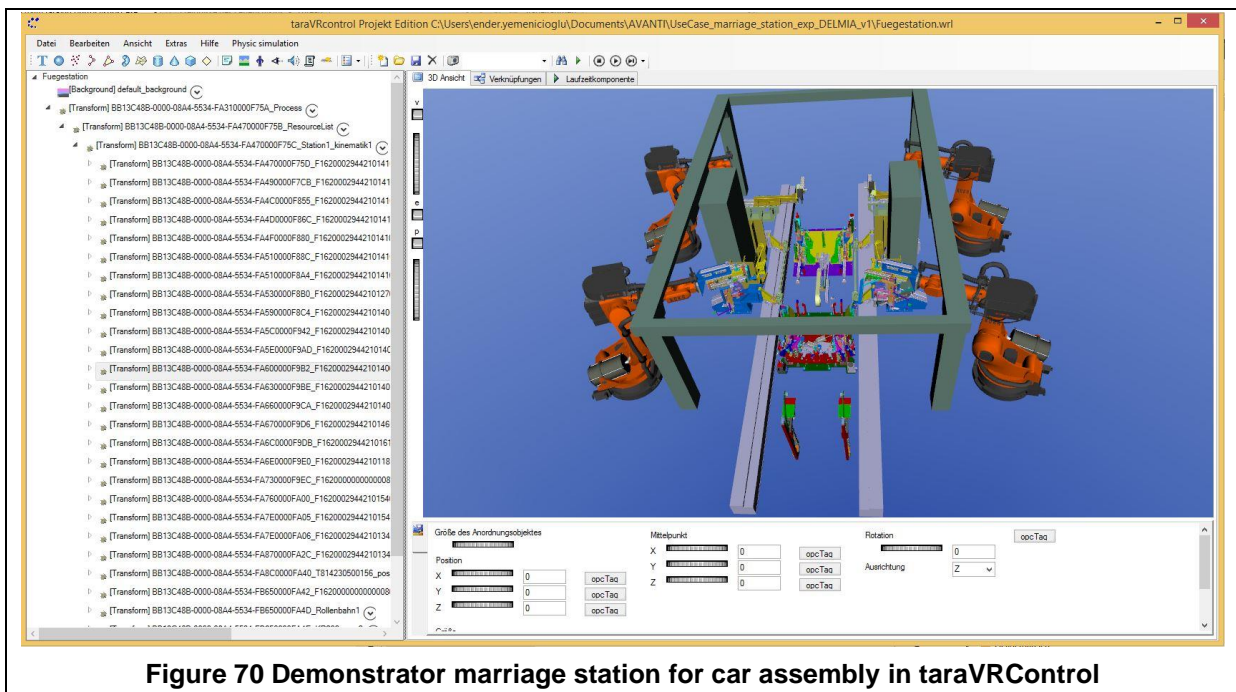


Figure 70 Demonstrator marriage station for car assembly in taraVRControl

8.3. Summary

The applications of the developed software components provide the verification for the modeling and implementation concept. The primary goal of this work is to improve the interoperability of a layout planning tool for virtual commissioning applications. The first sample explains the modeling in details, provides concrete results of exporting and importing actions.

The second part of the application describes how the software components are utilized for the demonstrators of the AVANTI project. The second demonstrator, which is for material handling systems, is the essential demonstrator regarding this work, as it is an example of the physics simulation in the co-simulation environment. The results of the simulation are evaluated from another tool originated from project partners, which participates into the co-simulation.

9. Technical and Economic Evaluation

“The product of mental labor — science — always stands far below its value, because the labor-time necessary to reproduce it has no relation at all to the labor-time required for its original production.” – Karl Marx in ‘Economic Manuscripts’

An implementation of a data exchange interface with the purpose of integrating a plant layout planning tool to the engineering chain of digital factory simulation is presented in this work. The solution is based on the standardized data exchange format AML. Description of the object models and the hierarchical object structure of the plant layout, as well as exchanging the geometrical and physical information were the subjects of the main effort. The first application for the developed software solutions is the AVANTI project, whose primary goal is to provide a test methodology for virtual commissioning based on behavior simulation of production systems [AVA16].

9.1. Technical evaluation

The description/creation of the mechatronical object models for material handling components in a simulation environment was the first aim of this work as defined in chapter 1.2. The information types, which are already defined in the standard, i.e. in AML Whitepaper, could be easily identified and transmitted through the different engineering tools. Plant topology, the position of the components, connections between the objects and predecessor-successor relationship in the material flow can be mentioned as such information. The classification and transformation of objects in AML are carried out with the help of role class libraries. The identification of an imported object or the compatibility of an exported object to external tools is only possible if the role class of the object is defined and the role is known for the related tools [Hun13]. Therefore, a common role class library and an interface class library for the material handling components has been developed in cooperation with AML partners [Aut16], and proprietary models are created based on these libraries as described in chapter 6.

Development of new software related solutions to enhance the data exchange in the digital factory toolchain was another defined goal of this work. The first step to reach that purpose is to export the internal data structure as AML internal hierarchy with the help of AML-Engine and proprietary software libraries in cooperation. The second step is converting the geometry to COLLADA format and referencing them in the AML document. There is also the use case that AML documents with COLLADA references are gathered from project partners for the process visualization. So, import functions for these document types are also implemented.

In this regard, converting the geometry primitives from the native format to the interchange file format and the other way, meaning conversion between X3D/VRML and COLLADA, was a time-consuming but a successful task. However, some bottlenecks are observed during

this practice. The boundaries of the conversion between COLLADA and X3D/VRML is reached quickly if boundary representation (BREP) should be handled, which is supported in COLLADA but not yet in VMRL/X3D. Non-uniform rational basis spline (NURBS) are existent in both formats. However, it is rare to see in industrial applications. A comparison of syntaxes in both formats for NURBS should be examined in the case of need. The kinematic and physical properties of the elements in COLLADA are also a challenge for the transfer. “Rigid body components” in X3D specification offers a solution for the kinematic information, but one-to-one conversion is not possible. Proprietary X3D prototype elements should be created to overcome this problem as explained in chapter 5.3.4. A conversion from prototype elements to COLLADA physics libraries has not been implemented, as there was no use case for this functionality and prototype elements are company-internal solutions which are not standardized (see chapter 5.3.4).

Tool support was also a challenge when dealing with COLLADA files with physics libraries i.e. for test purposes. Although the specification was published in 2008, there is no tool on the market known to the author, which supports the complete COLLADA 1.5 functionality. Autodesk Maya was the only tool to create reasonable COLLADA examples with physics, but it required extra effort to install external plugins with the right version. COLLADA model repository server, which should have to provide test samples, is out of service for a long time. That makes it difficult to find test samples with quality and variety. A general problem for COLLADA format is that it stays put since 2008 without any apparent intention to be developed further.

The data exchange occurs until now only unidirectional; that means a new scene is created with the import function and with export, the previous status of the data is lost. All objects have a unique signature as “Globally Unique Identifier” (GUID) and header information for the project identification is also provided. A “*ChangeMode*” attribute for the objects, which shows the status of the object if it is “created”, “changed” or logically “deleted”, are implemented to track the changes in object-level. For the proper use of this attribute, no object is allowed to be erased physically from the file and the changed objects should exist with its former and actual form [HPr13]. However, it is yet to be implemented. The software solution should now be enhanced to allow bi-directional data exchange with the support of versioning in the whole file and each instance objects. However, this functionality cannot be applied within the plugin concept; it also requires enhancements in the source code of the target software tool.

9.2. Economic evaluation

The layout planning and visualization tools for material handling and logistics, which are the main subject of this work, were so-called “islands” in the engineering toolchain with very few possibilities to exchange information with other engineering tools at the beginning. The current implementation of the data exchange libraries allows new communication opportunities with other software tools from engineering fields like CAD design, virtual commissioning, and material flow simulation.

Reusable physical component models and the existence of data exchange interfaces reduce engineering effort in the simulation process. The need of scripting the animation of the components is eliminated with the help of physics engine assignment if physics models are provided. This reduces the amount of the development effort while introducing new component models into the software tool.

Supporting more interchange formats widens the application possibilities for the software tool. The Khronos Group has a list of products that provide COLLADA plug-ins or that import or export COLLADA [The13]. This list contains more than 100 software tools. Some of them are leading CAD design tools like Autocad, Microstation, Google Sketchup, etc. Compatibility with such a spectrum of software tools is an attractive functionality for the customers, which are already using them for business. So COLLADA compatibility is a good marketing argument.

tarakos software tools are used for visualization and animation of material handling and manufacturing systems, but simulation and virtual commissioning are not a core functionality. Although the software itself does not provide such features, transporting the data to another software tool, which is specialized in this field, is now possible with the new AML-export function. The continuing efforts to standardize the AML export from tarakos software show some new possibilities in this area. A new project, named ADEX “AutomationDataEXchange” [FEN16], has already started to implement a continuous data exchange pipeline from a CAD design tool (i.e. Autodesk Inventor) into taraVRBuilder and further into a simulation tool (i.e. Plant Simulation).

tarakos GmbH is planning to market the exploitable results as a commercial product. The expected revenues result from the sale of licenses, service contracts and support services. Following expectations for the market potential can be mentioned as results of the work:

- Licensing and support of the AML integration for the end user
- Acquisition of new customers that defer the investment due to lack of data consistency and interoperability (additional tarakos software kit and AML integration licenses)
- Sustainable service contracts

Target industry fields are IT service providers, manufacturers of logistics and material handling systems, manufacturers of mechatronics systems, plant designers, and other mechanical engineering related simulation-users.

10. Summary and Results

“Words do not express thoughts very well. They always become a little different immediately after they are expressed, a little distorted, a little foolish. And yet it also pleases me and seems right that what is of value and wisdom to one man seems nonsense to another.” – Hermann Hesse in ‘Siddhartha’

The increasing complexity of the engineering design process and the necessity of product variety are two main technical challenges for the facility planners. An effective and efficient solution to these problems can ensure an essential advantage in the market. A continuous improvement of organizational and technical measures is, therefore, indispensable. The application of software solutions like digital factory concepts, simulation methods or virtual commissioning allows testing and validation of improvements in an early stage of the plant construction, which means enormous time and cost gain.

Despite these positive aspects, the growing complexity of software components, the variety of engineering tools in different engineering phases cause an increasing effort for the data management. A continuous data exchange through all phases of the engineering design -from product design step until the real commissioning- would be the ideal solution for the engineers. Although the state of the technology is far away from this goal, there is a continuous enhancement on this field.

As a part of these efforts, a data exchange solution via AML format for a layout planning tool is presented in this work. There is a variety of subtasks to achieve the end solution. First of all, an analysis of the mechatronical engineering design process is given. It is important to find out which kind of information is in use in the engineering design phases. The emphasis is of course on the digital factory methods for the layout planning. An introduction to some mathematical and software related aspects is also given.

After the introduction of primary concepts, a requirement analysis for methodological and technological aspects is done. The third chapter of the analyzation is about the data exchange formats. A data exchange format specifies the layout for the organization of the information. Standard data formats, which could fulfill the defined requirements, are analyzed, and selected. Data formats, which are chosen in this doctoral study as AML, COLLADA, X3D/VRML and FMI, are introduced in this chapter, too.

The preparation of the models for the data exchange is explained in the section for the modeling of material handling system elements. The focus lies on the topological, geometrical, kinematical, and physical information for the components of a material handling system. The process types of a material handling system are also exemplified.

The chapter about the software implementation gives an overview of the applied software solutions. It provides a brief introduction to the software components, which are defined as the

target tools for the data exchange enhancements. The architecture of the software libraries and some exemplary parts of the implementation are also given in this section.

The application chapter presents how the software solutions are applied to solve data exchange problem in a material handling use-case and a virtual commissioning use-case. An exemplary scene for a material handling system is introduced in this chapter. Also, there are three demonstrators for the virtual commissioning application, which are realized together with AVANTI project partners.

At the end, not only the feasibility of the assumptions has been proved, but also a commercial solution emerges from this work. tarakos software tools are now using the software libraries, which are developed during the praxis of this doctoral study. The first application was for the virtual commissioning use-case in AVANTI project, but it opens new doors into the new fields from physics simulation to material flow simulation and further. In conclusion, it can be said that a data exchange solution for physics-based simulation of material handling systems in the digital factory has been created. The experience, which is gained in this work, can be transmitted to further research subjects and industrial projects like ADEX [FEN16] and ENTOC [ITE16].

References

- [Agi16] Agile Alliance: *Agile manifesto*. <http://www.agilemanifesto.org/principles.html>, retrieved in 10.June.2016.
- [AML13] AutomationML consortium (2013): *Whitepaper AutomationML. Part 1 - Architecture and general requirements*. <https://www.automationml.org/o.red.c/dateien.html>, retrieved in 03.October.2016.
- [Arn07] Arnaud R. and Parisi T. (2007): *Developing web applications with collada and x3d. A whitepaper*. https://www.khronos.org/collada/presentations/Developing_Web_Applications_with_COLLADA_and_X3D.pdf, retrieved in 09. October 2016
- [Arn09] Arnold D. and Furmans K. (2009): *Materialfluss in Logistiksystemen (Vol. 4)*. Berlin: Springer Verlag.
- [Arna06] Arnaud R. and Barnes M.C. (2006): *COLLADA: sailing the gulf of 3D digital content creation*. CRC Press.
- [Aut13] AutomationML consortium (2013): *Whitepaper AutomationML. Part 3 - Geometry and Kinematics*. <https://www.automationml.org/o.red.c/dateien.html>, retrieved in 03.October.2016.
- [Aut16] AutomationML Material Handling Working Group (2016): *Application Recommendation Document Modelling of Material Handling in AutomationML*. (Application Recommendation).
- [AVA16] AVANTI Consortium (2016): *AVANTI: Test methodology for virtual commissioning based on behaviour simulation of production systems*. <http://avanti-project.de>, retrieved in 03.Juli.2016.
- [Bab06] Baber R. (2006): *Rigid body simulation. Doctoral Dissertation*. University of Warwick.
- [Bak16] Baker M.J.: *EuclideanSpace - Mathematics and Computing*. <http://www.euclideanspace.com/maths/algebra/matrix/index.htm>, retrieved in 20.March.2016.

- [Bar08] Barnes M. and Levy Fi E. (2008): *COLLADA – Digital Asset Schema Release 1.5.0*. https://www.khronos.org/files/collada_spec_1_5.pdf, retrieved in 2016.
- [Bar091] Barth M.; Strube M.; Fay A. et al. (2009): *Object-oriented engineering data exchange as a base for automatic generation of simulation models*. In: Industrial Electronics, 2009. IECON'09. 35th Annual Conference of IEEE. IEEE. p. 2465-2470.
- [Bar15] Barbieri G.; Goldoni G.; Borsari R. et al. (2015): *Modelling and Simulation for the Integrated Design of Mechatronic Systems*. In: IFAC-PapersOnLine, 48(10). p. 75-80.
- [Bar16] Barlas P. and Heavey C. (2016): *Automation of input data to discrete event simulation for manufacturing: A review*. In: International Journal of Modeling, Simulation, and Scientific Computing, 7(01), 1630001.
- [Bart09] Barth H. (2009): *Anforderung an Engineeringwerkzeugen für das „Mechatronic Engineering“ dezentral automatisierter Fertigungsanlagen*. (Diploma thesis) Otto-von-Guericke University Magdeburg - Faculty of Mechanical Engineering, Magdeburg.
- [Bas11] Bastian J.; Clauß C.; Wolf S. et al. (2011): *Master for Co-Simulation Using FMI*. In: 8th International Modelica Conference, Dresden.
- [Bau09] Baumbach P. (2009): *Analyse der Anwendbarkeit der Automation Markup Language im Rahmen des Engineeringprozesses von Produktionssystemen an einem Beispiel*. (Student research project) Otto-von-Guericke University Magdeburg - Faculty of Mechanical Engineering, Magdeburg.
- [Bei00] Beier K.-P. (2000): *Web-based virtual reality in design and manufacturing applications*. In: Proceedings of COMPIT. p. 45-55.
- [Ber06] Bergholz M. (2006): *Objektorientierte Fabrikplanung*. Faculty of Mechanical Engineering. Aachen University, Aachen. http://publications.rwth-aachen.de/record/59650/files/Bergholz_Markus.pdf, retrieved in 09. October 2016.
- [Ber09] Bergert M.; Kiefer J.; Hoeme S. et al. (2009): *Einsatz der Virtuellen Inbetriebnahme im automobilen Karosserierohbau–Ein Erfahrungsbericht*. In: Tagungsband der, 9. p. 388-397.
- [Ber10] Bergert M.; Höme S. and Hundt L. (2010): *Verhaltensmodellierung für die virtuelle Inbetriebnahme*. In: etz elektrotechnik & automation, 9.

- [Bha00] Bhandarkar M.P.; Downie B.; Hardwick M. et al. (2000): *Migrating from IGES to STEP: one to one translation of IGES drawing to STEP drafting data*. In: *Computers in industry*, 41(3), p. 261-277.
- [Bie14] Bierwirth D.; Kägebein S. and Simsek T. (2014): *Das digitale Baustellen-Management*. <http://www.computer-automation.de/unternehmensebene/produktionssoftware/artikel/109400/>, retrieved in 22. May 2016.
- [Blo11] Blochwitz T.; Otter M.; Arnold M. et al. (2011): *The functional mockup interface for tool independent exchange of simulation models*. In: 8th International Modelica Conference. Dresden. p. 20-22.
- [Blo12] Blochwitz T.; Otter M.; Åkesson J. et al. (2012): *Functional mockup interface 2.0: The standard for tool independent exchange of simulation models*. In: 9th International Modelica Conference.
- [Bös08] Bös M. (2008): *Methoden der Digitalen Fabrikplanung – ein praxisorientierter Ansatz für KMU*. In Rabe M. (Hg.), *Advances in Simulation for Production and Logistics Applications*. Stuttgart: Fraunhofer IRB Verlag. p. 407-415.
- [Bru10] Brutzman D. and Daly L. (2010): *X3D: extensible 3D graphics for Web authors*. Morgan Kaufmann Publishers, California, U.S.A.
- [Cam16] Cambridge University Press (2016): *Cambridge Dictionaries Online*. <http://dictionary.cambridge.org/dictionary/english/plug-in>, retrieved in 03.June.2016.
- [Cor12] Core Manufacturing Simulation Data Product Development Group (2012): *Standard for Core Manufacturing Simulation Data - XML Representation*. Orlando, USA: Simulation Interoperability Standards Organization.
- [Cou13] Coumans E. (2015): *Bullet 2.83 Physics SDK Manual*. https://github.com/bulletphysics/bullet3/blob/master/docs/Bullet_User_Manual.pdf, retrieved in 16.June.2016.
- [Din14] Dini G. and Spath D. (2014): *Material Flow*. In: *CIRP Encyclopedia of Production Engineering*. Springer Berlin Heidelberg. p. 844.
- [Doa12] Doan A.; Halevy A. and Zachary I. (2012): *Data Warehousing and Caching*. In: *Principles of data integration*. Elsevier. p. 277-291.

- [Dra08] Drath R.; Lüder A.; Peschke J. et al. (2008): *AutomationML – the glue for seamless Automation Engineering*. In: Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference. p. 616-623.
- [Dra09] Drath R. (Hg.) (2009): *Datenaustausch in der Anlagenplanung mit AutomationML: Integration von CAEX, PLCopen XML und COLLADA*. Springer-Verlag.
- [Dra13] Drath R. and Barth M. (2013): *Wie der Umgang mit unterschiedlichen Datenmodellen beim Datenaustausch im heterogenen Werkzeugumfeld gelingt*. In: Tagungsband AUTOMATION.
- [Dra131] Drath R. (2013): *Effizientes Programmieren mit AutomationML*. In: SPS Magazin, Jahrgang 2013, Heft 3, p. 38-40.
- [Dre13] Dreher S.; Nürnberger A.; Kägebein S. et al. (2013): *Digitales Baustellenmanagement für Produktionsanlagen*. http://www.zwf-online.de/ta003/na0/ar213228124249-1415/baustellenmanagement/INPRO-Innovationsakademie-Januar-2013_archiv.html, retrieved in 22.May.2016.
- [DrSc10] Drath R. and Schleipen M. (2010): *Grundarchitektur: das Objektmodell*. In: Datenaustausch in der Anlagenplanung mit AutomationML. Springer-Verlag Berlin Heidelberg. p. 55.
- [EKS16] EKS InTec GmbH: *Virtual Commissioning*. <http://www.eks-intec.de/en/services/virtual-commissioning.html>, retrieved in 20.May.2016.
- [Epp08] Epple U. (2008): *Begriffliche Grundlagen der leittechnischen Modellwelt*. In: atp edition-Automatisierungstechnische Praxis, 50(04), p. 83-91.
- [FEN16] F&E Netzwerk Assistenz in der Logistik (2016): *Forschungs- und Entwicklungsprojekt "ADEX"*. <http://logistik.exfa.de/index.php?fue-projekte-adex-1>, retrieved in 01.October.2016.
- [Fje08] Fjeldberg H.-C. (2008): *Polyglot Programming*. Trondheim/Norway: Norwegian University of Science and Technology.
- [Fow14] Fowler M. (2014): *UnitTest*. <http://martinfowler.com/bliki/UnitTest.html>, retrieved in 03.March.2016.

- [Fow95] Fowler J. (1995): *STEP for Data Management, Exchange and Sharing*. Great Britain.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.107.2051&rep=rep1&type=pdf>, retrieved in 09. October 2016.
- [Fri11] Fritzson P. (2011): *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. New Jersey: John Wiley & Sons, Inc.
- [Fri12] Frigg R. and Hartmann S. (2012): *Models in Science*.
<http://plato.stanford.edu/archives/fall2012/entries/models-science/>, retrieved in 06.March.2106.
- [Gal04] Gallaher M.P.; O'Connor A.C.; Dettbarn J.L.J. et al. (2004): *Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry*. Gaithersburg, Maryland.
- [Gla16] Glassfish Community: *Project JAXB*. <https://jaxb.java.net/>, retrieved in 03.Juli.2016.
- [Gra13] Graeser O.; Kumar B.; Niggemann O. et al. (2013): *AutomationML as a Shared Model for Offline-and Realtime-Simulation of Production Plants and for Anomaly Detection*. In: Informatics in Control, Automation and Robotics. Springer Berlin Heidelberg. p. 195-209.
- [Gri10] Grillitsch U. and Mayer G. (2010): *Auf dem Weg zum Standard-Virtuelle Inbetriebnahme von IT-Steuerungssystemen in der Produktionssteuerung*. In: Integrationsaspekte der Simulation. Karlsruhe: KIT Scientific Publishing. p. 591-598.
- [Gut03] Gutenschwager K. and Kemper J. (2003): *Neue Potentiale in der Logistiksimulation*. In: A&D Kompendium, p. 56-58.
- [HPr13] Hundt L. and Prinz J. (2013): *AutomationML – Engineering Workflow*. In: SPS Magazin 11.
- [Hun10] Hundt L.; Lüder A. and Estévez E.E. (2010): *Engineering of manufacturing systems within engineering networks*. In: Emerging Technologies and Factory Automation (ETFA). 2010 IEEE Conference.
- [Hun12] Hundt L. (2012): *Durchgängiger Austausch von Daten zur Verhaltensbeschreibung von Automatisierungssystemen*. Logos Verlag Berlin GmbH.

-
- [Hun13] Hundt L. and Prinz J. (2013): *AutomationML - Datenaustausch*. In: SPS Magazin 4.
- [IEE90] IEEE Standards Association (1990): *Standard glossary of software engineering terminology*.
- [Int16] International Federation of Automatic Control (2016): *Mechatronics, Robotics and Components/Mechatronic Systems/Scope*. <http://tc.ifac-control.org/4/2/scope>, retrieved in 03.October.2016.
- [Ise07] Isermann R. (2007): *Mechatronic systems: fundamentals*. Springer Science & Business Media.
- [ITE16] ITEA 3 (2016): *ENTOC - Engineering Tool Chain for Efficient and Iterative Development of Smart Factories*. <https://itea3.org/project/entoc.html>, retrieved in 1.October.2016.
- [Kas95] Kasper R. and Koch W. (1995): *Object-oriented behavioural modelling of mechatronic systems*. In: Proceedings of the third conference on mechatronics and robotics. Vieweg+ Teubner Verlag. p. 70-84.
- [KIE07] Kiefer J. (2007): *Mechatronikorientierte Planung automatisierter Fertigungszellen im Bereich Karosserierohbau*. (Doctoral thesis). Institute of Production Engineering, Saarland University, Saarbrücken.
- [Klo10] Kloss J. (2010): *X3D: Programmierung interaktiver 3D-Anwendungen für das Internet*. Pearson Deutschland GmbH.
- [Lac12] Lacour F.-F. (2012): *Modellbildung für die physikbasierte Virtuelle Inbetriebnahme materialflussintensiver Produktionsanlagen*. Herbert Utz Verlag.
- [Lar08] Larson W. (2008): *How to Write a Wrapper Library*. <http://lethain.com/how-to-write-a-wrapper-library/>, retrieved in 27.March.2016.
- [Lea16] Leach G.: *Lecture: Normal Vectors*. <http://goanna.cs.rmit.edu.au/~gl/teaching/Interactive3D/2012/normals/normals.xhtml>, retrieved in 26.March.2016.
- [LiC09] Li C. (2009): *XML Parsing, SAX/DOM*. In Encyclopedia of Database Systems. Springer US. p. 3598-3600.

- [Lip09] Lipps S. (2009): *Beschreibung von Geometrie und Kinematik mit COLLADA*. In: Datenaustausch in der Anlagenplanung mit AutomationML. Springer-Verlag. p. 95-134.
- [LNY16] Lüder A.; Schmidt N. and Yemenicioglu E. (2016): *Herstellerunabhängiger Austausch von Verhaltensmodellen mittels AutomationML*. In: VDI (Hg.): AUTOMATION 2016 Secure & reliable in the digital world. Baden-Baden. VDI-Berichte 2284. p. 177-179.
- [LSH13] Lüder A.; Schmidt N. and Helgermann S. (2013): *Lossless exchange of graph based structure information of production systems by AutomationML*. In: Emerging Technologies & Factory Automation (ETFA), IEEE 18th Conference.
- [Lud10] Lüder A.; Hundt L.; Foehr M. et al. (2010): *Manufacturing system engineering with mechatronical units*. In: Emerging Technologies and Factory Automation (ETFA). 2010 IEEE Conference. p. 1-8.
- [Lüd13] Lüder A.; Rosendahl R. and Schmidt N. (2013): *Validation of behavior specifications of production systems within different phases of the engineering process*. In: Emerging Technologies & Factory Automation (ETFA). IEEE 18th Conference.
- [May10] Mayer G. and Pöge C. (2010): *Auf dem Weg zum Standard–Von der Idee zur Umsetzung des VDA Automotive Bausteinkastens*. In: Integrationsaspekte der Simulation: Technik, Organisation, Personal. Tagungsband der 14. p. 29-36.
- [MC12] Microsoft Corporation (2012): *C# Language specification. Version 5.0*. <https://www.microsoft.com/en-us/download/details.aspx?id=7029>, retrieved in 13.March.2016.
- [MC16] Microsoft Corporation: *Creating a Project (Visual C#)*. <https://msdn.microsoft.com/en-us/library/ms173077%28v=vs.90%29.aspx#>, retrieved in 25.March.2016.
- [MCT16] Microsoft Corporation (2016): *Tessellation Overview*. <https://msdn.microsoft.com/en-us/library/ff476340%28v=VS.85%29.aspx>, retrieved in 19.May.2016.

-
- [Mic16] Microsoft Corporation (2016): *Overview of the.NET Framework*. <https://msdn.microsoft.com/en-us/library/zw4w595w.aspx>, retrieved in 03.October.2016.
- [Mic162] Microsoft AG: *How to: Use the XML Schema Definition Tool to Generate Classes and XML Schema Documents*. [https://msdn.microsoft.com/en-us/library/5s2x1sy7\(v=vs.100\)](https://msdn.microsoft.com/en-us/library/5s2x1sy7(v=vs.100)), retrieved in 03.Juli.2016.
- [Mih14] Mihelj M.; Novak D. and Begus S. (2014): *Introduction to Virtual Reality*. In *Virtual Reality Technology and Applications*. Springer Netherlands. p. 1-16.
- [MOD14] Modelica Association Project "FMI" (2014): *Functional Mock-up Interface for Model Exchange and Co-Simulation*. https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf, retrieved in 29.December.2015.
- [Msc16] Microsoft Corporation (2016): *Introducing Visual Studio*. <https://msdn.microsoft.com/en-us/library/6x6bk1f4%28v=vs.100%29.aspx>, retrieved in 09.June.2016.
- [Obj12] Object Management Group (2012): *OMG Systems Modeling Language (OMG SysML™) specification*. <http://www.omg.org/spec/SysML/1.3/PDF>, retrieved in 28.June.2016.
- [Ous98] Ousterhout J.K. (1998): *Scripting: Higher level programming for the 21st century*. In: *Computer* (31(3)). p. 23-30.
- [Pah13] Pahl G. and Beitz W. (2013): *Engineering design: a systematic approach*. Springer Science & Business Media.
- [Par08] Park S.C.; Park C.M.; Wang G.N. et al. (2008): *PLCStudio: Simulation based PLC code verification*. In: *Simulation Conference, 2008. WSC 2008. Winter. IEEE*. p. 222-228.
- [Paw14] Pawellek G. (2014): *Strukturplanung*. In: *Ganzheitliche Fabrikplanung: Grundlagen, Vorgehensweise, EDV-Unterstützung*. Springer-Verlag. p. 149-185.
- [Pri04] Pritschow G. and Röck S. (2004): *"Hardware in the Loop" Simulation of Machine Tools*. In: *CIRP Annals-Manufacturing Technology* 53(1), p. 295-298.

- [Pro14] Proesser M. (2014): *A New Approach to Systems Integration in the Mechatronic Engineering Design Process of Manufacturing Systems*. (Doctoral thesis) Leicester.
- [Ran11] Rank S. (2011): *Virtuelle Realität im Rahmen der Planung von Materialflusssystemen*. (Diplomarbeit) Dresden.
- [Rei071] Reinhart G. and Wunsch G. (2007): *Economic application of virtual commissioning to mechatronic production systems*. In: *Production Engineering*, 1(4), p. 371-379.
- [Rei08] Reif R. and Walch D. (2008): *Augmented & Virtual Reality applications in the field of logistics*. In: *The Visual Computer*, 24(11), p. 987-994.
- [Ren16] Rensselaer Polytechnic Institute (2016): *Mechatronics*. Rensselaer. <http://homepages.rpi.edu/~hurstj2/>, retrieved in 03.October.2016.
- [Ric90] Verein deutscher Ingenieure (VDI) (1990): *VDI 2860: Handhabungsfunktionen, Handhabungseinrichtungen: Begriffe, Definitionen, Symbole*. Düsseldorf.
- [Sau04] Sauer O. (2004): *Einfluss der Digitalen Fabrik auf die Fabrikplanung*. In: *wt werkstattstechnik online*, p. 31-34.
https://www.brainguide.de/upload/publication/f1/jvk8/5eed62be7d958f969448a597c66c533a_1311535234.pdf, retrieved in 09. October 2016.
- [Saw16] Sawant A.V. and Nazemetz J.W.: *Impediments to Data Exchange and Use of STEP for Data Exchange*. Oklahoma State University.
<http://www.okstate.edu/ind-engr/step/WEBFILES/Papers/Impediments.html>, retrieved in 16.June.2016.
- [ScDr09] Schleipen M. and Drath R. (2009): *Three-View-Concept for modeling process or manufacturing plants with AutomationML*. In: *Emerging Technologies & Factory Automation, 2009. ETFA 2009..*
- [Sch09] Schob U.; Blochwitz T.; Oelsner O. et al. (2009): *Model Based Virtual Startup of Automation Systems*. In: *7th International Modelica Conference*. p. 790-796.
- [Sei12] Seidel S.; Donath U. and Haufe J. (2012): *Towards an integrated simulation and virtual commissioning environment for controls of material handling systems*. In: *Proceedings of the 2012 Winter Simulation Conference*.

- [Sie10] Siemens Product Lifecycle Management Software Inc. (2010): *JT File Format Reference*.
http://www.freecadweb.org/tracker/file_download.php?file_id=503&type=bug, retrieved in 28.June.2016.
- [Sko11] Skoogh A. (2011): *Automation of Input Data Management. Increasing Efficiency in Simulation of Production Flows*. (Doctoral thesis) Gothenburg, Sweden.
- [Spi09] Spitzweg M. (2009): *Methode und Konzept für den Einsatz eines physikalischen Modells in der Entwicklung von Produktionsanlagen*. (Doctoral thesis) Munich.
- [Str12] Strahilov A.; Ovtcharova J. and Bär T. (2012): *Development of the physics-based assembly system model for the mechatronic validation of automated assembly systems*. In: Simulation Conference (WSC), Proceedings of the 2012 Winter. IEEE.
- [Str15] Strahilov A. (2015): *Simulation des physikalischen Verhaltens bei der digitalen Absicherung von automatisierten Montageanlagen*. (Doctoral thesis) Berlin.
- [tar08] tarakos GmbH (2008): *taraVRControl Manual*. Magdeburg.
- [tar15] tarakos GmbH (2015): *taraVRBuilder Manual Version 12.0*. Magdeburg.
- [Tas02] Tasse G. (2002): *The economic impacts of inadequate infrastructure for software testing*. NIST U.S. Department of Commerce Technology Administration.
<https://www.nist.gov/sites/default/files/documents/director/planning/report02-3.pdf>, retrieved in 03.October.2016.
- [Tec16] TechTarget: *File format*. <http://whatis.techtarget.com/definition/file-format>, retrieved in 14.June.2016.
- [Tet09] Tetzner T. and Gewalt N. (2009): *Mechatronisches Konzept im Engineering von Industrieanlagen-Anforderungen aus Anwendersicht*. In: Symposium Informationstechnologien für Entwicklung und Produktion in der Verfahrenstechnik, March. Vol 26. p. 27.

- [The13] The Khronos Group (2013): *COLLADA. Digital Asset and FX Exchange Schema*. https://www.khronos.org/collada/wiki/Portal:Products_directory, retrieved in 15.May.2016.
- [VDI08] VDI (2008): *Digitale Fabrik Grundlagen VDI-Richtlinie 4499, Blatt 1*.
- [VDI93] VDI Richtlinie (1993): *VDI Richtlinie 3633 Blatt 1: Simulation von Logistik-, Materialfluß- und Produktionssystemen*. Berlin: Beuth.
- [Ver11] Verein Deutscher Ingenieure (2011): *XML in der Automation Klassifikation ausgewählter Anwendungen*. Düsseldorf: VDI, VDI/VDE 3690.
- [Ver701] Verein Deutscher Ingenieure (VDI) (1970): *VDI 2411: Begriffe und Erläuterungen im Förderwesen*. Berlin.
- [Wal98] Walsh N. (1998): *A technical introduction to XML*. O'Reilly & Associates, Inc. <http://www.math.unipd.it/~basidati/docs/introduction.pdf>, retrieved in 22.May.2016.
- [Web13] Web3D Consortium (2013): *X3D Architecture and base components V3*. <http://www.web3d.org/documents/specifications/19775-1/V3.3/index.html>, retrieved in 26.March.2016.
- [Wei98] Weiss P.L. and Jessel A.S. (1998): *Virtual reality applications to work*. In: *Work*, 11(3), p. 277-293.
- [Weu98] Weustink P.B.T.; De Vries T.J.A. and Breedveld P.C. (1998): *Object-oriented modeling and simulation of mechatronic systems with 20-sim 3.0*. In: *Mechatronics*, 98, p. 873-787.
- [Wid09] Widemann D. and Drath R. (2010): Einleitung. In Drath R. (Hg.), *Datenaustausch in der Anlagenplanung mit AutomationML: Integration von CAEX, PLCopen XML und COLLADA*. Berlin Heidelberg: Springer-Verlag. S. 1-43.
- [Wor00] World Wide Web Consortium (2000): *XML Schema*. <https://www.w3.org/XML/Schema>, retrieved in 20.August.2016.
- [Yem11] Yemenicioglu E. (2011): *Representation of Graphical and Kinematic Information as Part of the Mechatronic Engineering Process in the Factory Automation*. (Master thesis) Otto-von-Guericke University Magdeburg - Faculty of Mechanical Engineering, Magdeburg.

- [Zäh06] Zäh M.F.; Wunsch G.; Hensel T. et al. (2006): *Feldstudie–Virtuelle Inbetriebnahme*. In: wt Werkstattstechnik online, Jg, 96, p. 767-771.
- [Zim05] Zimmer H. (2005): *Voronoi and Delaunay Techniques*. <http://elliptic-pde.googlecode.com/hg-history/0d72f611e1ad6e1ad7a1ce435b9ed167572b99a8/papers/VoronoiDelaunay.pdf>, retrieved in 21.03.2016.
- [Žiž06] Žižek S. (2006): *The matrix, or, the two sides of perversion*. <http://www.lacan.com/zizek-matrix.htm>, retrieved in 09. October 2016.

Addendum

Appendix A Material Handling Role Class and Interface Libraries in AutomationML

```

<?xml version="1.0" encoding="utf-8"?>

<CAEXFile                               FileName="MaterialHandlingRoleClassLibrary.aml"
SchemaVersion="2.15"
xsi:noNamespaceSchemaLocation="CAEX_ClassModel_V2.15.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <AdditionalInformation>
    <WriterHeader>
      <WriterName>taraVRbuilder</WriterName>
      <WriterID>taraVRbuilder</WriterID>
      <WriterVendor>tarakos</WriterVendor>
      <WriterVendorURL>www.tarakos.de</WriterVendorURL>
      <WriterVersion>2016</WriterVersion>
      <WriterRelease>12.1.5</WriterRelease>
      <LastWritingDateTime>2016-05-19T10:55:39Z</LastWritingDateTime>
      <WriterProjectTitle>MaterialHandlingLibrary
    </WriterProjectTitle>
      <WriterProjectID>MaterialHandlingLibrary</WriterProjectID>
    </WriterHeader>
  </AdditionalInformation>
  <AdditionalInformation AutomationMLVersion="2.0" />
  <InterfaceClassLib Name="MaterialsHandlingInterfaceClassLib"
ChangeMode="create">
    <Description>Material flow interface classes</Description>
    <Version>1.0.0</Version>
    <InterfaceClass Name="MaterialHandlingProductionProcessConnector"
RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface"
ChangeMode="create">
      <Description>Defines the connection between material handling
level and production process level</Description>
    </InterfaceClass>
    <InterfaceClass Name="MaterialHandlingInterface"
RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/O
rder" ChangeMode="create">
      <Description>General interface to define the connection of
material flow elements</Description>
    <InterfaceClass Name="MaterialHandlingConnector"
RefBaseClassPath="MaterialsHandlingInterfaceClassLib/MaterialHandlingInterf
ace" ChangeMode="create">
      <Description>Connects the material handling components
directly with each other (1. level)</Description>
    </InterfaceClass>
    <InterfaceClass Name="MaterialHandlingConnectionPointConnector"
RefBaseClassPath="MaterialsHandlingInterfaceClassLib/MaterialHandlingInterf
ace" ChangeMode="create">
      <Description>Connects the material handling components
through connection points (2. level)</Description>
    </InterfaceClass>
  </InterfaceClassLib>

```

```

    <RoleClassLib Name="MaterialsHandlingClassLib" ChangeMode="change">
      <Description>Role class library for material flow
components</Description>
      <Version>1.1</Version>
      <RoleClass Name="Turner"
RefBaseClassPath="AutomationMLDMIRoleClassLib/DiscManufacturingEquipment/Tr
ansport" ChangeMode="create">
        <Description>A general functional role for the elements, which
are turning materials without any position change</Description>
        <Attribute Name="Range" Unit="deg"
AttributeDataType="xs:double" ChangeMode="create">
          <Description>Range of the turner</Description>
        </Attribute>
        <Attribute Name="MaximumLoad" Unit="kg"
AttributeDataType="xs:double" ChangeMode="create">
          <Description>Maximum sustainable load</Description>
        </Attribute>
        <Attribute Name="MaximumAngularAcceleration" Unit="rad/s^2"
AttributeDataType="xs:double" ChangeMode="create">
          <Description>Angular acceleration of the
rotation</Description>
        </Attribute>
        <Attribute Name="MaximumAngularVelocity" Unit="rad/s"
AttributeDataType="xs:double" ChangeMode="create">
          <Description>Angular velocity of the rotation</Description>
        </Attribute>
        <RoleClass Name="VerticalTurner"
RefBaseClassPath="MaterialsHandlingClassLib/Turner" ChangeMode="create">
          <Description>An element, which is turning materials on
vertical axis</Description>
        </RoleClass>
        <RoleClass Name="LateralTurner"
RefBaseClassPath="MaterialsHandlingClassLib/Turner" ChangeMode="create">
          <Description>An element, which is turning materials on
lateral axis</Description>
        </RoleClass>
        <RoleClass Name="LongitudinalTurner"
RefBaseClassPath="MaterialsHandlingClassLib/Turner" ChangeMode="create">
          <Description>An element, which is turning materials on
longitudinal axis</Description>
        </RoleClass>
      </RoleClass>
      <RoleClass Name="MaterialHandlingConveyor"
RefBaseClassPath="AutomationMLExtendedRoleClassLib/Conveyor"
ChangeMode="create">
        <Description>A functional role for the elements, which move
materials from one location to another but does not change its own
place</Description>
        <Attribute Name="Velocity" AttributeDataType="xs:double"
Unit="m/s" ChangeMode="create">
          <Description>Velocity of the conveyor</Description>
        </Attribute>
        <Attribute Name="Width" AttributeDataType="xs:double" Unit="m"
ChangeMode="create">
          <Description>Usable width of the conveyor</Description>
        </Attribute>
        <RoleClass Name="StraightConveyor"
RefBaseClassPath="MaterialsHandlingClassLib/MaterialHandlingConveyor"
ChangeMode="create">
          <Description>Moves materials from one location to another
on a straight line</Description>

```

```

        <Attribute Name="Length" AttributeDataType="xs:double"
Unit="m" ChangeMode="create">
        <Description>Length of the conveyor</Description>
        </Attribute>
        <Attribute Name="Tilt" AttributeDataType="xs:double"
Unit="deg" ChangeMode="create">
        <Description>Rotational angle over the y-
axis</Description>
        </Attribute>
        <RoleClass Name="Bevel" ChangeMode="create"
RefBaseClassPath="MaterialsHandlingClassLib/MaterialHandlingConveyor/Straig
htConveyor">
        <Description>Moves materials from one location to
another and changes the direction of output with an angle</Description>
        <Attribute Name="Angle" AttributeDataType="xs:double"
Unit="deg">
        <Description>Angle of bevel</Description>
        <DefaultValue>45</DefaultValue>
        </Attribute>
        </RoleClass>
    </RoleClass>
    <RoleClass Name="CurvedConveyor"
RefBaseClassPath="MaterialsHandlingClassLib/MaterialHandlingConveyor"
ChangeMode="create">
        <Description>Moves materials from one location to another
on a curve</Description>
        <Attribute Name="Radius" AttributeDataType="xs:double"
ChangeMode="create" Unit="m">
        <Description>Center radius</Description>
        </Attribute>
        <Attribute Name="Angle" AttributeDataType="xs:double"
Unit="deg" ChangeMode="create">
        <Description>Angle of the curve</Description>
        </Attribute>
        <RoleClass Name="SpiralConveyor" ChangeMode="create"
RefBaseClassPath="MaterialsHandlingClassLib/MaterialHandlingConveyor/Curved
Conveyor">
        <Description>Moves materials from one location to
another on a spiral</Description>
        <Attribute Name="Height" Unit="m"
AttributeDataType="xs:double" ChangeMode="create">
        <Description>Height of the spiral</Description>
        <DefaultValue>1</DefaultValue>
        </Attribute>
        </RoleClass>
    </RoleClass>
    <RoleClass Name="VerticalConveyor" ChangeMode="create"
RefBaseClassPath="MaterialsHandlingClassLib/MaterialHandlingConveyor">
        <Description>Moves materials from one location to another
on a line flapped around the connection point</Description>
        <Attribute Name="TiltList" Unit="deg">
        <Description>List of tilt angles</Description>
        <RefSemantic
CorrespondingAttributePath="OrderedListType" />
        </Attribute>
        <Attribute Name="SwingVelocity" Unit="m/s"
AttributeDataType="xs:double">
        <Description>Velocity of the swinging from one tilt
angle to another</Description>
        </Attribute>

```

```

        <Attribute Name="Length" AttributeDataType="xs:double"
Unit="m" ChangeMode="create">
        <Description>Length of the conveyor</Description>
        </Attribute>
    </RoleClass>
</RoleClass>
<RoleClass Name="Positioner"
RefBaseClassPath="AutomationMLDMIRoleClassLib/DiscManufacturingEquipment/Tr
ansport" ChangeMode="create">
    <Description>A functional role for the elements, which carry
materials from one location to another. (Positioner has also location
change.)</Description>
    <Attribute Name="MaximumLoad" AttributeDataType="xs:double"
Unit="kg" ChangeMode="create">
        <Description>Maximum sustainable load</Description>
        </Attribute>
        <Attribute Name="MaximumAcceleration"
AttributeDataType="xs:double" Unit="m/s^2" ChangeMode="create">
            <Description>Maximum acceleration of the linear
movement</Description>
            </Attribute>
            <Attribute Name="MaximumVelocity" AttributeDataType="xs:double"
Unit="m/s" ChangeMode="create">
                <Description>Maximum velocity of the linear
movement</Description>
                </Attribute>
                <RoleClass Name="LinearPositioner"
RefBaseClassPath="MaterialsHandlingClassLib/Positioner"
ChangeMode="create">
                    <Description>A positioner, which carries materials on a
linear axis</Description>
                    <Attribute Name="Range" AttributeDataType="xs:double"
Unit="m" ChangeMode="create">
                        <Description>Range of the positioner</Description>
                        </Attribute>
                        <RoleClass Name="VerticalPositioner"
RefBaseClassPath="MaterialsHandlingClassLib/Positioner/LinearPositioner"
ChangeMode="create">
                            <Description>A positioner, which carries materials on
linear vertical axis</Description>
                            </RoleClass>
                            <RoleClass Name="LateralPositioner"
RefBaseClassPath="MaterialsHandlingClassLib/Positioner/LinearPositioner"
ChangeMode="create">
                                <Description>A positioner, which carries materials on
linear lateral axis</Description>
                                </RoleClass>
                                <RoleClass Name="LongitudinalPositioner"
RefBaseClassPath="MaterialsHandlingClassLib/Positioner/LinearPositioner"
ChangeMode="create">
                                    <Description>A positioner, which carries materials on
linear longitudinal axis</Description>
                                    </RoleClass>
                                    </RoleClass>
                                    <RoleClass Name="ArealPositioner"
RefBaseClassPath="MaterialsHandlingClassLib/Positioner"
ChangeMode="create">
                                        <Description>A positioner, which carries materials in a
defined area</Description>
                                        <Attribute Name="Direction" Unit="m[]" ChangeMode="create">

```



```

        <Description>Direction of the movement as
vector</Description>
        <RefSemantic
CorrespondingAttributePath="OrderedListType" />
        </Attribute>
        <Attribute Name="Range" AttributeDataType="xs:double"
Unit="m" ChangeMode="create">
        <Description>Range of the positioner</Description>
        </Attribute>
    </RoleClass>
    <RoleClass Name="FreePositioner"
RefBaseClassPath="MaterialsHandlingClassLib/Positioner"
ChangeMode="create">
        <Description>A positioner, which can carry materials in 3D
space</Description>
        <Attribute Name="TaskSpace" ChangeMode="create">
        <Description>The task space region of the
positioner</Description>
        <Attribute Name="x_min" AttributeDataType="xs:double"
ChangeMode="create" Unit="m">
            <Description>Minimum x value</Description>
        </Attribute>
        <Attribute Name="x_max" AttributeDataType="xs:double"
ChangeMode="create" Unit="m">
            <Description>Maximum x value</Description>
        </Attribute>
        <Attribute Name="y_min" AttributeDataType="xs:double"
ChangeMode="create" Unit="m">
            <Description>Minimum y value</Description>
        </Attribute>
        <Attribute Name="y_max" AttributeDataType="xs:double"
ChangeMode="create" Unit="m">
            <Description>Maximum y value</Description>
        </Attribute>
        <Attribute Name="z_min" AttributeDataType="xs:double"
ChangeMode="create" Unit="m">
            <Description>Minimum z value</Description>
        </Attribute>
        <Attribute Name="z_max" AttributeDataType="xs:double"
ChangeMode="create" Unit="m">
            <Description>Maximum z value</Description>
        </Attribute>
    </RoleClass>
    </RoleClass>
    <RoleClass Name="ConveyorTechnology"
RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Resourc
e" ChangeMode="create">
        <Description>Defines the carrier element of a conveyor like
rolls, belts, chains, etc.</Description>
        <RoleClass Name="BeltCarrier"
RefBaseClassPath="MaterialsHandlingClassLib/ConveyorTechnology"
ChangeMode="create">
            <Description>Belt type carrier element</Description>
        </RoleClass>
        <RoleClass Name="ChainCarrier"
RefBaseClassPath="MaterialsHandlingClassLib/ConveyorTechnology"
ChangeMode="create">
            <Description>Chain type carrier element</Description>
        </RoleClass>

```

```

    <RoleClass Name="SkidCarrier"
RefBaseClassPath="MaterialsHandlingClassLib/ConveyorTechnology"
ChangeMode="create">
    <Description>Skid type carrier element</Description>
</RoleClass>
    <RoleClass Name="RollCarrier"
RefBaseClassPath="MaterialsHandlingClassLib/ConveyorTechnology"
ChangeMode="create">
    <Description>Roll type carrier element</Description>
</RoleClass>
    <RoleClass Name="LinkBeltCarrier"
RefBaseClassPath="MaterialsHandlingClassLib/ConveyorTechnology"
ChangeMode="create">
    <Description>Link belt type carrier element</Description>
</RoleClass>
    <RoleClass Name="BallCarrier"
RefBaseClassPath="MaterialsHandlingClassLib/ConveyorTechnology"
ChangeMode="create">
    <Description>Ball type carrier element</Description>
</RoleClass>
    <RoleClass Name="WheelCarrier"
RefBaseClassPath="MaterialsHandlingClassLib/ConveyorTechnology"
ChangeMode="create">
    <Description>Wheel type carrier element</Description>
</RoleClass>
</RoleClass>
    <RoleClass Name="MaterialHandlingStorage"
RefBaseClassPath="AutomationMLDMIRoleClassLib/DiscManufacturingEquipment/St
orage" ChangeMode="create">
    <Description>Represents storage for materials</Description>
    <AdditionalInformation>see VDI 2860
(Speichern)</AdditionalInformation>
    <RoleClass Name="SortedStorage"
RefBaseClassPath="MaterialsHandlingClassLib/MaterialHandlingStorage"
ChangeMode="create">
    <Description>Represents a storage, in which all
translational and rotational degrees of freedom for materials are
defined</Description>
    <AdditionalInformation>see VDI 2860 (Geordnetes
Speichern)</AdditionalInformation>
</RoleClass>
    <RoleClass Name="PartlySortedStorage"
RefBaseClassPath="MaterialsHandlingClassLib/MaterialHandlingStorage"
ChangeMode="create">
    <Description>Represents a storage, in which some
translational and rotational degrees of freedom for materials are
defined</Description>
    <AdditionalInformation>see VDI 2860 (Teilgeordnetes
Speichern)</AdditionalInformation>
</RoleClass>
    <RoleClass Name="UnsortedStorage"
RefBaseClassPath="MaterialsHandlingClassLib/MaterialHandlingStorage"
ChangeMode="create">
    <Description>Represents a storage, in which translational
and rotational degrees of freedom for materials are arbitrary</Description>
    <AdditionalInformation>see VDI 2860 (Ungeordnetes
Speichern)</AdditionalInformation>
</RoleClass>
</RoleClass>

```

```
<RoleClass Name="MaterialHandlingConnectionPoint"
RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Structure/ResourceStructure" ChangeMode="create">
  <Description>Represent the point where the material transfer occurs</Description>
  <ExternalInterface Name="MaterialFlowConnection"
RefBaseClassPath="MaterialsHandlingInterfaceClassLib/MaterialHandlingInterface/MaterialHandlingConnectionPointConnector" ID="9d8b88a2-a251-452a-bc96-a3b5e8e78ca8" ChangeMode="create">
    <Description>Interface for linking the connection point with another one</Description>
  </ExternalInterface>
</RoleClass>
<RoleClass Name="ConveyorAttachments"
RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Resource" ChangeMode="create">
  <Description>A general role for conveyor attachments</Description>
  <RoleClass Name="Stopper"
RefBaseClassPath="MaterialsHandlingClassLib/ConveyorAttachments" ChangeMode="create">
    <Description>Decelerates or stops materials safely</Description>
  </RoleClass>
  <RoleClass Name="Pusher"
RefBaseClassPath="MaterialsHandlingClassLib/ConveyorAttachments" ChangeMode="create">
    <Description>Diverts materials from one conveyor line to another</Description>
  </RoleClass>
</RoleClass>
<RoleClass Name="EndEffector"
RefBaseClassPath="AutomationMLDMIRoleClassLib/DiscManufacturingEquipment/Tool" ChangeMode="create">
  <Description>Represents a device at the end of a manipulator, designed to interact with the environment.</Description>
  <RoleClass Name="Gripper"
RefBaseClassPath="MaterialsHandlingClassLib/EndEffector" ChangeMode="create">
    <Description>Represents a device which enables the holding of an object to be manipulated</Description>
    <Attribute Name="GrippingForce"
AttributeDataType="xs:double" Unit="N" ChangeMode="create">
      <Description>Gripping force of the handling element</Description>
    </Attribute>
    <Attribute Name="Weight" AttributeDataType="xs:double" Unit="kg" ChangeMode="create">
      <Description>Weight of the gripper</Description>
    </Attribute>
    <RoleClass Name="HydraulicGripper"
RefBaseClassPath="MaterialsHandlingClassLib/EndEffector" ChangeMode="create">
      <Description>Represents a hydraulic driven gripper</Description>
    </RoleClass>
    <RoleClass Name="VacuumGripper"
RefBaseClassPath="MaterialsHandlingClassLib/EndEffector" ChangeMode="create">
      <Description>Represents a vacuum based gripping mechanism</Description>
    </RoleClass>
  </RoleClass>
</RoleClass>
```

```

        </RoleClass>
        <RoleClass Name="ServoElectricGripper"
RefBaseClassPath="MaterialsHandlingClassLib/EndEffector"
ChangeMode="create">
            <Description>Represents a servo electric driven
gripper</Description>
        </RoleClass>
        <RoleClass Name="MagneticGripper"
RefBaseClassPath="MaterialsHandlingClassLib/EndEffector"
ChangeMode="create">
            <Description>Represents a magnetism based gripping
mechanism</Description>
        </RoleClass>
        <RoleClass Name="PneumaticGripper"
RefBaseClassPath="MaterialsHandlingClassLib/EndEffector"
ChangeMode="create">
            <Description>Represents a pneumatic driven
gripper</Description>
        </RoleClass>
        </RoleClass>
        </RoleClass>
        <RoleClass Name="DriveTechnology"
RefBaseClassPath="AutomationMLCSRClassLib/ControlEquipment/Actuator"
ChangeMode="create">
            <Description>A general role for the driving technology in
material flow. Could be electrical, pneumatical or
hydraulic.</Description>
            <Attribute Name="MomentOfInertia" AttributeDataType="xs:double"
Unit="kgm^2" ChangeMode="create">
                <Description>Moment of inertia for the load</Description>
            </Attribute>
        </RoleClass>
        <RoleClass Name="LoadingEquipment"
RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Product
" ChangeMode="create">
            <Description>A general role for loading equipments like
containers, pallets, boxes. They move together with the product on the
production line. It is also possible that they are delivered together with
the product. That is an element which has the characteristics of both the
product and the ressource.</Description>
            <Attribute Name="Length" AttributeDataType="xs:double" Unit="m"
ChangeMode="create">
                <Description>Length of the loading equipment</Description>
            </Attribute>
            <Attribute Name="Width" AttributeDataType="xs:double" Unit="m"
ChangeMode="create">
                <Description>Width of the loading equipment</Description>
            </Attribute>
            <Attribute Name="Height" AttributeDataType="xs:double" Unit="m"
ChangeMode="create">
                <Description>Height of the loading equipment</Description>
            </Attribute>
            <Attribute Name="Capacity" AttributeDataType="xs:double"
Unit="kg" ChangeMode="create">
                <Description>Capacity of the loading
equipment</Description>
            </Attribute>
        </RoleClass Name="Container"
RefBaseClassPath="MaterialsHandlingClassLib/LoadingEquipment"
ChangeMode="create">

```

```

        <Description>Represents any container for
materials</Description>
        </RoleClass>
        <RoleClass Name="Pallet"
RefBaseClassPath="MaterialsHandlingClassLib/LoadingEquipment"
ChangeMode="create">
        <Description>Represents any pallet for
materials</Description>
        </RoleClass>
        <RoleClass Name="Box"
RefBaseClassPath="MaterialsHandlingClassLib/LoadingEquipment"
ChangeMode="create">
        <Description>Represents box for materials</Description>
        </RoleClass>
        </RoleClass>
        <RoleClass Name="ComboElement"
RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Structu
re/ResourceStructure" ChangeMode="create">
        <Description>A marker role to show that the component consists
of subcomponents</Description>
        </RoleClass>
        </RoleClassLib>
        <RoleClassLib Name="MaterialsHandlingProcessClassLib"
ChangeMode="create">
        <Description>Role class library for material flow processes. Mostly
adopted from VDI 2860.</Description>
        <Version>1.0.0</Version>
        <RoleClass Name="Behavior"
RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Process
" ChangeMode="create">
        <Description>Defines handling processes for the distribution,
routing, usw.</Description>
        <RoleClass Name="Congestion"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Behavior"
ChangeMode="create">
        <Description>Defines congestion behavior</Description>
        </RoleClass>
        <RoleClass Name="Priority"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Behavior"
ChangeMode="create">
        <Description>Defines the priority in the material
flow</Description>
        </RoleClass>
        <RoleClass Name="Consolidation"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Behavior"
ChangeMode="create">
        <Description>Defines consolidation of multiple material
handling objects</Description>
        <AdditionalInformation>see VDI 2860
(Zusammenführen)</AdditionalInformation>
        </RoleClass>
        <RoleClass Name="Sorting"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Behavior"
ChangeMode="create">
        <Description>Defines the sorting process in material
handling</Description>
        <AdditionalInformation>see VDI 2860
(Sortieren)</AdditionalInformation>
        </RoleClass>

```

```

    <RoleClass Name="Dividing"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Behavior"
ChangeMode="create">
    <Description>A general process role to define how to create
subsets from an amount</Description>
    <AdditionalInformation>see VDI 2860
(Teilen)</AdditionalInformation>
    <RoleClass Name="Partition"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Behavior/Dividing"
ChangeMode="create">
    <Description>Defines a process to form subsets with
defined size</Description>
    <AdditionalInformation>see VDI 2860
(Abteilen)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="Distribution"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Behavior/Dividing"
ChangeMode="create">
    <Description>Defines a process to form subsets with
defined size and transfer them to a defined place</Description>
    <AdditionalInformation>see VDI 2860
(Zuteilen)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="Branching"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Behavior/Dividing"
ChangeMode="create">
    <Description>Defines a process to split a material flow
to smaller flows</Description>
    <AdditionalInformation>see VDI
2860 (Verzweigen)</AdditionalInformation>
    </RoleClass>
    </RoleClass>
    </RoleClass>
    <RoleClass Name="Moving"
RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Process
" ChangeMode="create">
    <Description>Defines a changement in the spatial arrangement of
a body</Description>
    <AdditionalInformation>see VDI
2860 (Bewegen)</AdditionalInformation>
    <RoleClass Name="Turning"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Moving"
ChangeMode="create">
    <Description>Changing the orientation of the body without
changing the position</Description>
    <AdditionalInformation>see VDI 2860
(Drehen)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="Translating"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Moving"
ChangeMode="create">
    <Description>Moving a body in a linear direction without
changing the orientation</Description>
    <AdditionalInformation>see VDI 2860
(Verschieben)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="Swinging"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Moving"
ChangeMode="create">

```

```

        <Description>Rotation of a body over a center of an axis,
which is not within the body, resulting an orientation and position
change</Description>
        <AdditionalInformation>see VDI 2860
(Schwenken)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="Orientating" ChangeMode="create"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Moving">
        <Description>Change of the orientation from an undefined to
a defined one. Position change of the body is disregarded.</Description>
        <AdditionalInformation>see VDI 2860
(Orientieren)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="Positioning"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Moving"
ChangeMode="create">
        <Description>Moving a body to a defined position.
Orientation of the body is disregarded.</Description>
        <AdditionalInformation>see VDI 2860
(Positionieren)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="Arranging"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Moving"
ChangeMode="create">
        <Description>Moving a body from an undefined place to a
defined orientation and position providing arrangement.</Description>
        <AdditionalInformation>see VDI 2860
(Ordnen)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="Directing"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Moving"
ChangeMode="create">
        <Description>Moving a body on a defined path from a defined
place to another defined place</Description>
        <AdditionalInformation>see VDI 2860
(Führen)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="HandingOver"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Moving"
ChangeMode="create">
        <Description>Moving a body on an undefined path from a
defined place to another defined place</Description>
        <AdditionalInformation>see VDI 2860
(Weitergeben)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="Conveying"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Moving"
ChangeMode="create">
        <Description>Moving a body from any place to another place.
Orientation and position of the body during the movement is not necessarily
defined.</Description>
        <AdditionalInformation>see VDI 2860
(Fördern)</AdditionalInformation>
    </RoleClass>
    </RoleClass>
    <RoleClass Name="Securing"
RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Process
" ChangeMode="create">
        <Description>A general role to define securing a defined
status</Description>

```

```

    <AdditionalInformation>see VDI 2860
(Sichern)</AdditionalInformation>
    <RoleClass Name="Holding"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Securing"
ChangeMode="create">
    <Description>Securing the orientation and position of a
body</Description>
    <AdditionalInformation>see VDI 2860
(Halten)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="Loosening"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Securing"
ChangeMode="create">
    <Description>Loosening the orientation and position of a
body, reversal of holding.</Description>
    <AdditionalInformation>see VDI 2860
(Lösen)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="Tightening"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Securing"
ChangeMode="create">
    <Description>Securing the orientation and position of a
body with applying force</Description>
    <AdditionalInformation>see VDI 2860
(Spannen)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="Relieving"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Securing"
ChangeMode="create">
    <Description>Relieving the force which is blocking the
orientational and positional change, reversal of tightening.</Description>
    <AdditionalInformation>see VDI 2860
(Entspannen)</AdditionalInformation>
    </RoleClass>
    </RoleClass>
    <RoleClass Name="Verification"
RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Process
" ChangeMode="create">
    <Description>Defines a general role for inspection and
measuring of properties and status</Description>
    <AdditionalInformation>see VDI 2860
(Kontrollieren)</AdditionalInformation>
    <RoleClass Name="Inspection"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification"
ChangeMode="create">
    <Description>Inspection of properties and status. Admission
of information, comparison with target properties, status or
decisions.</Description>
    <AdditionalInformation>see VDI 2860
(Prüfen)</AdditionalInformation>
    <RoleClass Name="PresenceCheck"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification/Inspection"
ChangeMode="create">
    <Description>Determining if a body is present in a
defined place</Description>
    <AdditionalInformation>see VDI 2860 (Anwesenheit
prüfen)</AdditionalInformation>
    </RoleClass>
    <RoleClass Name="IdentityCheck"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification/Inspection"
ChangeMode="create">

```



```

        <Description>Determining if a body satisfies the
defined properties</Description>
        <AdditionalInformation>see VDI 2860 (Identität
prüfen)</AdditionalInformation>
        </RoleClass>
        <RoleClass Name="ShapeCheck"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification/Inspection"
ChangeMode="create">
        <Description>Determining if a body has the defined
shape.</Description>
        <AdditionalInformation>see VDI 2860 (Form
prüfen)</AdditionalInformation>
        </RoleClass>
        <RoleClass Name="SizeCheck"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification/Inspection"
ChangeMode="create">
        <Description>Determining if a body has the defined
sizes/measurements.</Description>
        <AdditionalInformation>see VDI 2860 (Größe
prüfen)</AdditionalInformation>
        </RoleClass>
        <RoleClass Name="ColorCheck"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification/Inspection"
ChangeMode="create">
        <Description>Determining if a body or body areas have
the defined colors</Description>
        <AdditionalInformation>see VDI 2860 (Farbe
prüfen)</AdditionalInformation>
        </RoleClass>
        <RoleClass Name="WeightCheck"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification/Inspection"
ChangeMode="create">
        <Description>Determining if a body has the defined
weight</Description>
        <AdditionalInformation>see VDI 2860 (Gewicht
prüfen)</AdditionalInformation>
        </RoleClass>
        <RoleClass Name="PositionCheck"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification/Inspection"
ChangeMode="create">
        <Description>Determining if a body has the defined
position</Description>
        <AdditionalInformation>see VDI 2860 (Position
prüfen)</AdditionalInformation>
        </RoleClass>
        <RoleClass Name="OrientationCheck"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification/Inspection"
ChangeMode="create">
        <Description>Determining if a body has the defined
orientation</Description>
        <AdditionalInformation>see VDI 2860 (Orientatierung
prüfen)</AdditionalInformation>
        </RoleClass>
        </RoleClass>
        <RoleClass Name="Measuring"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification"
ChangeMode="create">
        <Description>Measuring the values in accordance to
reference values</Description>

```

```

        <RoleClass Name="Counting"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification/Measuring"
ChangeMode="create">
        <Description>Determining the amount of
bodies</Description>
        <AdditionalInformation>see VDI 2860
(Zählen)</AdditionalInformation>
        </RoleClass>
        <RoleClass Name="OrientationMeasuring"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification/Measuring"
ChangeMode="create">
        <Description>Determining the orientation of a body in
accordance to a reference coordinate system</Description>
        <AdditionalInformation>see VDI 2860 (Orientation
messen)</AdditionalInformation>
        </RoleClass>
        <RoleClass Name="PositionMeasuring"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Verification/Measuring"
ChangeMode="create">
        <Description>Determining the position of a body in
accordance to a reference coordinate system</Description>
        <AdditionalInformation>see VDI 2860 (Position
messen)</AdditionalInformation>
        </RoleClass>
        </RoleClass>
        <RoleClass Name="Source&Sink"
RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Process
" ChangeMode="create">
        <Description>General role to define source and sink
processes</Description>
        <RoleClass Name="Source"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Source&Sink"
ChangeMode="create">
        <Description>Defines the source of a material
flow</Description>
        <Attribute Name="OutputMode" AttributeDataType="xs:string"
ChangeMode="create">
        <Description>Creating goods randomly or
in a synchronized manner</Description>
        <DefaultValue>clocked</DefaultValue>
        </Attribute>
        <Attribute Name="DelayMode" AttributeDataType="xs:string"
ChangeMode="create">
        <Description>Specified or determined randomly
within a selectable time range</Description>
        <DefaultValue>fix</DefaultValue>
        </Attribute>
        <Attribute Name="CycleTime" AttributeDataType="xs:integer"
Unit="s" ChangeMode="create">
        <Description>Cycle time</Description>
        </Attribute>
        <Attribute Name="DelayTime" AttributeDataType="xs:integer"
Unit="s" ChangeMode="create">
        <Description>Delay time</Description>
        </Attribute>
        </RoleClass>
        <RoleClass Name="Sink"
RefBaseClassPath="MaterialsHandlingProcessClassLib/Source&Sink"
ChangeMode="create">

```

```
                <Description>Defines the sink of a material
flow</Description>
    </RoleClass>
</RoleClass>
<RoleClass Name="GoodProcess"
RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Process
" ChangeMode="create">
    <Description>A general class to define good
processes</Description>
    <Attribute Name="ProcessTime"
AttributeDataType="xs:nonNegativeInteger" Unit="sec" ChangeMode="create">
        <DefaultValue>0</DefaultValue>
    </Attribute>
    <RoleClass Name="Loading"
RefBaseClassPath="MaterialsHandlingProcessClassLib/GoodProcess"
ChangeMode="create">
        <Description>Defines a good loading process</Description>
    </RoleClass>
    <RoleClass Name="Discharging"
RefBaseClassPath="MaterialsHandlingProcessClassLib/GoodProcess"
ChangeMode="create">
        <Description>Defines a good discharging
process</Description>
    </RoleClass>
    </RoleClass>
    </RoleClassLib>
</CAEXFile>
```