TU WIEN Informatics

Validierung von AutomationML Modellen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Martin Glatz, BSc
Matrikelnummer 00525211

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.-Prof. Mag. Dr. Manuel Wimmer
Zweitbetreuung: O.Univ.-Prof. DI Mag. Dr. Gerti Kappel

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Wien, 31. Jänner 2023

Martin Glatz	Manuel Wimmer

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

TU WIEN Informatics

Validation of AutomationML Models

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business informatiks

by

Martin Glatz, BSc
Registration Number 00525211

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.-Prof. Mag. Dr. Manuel Wimmer
Second advisor: O.Univ.-Prof. DI Mag. Dr. Gerti Kappel

Vienna, 31st January, 2023

Martin Glatz   Manuel Wimmer

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Martin Glatz, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 31. Jänner 2023

Martin Glatz

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

An dieser Stelle bedanke ich mich bei all denjenigen, die mich beim Schreiben dieser Diplomarbeit unterstützt und begleitet haben.

Insbesondere möchte ich mich bei meinem Betreuer Univ.-Prof. Mag. Dr. Manuel Wimmer für die Begutachtung der Diplomarbeit bedanken sowie für seine hilfreichen Anregungen, seinen fachlichen Input und seine anhaltende Unterstützung.

Mein besonderer Dank gilt Frau O.Univ.-Prof. DI Mag. Dr. Gerti Kappel, die mich stets motivierte die Diplomarbeit neben meinen beruflichen und privaten Herausforderungen im letzten Jahr abzuschließen. Ihre motivierende und nachhaltige Unterstützung hat maßgeblich dazu beigetragen, dass die Diplomarbeit nun in dieser Form vorliegt.

Ebenso gilt mein Dank Lara für das Korrekturlesen dieser Diplomarbeit und ihre stets motivierenden Anregungen.

Abschließend möchte ich mich bei meiner Verlobten Angelika bedanken, die mir mit viel Geduld, ihrem Rückhalt und Hilfsbereitschaft während dem Schreiben dieser Diplomarbeit stets zur Seite stand.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Die Planung, Umsetzung und Inbetriebnahme von industriellen Produktionsanlagen ist aufgrund der Vielzahl von involvierten Industriezweigen (z.B. Elektrotechnik, Maschinenbau, etc.) ein komplexes Unterfangen mit vielen wechselseitigen Abhängigkeiten. Aufgrund der branchenspezifisch sehr individuellen Softwareanforderungen kommt in den involvierten Industriezweigen hochspezialisierte Software zum Einsatz. Dies hat aktuell zur Folge, dass ein toolübergreifender Dateiaustausch aufgrund proprietärer Dateiformate nicht ohne manuellen Zusatzaufwand durchführbar ist. Darüber hinaus besteht das Risiko, dass Daten falsch interpretiert oder verloren werden.
Dieser Umstand wurde von führenden Unternehmen im Bereich der Automatisierung erkannt und als Lösung des Problems das allgemein verwendbare Datenaustauschformat *AutomationML* entwickelt. *AutomationML* setzt auf dem bereits existierenden Datenformat Computer Aided Engineering eXchange (*CAEX*) auf und definiert hinsichlich des Ausgangsformats diverse Erweiterungen und Einschränkungen.
Um einen effizienten und fehlerfreien Datenaustausch zu gewährleisten, muss sichergestellt sein, dass die Daten dem zugrundeliegenden Datenformat entsprechen. Bislang gibt es für *AutomationML* Modelle keine Möglichkeit ihre Konformität hinsichlich der über *CAEX* hinausgehend definierten Regeln zu prüfen.
Diese Diplomarbeit greift dieses Problem auf und implementiert mithilfe von Tools aus dem Bereich der Modellbasierten Software Entwicklung ein Validierungsframework für *AutomationML* Modelle. Die bereits existierende Validerung hinsichtlich Konformität zum *CAEX* Datenformat wird als Ausgangsbasis verwendet. Darüber hinaus werden die seitens *AML* Spezifikation definierten Erweiterungen und Einschränkungen im Rahmen der Umsetzungen formalisiert.
Das implementierte Validierungsframework untersucht *AutomationML* Modelle in einem strukturierten Prozess auf Konformität hinsicht des Dateiformats und informiert über etwaige Abweichungen.
Zur Evaluierung wurden Testmodelle erstellt, die bewusste Abweichungen zur *AutomationML* Spezifikation aufweisen. Auf Basis derer wurde das korrekte Auffinden von Abweichungen überprüft. Darüber hinaus wurden der *AutomationML* Spezifikation entsprechende Modelle erstellt, um zu gewährleisten, dass korrekte Modelle auch als solche erkannt werden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

The planning, implementation and commissioning of industrial production plants is a complex endeavour which requires the collaboration of a variety of engineering disciplines (e.g. electrical engineering, mechanical engineering, etc.). Due to industry specific software requirements, highly specialized software is used in the involved branches. This results in proprietary data formats without possibility of data exchange between different engineering disciplines. As this is a time consuming and error prone task with the risk of misinterpretation or lost data this issue was identified as reason for delayed projects.
Therefore leading companies in the field of automation triggered the creation of *AutomationML* as a solution to the problem. It uses the hierarchical, neutral, XML based data format Computer Aided Engineering eXchange (*CAEX*) as its top-level format and adds additional restrictions and extensions.
In order to ensure efficient and error-free data exchange, it must be ensured that the data conforms to the underlying data format. So far there has been no way for *AutomationML* models to validate their conformity with respect to restrictions and extensions invented by the *AML* specification.
This thesis implements a validation framework for *AutomationML* models using tools from the field of model-based software development. The already existing validation regarding conformity to the *CAEX* data format is used as basis. In addition, the extensions and restrictions defined by the *AML* specification are formalized as part of the implementation.
The implemented validation framework examines *AutomationML* models in a structured process for conformity with regard to the *AML* specification and reports deviations.
For evaluation purposes test models which deliberately deviate from the *AutomationML* specification were created. Based on these faulty models the correct detection of deviations is checked. In addition *AML* models conforming to the *AutomationML* were created to ensure that correct models are recognized as such.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung **ix**

Abstract **xi**

Contents **xiii**

1 Introduction **1**
1.1 Motivation 1
1.2 Problem Statement 2
1.3 Aim of this Thesis 3
1.4 Structure of this Work 3

2 AutomationML at a Glance **5**
2.1 Motivation for the Invention of AutomationML 5
2.2 Requirements and Solution Approach 7
2.3 AutomationML Architecture Specification 8
2.4 *AML* Base Libraries 22

3 Related Work **35**
3.1 Validation Approaches 35
3.2 Standardization Approaches 37

4 Used Methods and Techniques **43**
4.1 Model-Driven Software Engineering Basics 43
4.2 Modelling Languages 44
4.3 Eclipse Modelling Framework 45
4.4 Model Validation 48

5 Validation Framework **51**
5.1 *CAEX* Validation Phase 51
5.2 *AML* Validation Phase 51
5.3 Software Metrics of the Validation Framework 65

6 Evaluation **67**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6.1 General Evaluation Approach 67
6.2 Evaluation of *AML* Document Version Validation 68
6.3 Evaluation of *AML* Object Identification Validation 68
6.4 Evaluation of *AML* Path Validation 71
6.5 Evaluation of *AML* Class Requirements Validation 73
6.6 Evaluation of *AML* Instance Requirements Validation 74
6.7 Evaluation of *AML* Meta information Validation 76
6.8 Evaluation Result 77

7 Conclusion and Future work **79**
7.1 Conclusion 79
7.2 Future Work 80

A Appendix **81**
A.1 AML Base Libraries 81

List of Figures **85**

List of Tables **87**

Listings **87**

Bibliography **89**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 1

Introduction

1.1 Motivation

While the market demands continuously shorter development cycles for industrial production facilities, the planning and construction is getting more and more complex [14, 19]. One reason for the raised complexity is the tendency towards distributed and highly sophisticated automation systems used in industrial plants [19].

The automation system's implementation requires the collaboration of a variety of engineering disciplines e.g., system design, mechanical engineering, electrical engineering etc. [8, 18]. Besides the involvement of a lot of concerned parties, the plant engineering process has a strong project phase separation as depicted in Figure 1.1 [34, 17]. These issues raise the complexity of data exchange between project phases and the involved project teams.

While the need for a sophisticated tool support within the individual project phases has been recognized and treated long time ago, an efficient tool support between project phases and engineering disciplines is still not available [14, 18]. The used software tools are often heterogeneous and the situation can be described as a chain of isolated tool islands [19, 17]. Nevertheless, an efficient method to avoid media disruption between engineering disciplines and project phases is vital to avoid data inconsistencies, project delay and therefore increased time to market [19].

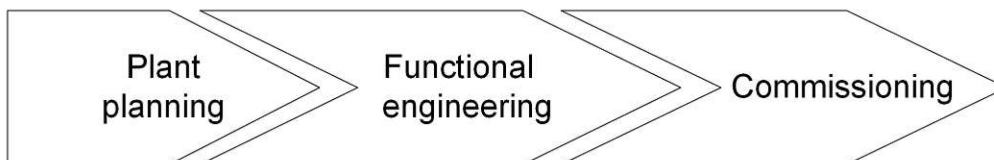

Figure 1.1: Simplified plant engineering process [34]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

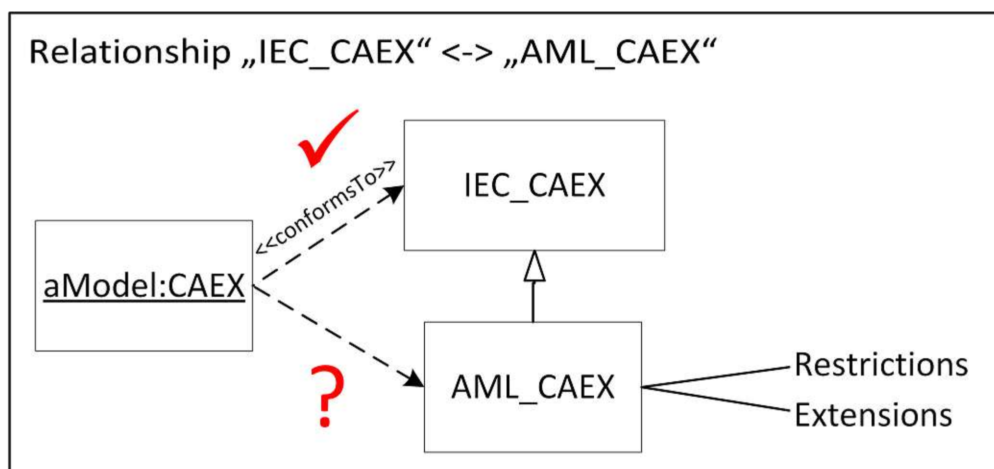

Figure 1.2: Standard IEC CAEX vs AML CAEX

To address this media disruption problem, Daimler initiated the development of the data-exchange language *AutomationML* (*AML*) in cooperation with the universities of Magdeburg and Karlsruhe[1] [18]. *AML* is a standardized, neutral, free to use, *XML* based data exchange format. It uses the hierarchical, neutral, XML based data format *Computer Aided Engineering eXchange* (*CAEX*), which was standardized by the *International Eletrotechnical Commission (IEC)*[2], as its top-level format. *CAEX* is enhanced with additional restrictions and extensions within the *AML* specification [18]. This thesis will furthermore refer to *CAEX* documents which also adhere to the additional restrictions and extensions invented by the *AML* specification as *AML CAEX*.

It has been recognized that the adherence to the *AML* specification is vital for an error-free and unambiguous data-exchange [19]. Any deviation from the formal specification would compromise the target of an indisputable data-exchange between software tools. In addition the explicit expressiveness during plant-modelling is impaired by mistakable data-exchange throughout the various project phases. However, no formalisation of restrictions and extensions introduced by the *AML CAEX* specification exists [32].

The intention of this thesis is to formalize the restrictions and extensions defined by *AML CAEX* and provide a method for data validation of *AML* models.

1.2 Problem Statement

As *AML* utilizes *CAEX* as basis but refines its usage within the *AML* specification, validity checks available for standard *CAEX are not sufficient* for *AML CAEX* structures. This means a valid *AML CAEX* structure is a valid standard *CAEX* structure. Due

[1] TU Wien is collaborating as academic member.
[2] IEC 62424 standard (`https://webstore.iec.ch/publication/7000`)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

to the restrictions and extensions in the *AML* specification, a valid standard *CAEX* structure is not necessarily a valid *AML CAEX* structure, as depicted in Figure 1.2.

Existing solutions only test *AML* documents based on their standard *CAEX* conformity, but ignore the *AML CAEX* specialities. Based on this shortcoming the validation of *AML CAEX* is not in line with its formal specification which makes inconsistencies in *AML* documents hard to detect and potentially increases debugging time.

Therefore the problem-statement can be summarized as the lack of a formalisation of restrictions and extensions introduced by the *AML CAEX* specification. In addition, no automated, and easy to use method to validate *AML CAEX* conformity based on its specification is available.

1.3 Aim of this Thesis

This thesis proposes the creation of a validation suite for *CAEX AML*. Based on the meta-model of *CAEX* a validation suite utilizing Model-Driven Software Engineering is implemented. The various constraints and extensions introduced by *AML* in comparison to *CAEX* are formalized in the *Object Constraint Language* (*OCL*) [29] dialect *Epsilon Validation Language* (*EVL*) [3]. Based on this formalization a validation process consisting of sequential validation steps is established. For evaluation purposes a validation framework based on JUnit tests is created which provides at least one JUnit test for each formalized *AML* concept. This ensures model consistency and is a cornerstone towards an error free and unambiguous data exchange in plant construction projects.

1.4 Structure of this Work

Chapter 2 describes the motivation to create *AML* and explains the requirements and the selected solution approach. Based on that, the basic *AML* concepts are introduced.

Chapter 3 presents existing approaches in the field of *AML* validation. Beyond that advances in the field of *AML* and possibilities to standardize semantics as well as modelling techniques are introduced.

Chapter 4 introduces the basic principles of Model-Driven Software Engineering and presents different modelling languages. Beyond that, it presents the Eclipse Modelling Framework, which was used for the implementation of this thesis. In addition it introduces modelling languages *OCL* and *EVL*, which was utilized for the evaluation of *AML* models.

Chapter 5 describes the process the validation of *AML* models is based on. In addition it gives an in depth insight how the implementation is structured and what model artifacts are validated.

Chapter 6 describes the approach which was implemented to verify that the implementation is working properly. It discusses the general approach and shows based on multiple categories which and how many test models were utilized for the evaluation process.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Chapter 7 sums up the findings of this thesis and gives a lookout what are possible next steps in the field of *AML* model validation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 2

AutomationML at a Glance

This chapter describes the problems identified during construction of production facilities and how *AutomationML* intends to solve these issues. It presents the main concepts of *AutomationML* and highlights the additions and restrictions invented by *AutomationML* in comparison to its base format *CAEX*.

2.1 Motivation for the Invention of AutomationML

The construction of production facilities is a complex endeavour with lots of competitors and low margins. Therefore it is required that projects are executed as efficient as possible [6]. Anyhow as depicted in Figure 2.1 projects often finish late which leads to increased cost and postponed start of production.

Studies showed, that one reason for delay, is an inefficient data exchange between project participants during the project's life span [6, 14]. The overall plant construction is typi-

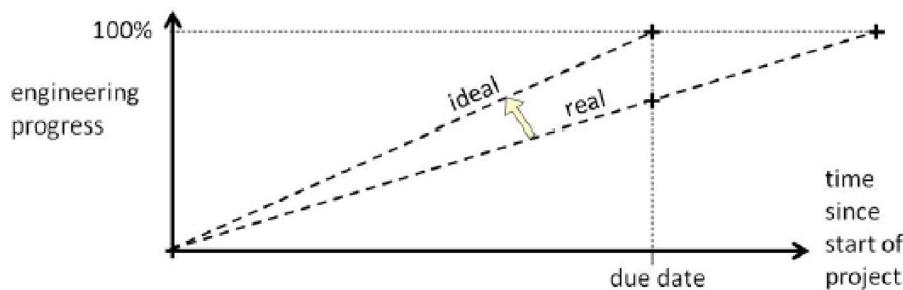

Figure 2.1: Planned vs real project progression [6]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

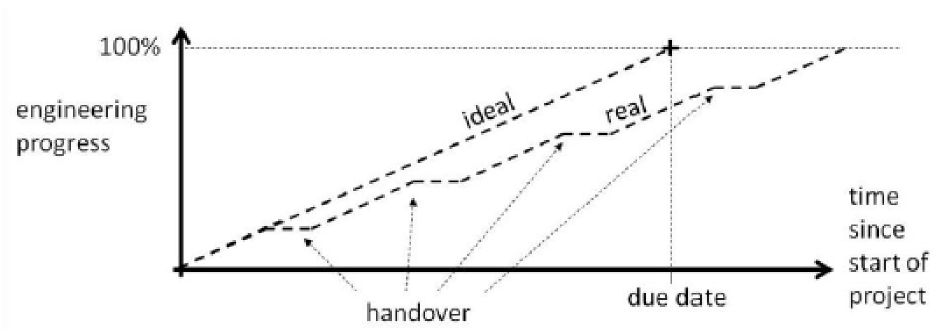

Figure 2.2: Planned vs real project progression with handover efforts [6]

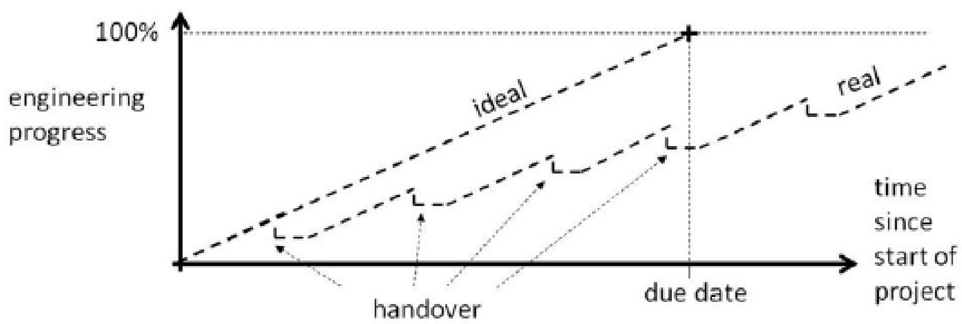

Figure 2.3: Planned vs real project progression with data loss[6]

cally separated in sequential project phases e.g. plant planning, functional engineering and commissioning and involves many different engineering disciplines [6, 14]. This requires to frequently exchange data between involved parties [6, 14, 24]. As every involved engineering discipline typically utilizes tools which fit their requirements best, the tool landscape is very heterogeneous [6, 34]. This results in the so called "paper-interface" [14]. It describes the inability of other project participants to incorporate existing artefacts into their tool landscape and brings delays as depicted in Figure 2.2. One reason for this delay is the need to analyse and import artefacts from other project participants into the own tool landscape [6].

Beyond that, it is also possible that engineering details are lost during handover which results in a timeline as depicted in Figure 2.3. Potential reasons for that are difficulties in understanding artefacts due to missing common semantics or non existing interfaces to lossless import data [6].

The mentioned problems were recognized by industry and the following Section describes the basic ideas during the design of *AutomationML* to make the data exchange more efficient.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.2 Requirements and Solution Approach

To address the problems mentioned in Section 2.1, Daimler initiated the development of a data-exchange language in cooperation with the universities of Magdeburg and Karlsruhe [18].

During language definition, the following main requirements were defined [14]:

- **Open and freely available:** It shall be available for industry and research without licence fees
- **Enable data exchange:** It shall enable data exchange between project phases and engineering disciplines
- **High applicability:** It shall enable the modeling of artefacts independent of their complexity and be applicable from plant planning to commissioning
- **High market usage:** It shall attract market leader to integrate the standard into their leading products
- **Usage of existing formats:** It shall utilize existing industry formats
- **Extensible and standardization:** Is shall be extendable and standardized

The result was *AutomationML* (*AML*), a standardized, neutral, free to use, *XML* based data exchange format which enables standardisation of syntax and semantic level [24, 34, 14]. *AML* is primarily based on the three existing data formats *Computer Aided Engineering eXchange* (*CAEX*)[1], *COLLAborative Design* (*COLLADA*)[2] and *PLCopen XML*[3][18, 19, 34]. *CAEX* is used as top level format and the connecting link between the involved data formats [24].

With these established data formats, *AML* is capable to express the following facets, necessary for the planning, engineering and commissioning of industrial plants [19, 34]:

- Topology information (*CAEX*)
- Geometry and kinematics information (*COLLADA*)
- Logic information (*PLCopen XML*)

As this thesis focuses on the modelling of plant topologies, further references to *AML* are only related to the *CAEX* based part of *AML*.

The following Section gives a brief overview of the main concepts of *AML* with focus on the additions and restrictions invented by *AML* in comparison to *CAEX*.

[1]IEC 62424 standard (https://webstore.iec.ch/publication/7000)
[2]ISO/PAS 17506:2012 standard (http://www.iso.org/iso/catalogue_detail.htm?csnumber=59902)
[3]IEC 61131-3 standard (https://webstore.iec.ch/publication/4552)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

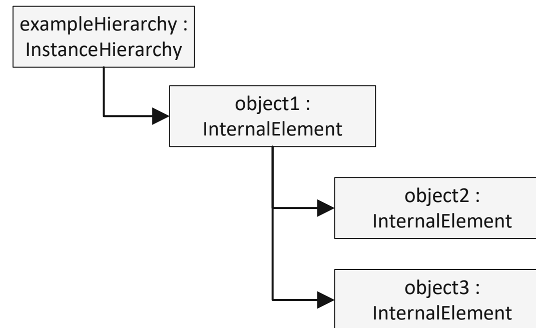

Figure 2.4: Pure **IH**

2.3 AutomationML Architecture Specification

This section describes the library concept introduced by *AML*. These libraries build the foundation for modelling hierarchies, semantics, interfaces and components. Furthermore the concept of paths for object and class references is introduced.
As *AML* uses an object hierarchy, this section describes possible relations which can be used for modelling with *AML*. Furthermore concepts for document versioning, storing the model's meta data and object identification are introduced.

2.3.1 *AML* Library concept

To fulfill the requirements defined in Section 2.2, *AML* consists of four modelling means [24]:

- Definition of hierarchies
- Definition of semantics
- Definition of relations
- Definition of libraries

The main target of *AML* is to enable the modelling of concrete plant topologies. For that purpose it utilizes *InternalElements* (**IE**s), which represent concrete artefacts of the plant. These **IE**s follow the object oriented paradigm and are hierarchically arranged within *InstanceHierarchies* (**IH**s).
Figure 2.4 depicts an **IH** that describes a model with an artefact *Object1*, which consists of two child elements *Object2* and *Object3*. This **IH** only describes the hierarchical arrangement of objects, without defining semantics or relations of its elements.

To add semantics, *AML* enables the definition of *RoleClasses* (**RC**s), which are grouped in *RoleClassLibraries* (**RClib**s). A **RC** defines the abstract functionality of an object without specifying the concrete technical implementation [34, 14]. Figure 2.5 depicts a **RClib** which defines the **RC**s *Robot* and *Station*.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

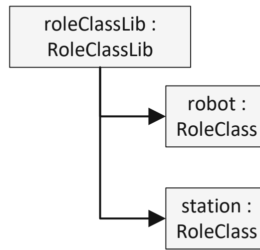

Figure 2.5: Pure **RClib**

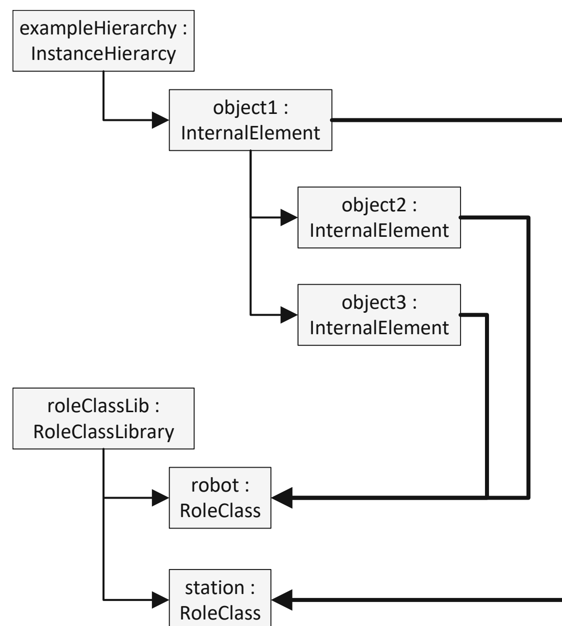

Figure 2.6: **IH** with **RClib**

Based on the defined **RC**s it is possible to assign semantics to **IE**s as depicted in Figure 2.6. The enhanced model shows, that the artefact *Object1* represents a *Station*. Its child objects *Object2* and *Object3* represent two *Robots*.

As objects are typically highly interlinked, *AML* enables the definition of relations with *InterfaceClasses* (**IC**s). **IC**s are grouped in *InterfaceClassLibraries* (**IClib**s) and are used to define interfaces which are used to interlink **IE**s. Figure 2.7 depicts an **IClib** with one **IC** *RobotInterface*.

IC instances are modelled as *ExternalInterfaces* (**ExtI**s). In Figure 2.8 the model is

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

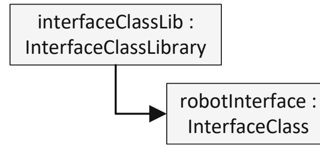

Figure 2.7: Pure **IClib**

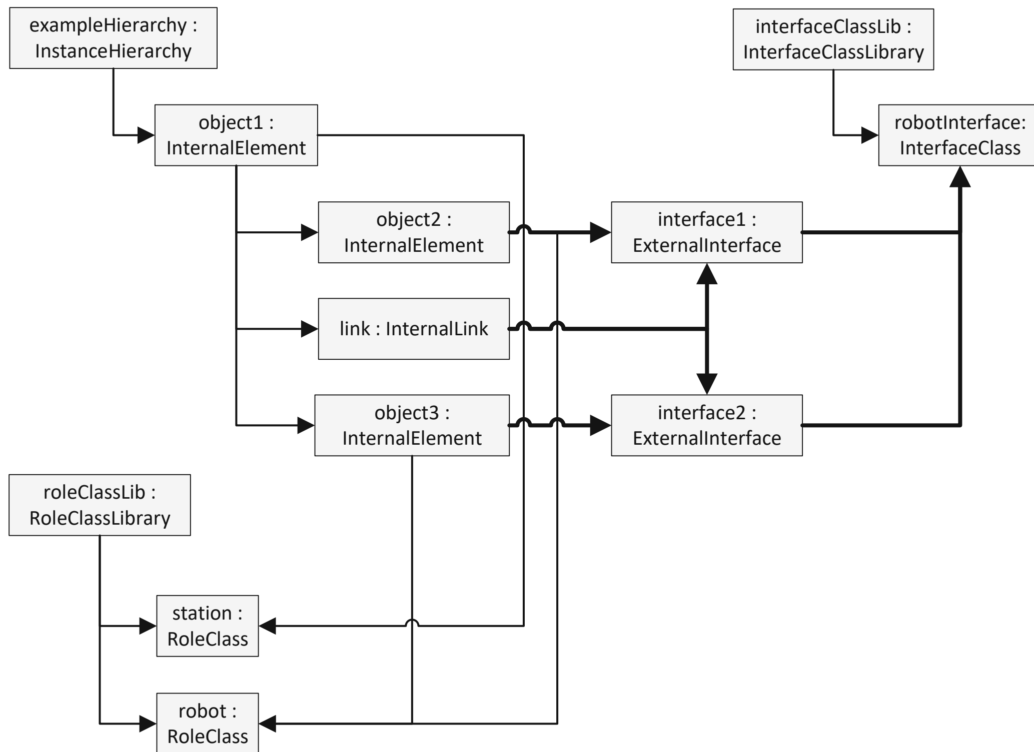

Figure 2.8: **IH** with **RClib** and **IClib**

enhanced with **IC** instances *Interface1* and *Interface2*. They are connected with an *InternalLink* (**IL**).

So far the **IH** contains **IE**s with semantic and linked interfaces, but the concrete implementation of the objects is still missing.
AML provides the possibility to model objects in *SystemUnitClasses* (**SUC**s) which are grouped in *SystemUnitClassLibraries* (**SUClib**s). With **SUC**s it is possible to model ready to use components comprising semantics, interfaces, etc. which can be directly added in an **IH**. This enables vendors to provide their full range of products (e.g., different

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

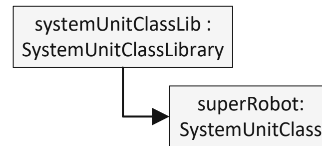

Figure 2.9: Pure **SUClib**

kinds of robots) as **SUC**s. Figure 2.9 depicts a **SUClib** containing one **SUC** *SuperRobot*. In Figure 2.10 the concrete technical implementation of *Object2* and *Object3* is added by assigning them the **SUC** *SuperRobot*.

To provide a possibility to integrate external models (e.g. **SUClib**s from vendors), *AML* allows to split the model into multiple *AML* files which can be referenced. Figure 2.11 depicts the model with an external **SUC**.

Based on this basic concepts the next sections will go into further details of *AML* and present restrictions and additions of *AML* in comparison to *CAEX*.

2.3.2 *AML* Document Versions

During data exchange, it is vital to identify the version of the *AML* model as well as the version of its containing libraries. Therefore, [24] defines the following artefacts as mandatory:

- *AML* version of the model
- Library version for **IClib**, **RClib** and **SUClib**

The model's *AML* version shall be stored in the attribute *AutomationMLVersion* of the element *AdditionalInformation*. Listing 2.1 shows an example of an *AML* document based on *AML* version 2.0.

```
1 <CAEXFile SchemaVersion="2.15"
2         FileName="Example.aml"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:noNamespaceSchemaLocation="ClassModel_V2.15.xsd">
5           <AdditionalInformation AutomationMLVersion="2.0"/>
6 </CAEXFile>
```

Listing 2.1: *AML* document versioning

If the *AML* model consists of multiple *AML* files as described in Section 2.3.1, every referenced model must conform to the same *AML* version. In addition, it is required that every library defines its version with the attribute *Version* as depicted in Listing 2.2. Multiple versions of the same library must not exist within the same *AML* file.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

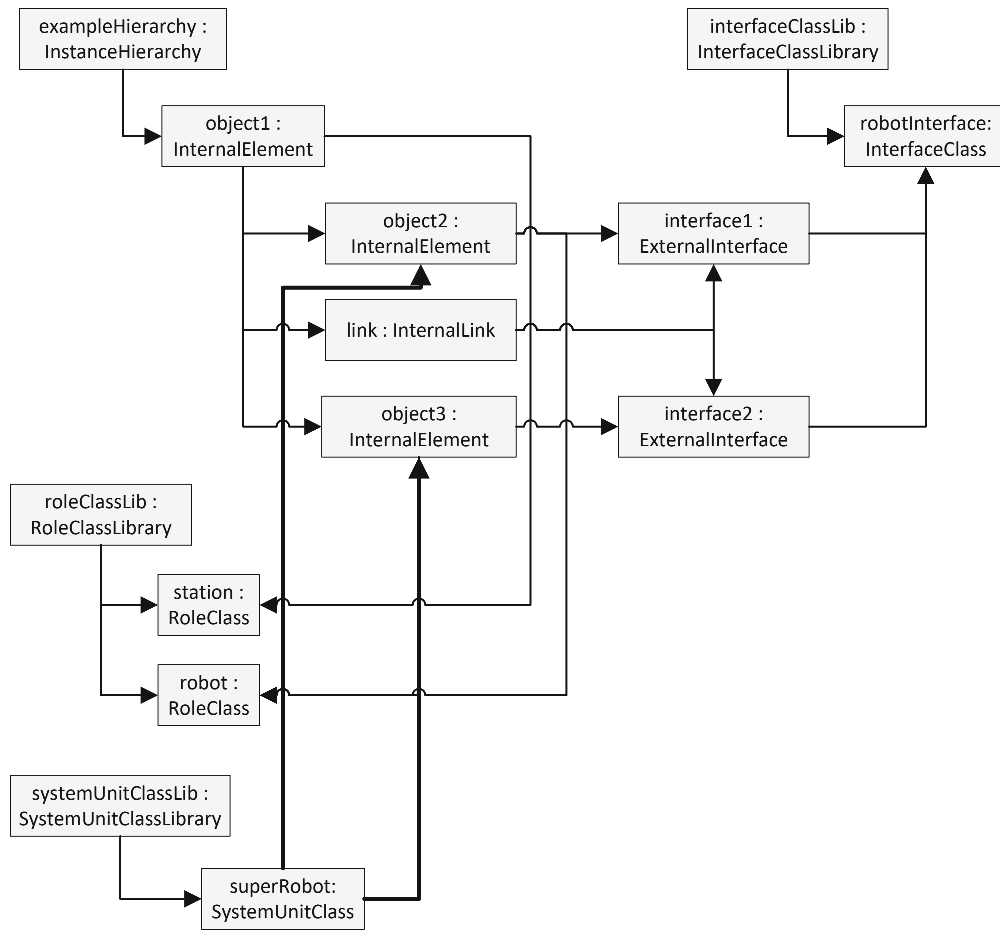

Figure 2.10: **IH** with **RClib**, **IClib** and **SUClib**

```
1 <InterfaceClassLib Name="AutomationMLInterfaceClassLib">
2     <Version>2.10.0</Version>
3 </InterfaceClassLib>
```

Listing 2.2: *AML* library versioning

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Figure 2.11: **IH** external **SUClib**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.3.3 *AML* Meta Information

Based on [24], it is required that every tool which updates an *AML* model, adds the following metadata to the model:

Element name	Type	Level
WriterName	xs:string	Mandatory
WriteID	xs:string	Mandatory
WriterVendor	xs:string	Mandatory
WriterVendorURL	xs:string	Mandatory
WriterVersion	xs:string	Mandatory
WriterRelease	xs:string	Mandatory
LastWritingDateTime	xs:DateTime	Mandatory
WriterProjectTitle	xs:string	Optional
WriterProjectID	xs:string	Optional

Table 2.1: AML metadata information [24]

The meta data information shall be stored in the element *AdditionalInformation* as child elements of the element *WriterHeader* as depicted in Listing 2.3. The order of the elements must conform to Table 2.1. If the *AML* model is updated from multiple tools, every tool shall store its meta information [24].

```
1  <AdditionalInformation>
2    <WriterHeader>
3      <WriterName>AutomationML Editor</WriterName>
4      <WriterID>916578CA-FE0D-474E-A4FC-9E1719892369</WriterID>
5      <WriterVendor>AutomationML e.V.</WriterVendor>
6      <WriterVendorURL>www.AutomationML.org</WriterVendorURL>
7      <WriterVersion>6.1.0.0</WriterVersion>
8      <WriterRelease>6.1.0.0</WriterRelease>
9      <LastWritingDateTime>2022-10-23T01:49:19.2168921</LastWritingDateTime>
10     <WriterProjectTitle>unspecified</WriterProjectTitle>
11     <WriterProjectID>unspecified</WriterProjectID>
12   </WriterHeader>
13  </AdditionalInformation>
```

Listing 2.3: *AML* meta information

2.3.4 *AML* Object Identification

AML uses the object oriented paradigm, therefore it is vital that objects can be identified unambiguously. Whereas in [23] the attribute *ID* for **`IE`**s and **`ExtI`** is considered optional, it is required to be a mandatory global unique identifier following [24].
An exemplary **`IE`** is depicted in Listing 2.4. Object identification is done based on the attribute ID, the value of the attribute *Name* has only informative character. The ID must not change during the lifetime of the instance.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
1 <InternalElement Name="Robot" ID="151d7f95-39f0-40a5-b305-9424608d620a"/>
```

Listing 2.4: *AML* object identification

It is vital to mention, that unique identification is not only important for **IE** but is also mandatory for libraries and classes. Whereas for class instances the attribute *ID* is used as identifier, for libraries and classes it is the attribute *Name*. For **IClib**, **RClib** and **SUClib** the name of the libraries must be unique within an *AML*.

Furthermore, it is required that class names are unique within their libraries. Listing 2.5 shows a **RClib** *RoleClassLib* which conforms to [24].

```
1 <RoleClassLib Name="RoleClassLib">
2     <Version>1.20</Version>
3     <RoleClass Name="UniqueRC1"/>
4     <RoleClass Name="UniqueRC2"/>
5     <RoleClass Name="UniqueRC3"/>
6 </RoleClassLib>
```

Listing 2.5: *AML* class identification

2.3.5 Paths in *AML*

AML provides a wide range of possibilities for references to classes, objects, attributes and interfaces, therefore it is vital to understand how this references are established with paths [24]. A path contains the required hierarchical node sequence to navigate from the root node to the desired target.

To distinguish between different object types, the following delimiter characters are defined by [23]:

- Object-separator: "/"
- Attribute-separator: "."
- Interface-separator: ":"
- Alias-separator: "@"

Considering the simplified *AML* model depicted in Figure 2.12, exemplary path navigation look as follows:

Target Element	Path
A	SUCL/Parent/Child
B	SUCL/Parent/Child.Attribute
C	GUID3::Interface1
D	Vendor@VendorSUCLib/VendorSUC

Table 2.2: AML path examples [24]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

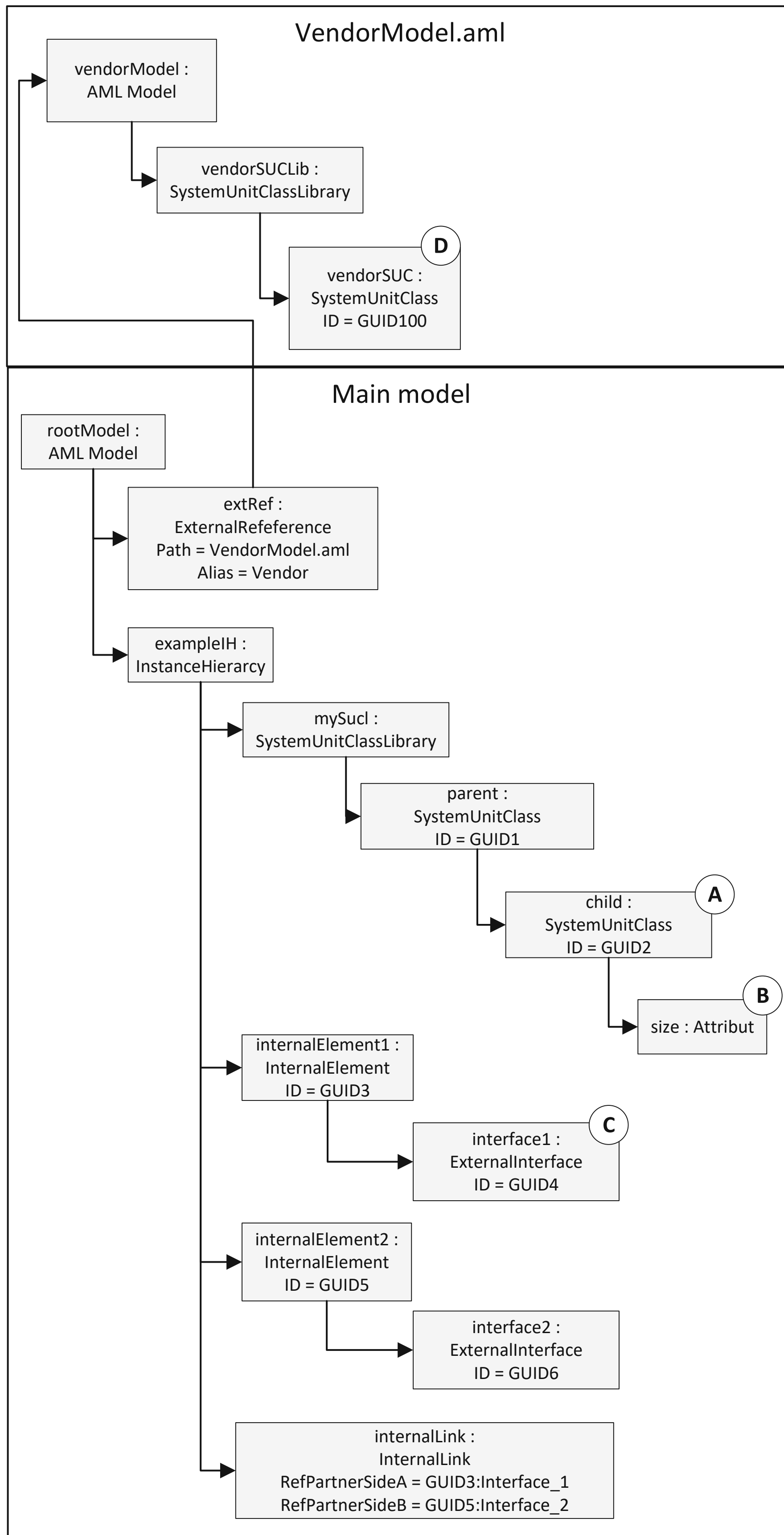

Figure 2.12: Paths

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

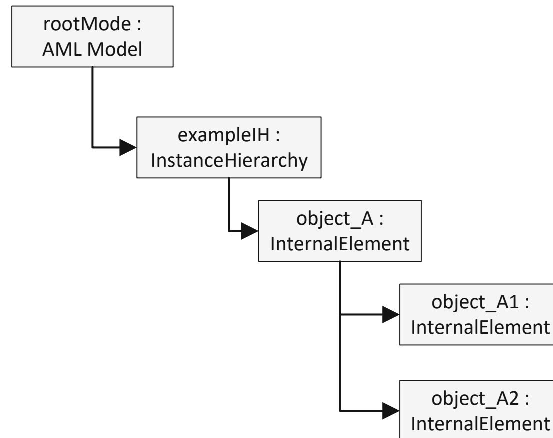

Figure 2.13: Object Parent-Child Relationship

2.3.6 *AML* Relations

As *AML* uses an object hierarchy to build up modelling artefacts, relations between elements is a very important modelling utility. In the following sections possible relations available in *AML* are described.

2.3.6.1 Parent-Child Relationships

Following [24], parent-child relationships are possible for *AML* objects as well as for *AML* classes.

A parent-child relationship between *AML* objects represents a *consists-of-relationship*. Figure 2.13 shows a typical example, which consists of an **IE** *Object_A* that contains the **IE**s *Object_A1* and *Object_A2*. Listing 2.6 depicts its *AML* representation.

```
1 <InternalElement Name="Object_A" ID="GUID1">
2     <InternalElement Name="Object_A1" ID="GUID2"/>
3     <InternalElement Name="Object_A2" ID="GUID3"/>
4 </InternalElement>
```

Listing 2.6: *AML* object parent-child relationship

Furthermore it is also possible to establish parent-child relationships between classes, which represent an inner class without inheritance relationship. Figure 2.14 depicts a **SUC** *Parent* which contains a **SUC** *Child*. Listing 2.7 shows its *AML* representation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

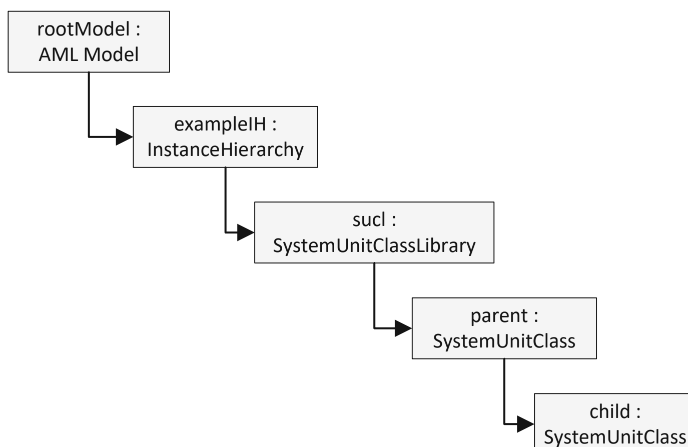

Figure 2.14: Class Parent-Child Relationship

```
1 <SystemUnitClass Name="Parent" ID="GUID1">
2       <SystemUnitClass Name="Child" ID="GUID2"/>
3 </SystemUnitClass>
```

Listing 2.7: *AML* class parent-child relationship

2.3.6.2 Inheritance

As *AML* utilizes an object-oriented approach, inheritance is a central concept. Inheritance is used to create a new class based on an existing base class with the intention to refine its implementation. The internal structure of the base class is inherited by the derived class [24, 14]. Figure 2.15 depicts an example inheritance, Listing 2.8 shows the *AML* model.

```
1 <SystemUnitClassLib Name="SUCL">
2     <SystemUnitClass Name="BaseClass" ID="GUID1">
3       <SystemUnitClass Name="DerivedClass" ID="GUID2" RefBaseClassPath="SUCL/BaseClass" />
4     </SystemUnitClass>
5 </SystemUnitClassLib>
```

Listing 2.8: AML Listing Inheritance

An inheritance relation is created by specifying the full path to the base class in the attribute *RefBaseClassPath*. In the concrete example the class *DerivedClass* is derived from the class *BaseClass*. If the base class is placed as in this example one hierarchy level above the child class, it is also allowed to only specify the name of the base class (e.g. *RefBaseClassPath=BaseClass*).
Inheritance is allowed for **SUC**s, **RC**s and **IC**s. It is important to mention that classes can only inherit from one class and only from classes of the same type (e.g. a **SUC** can only be derived from a **SUC**).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

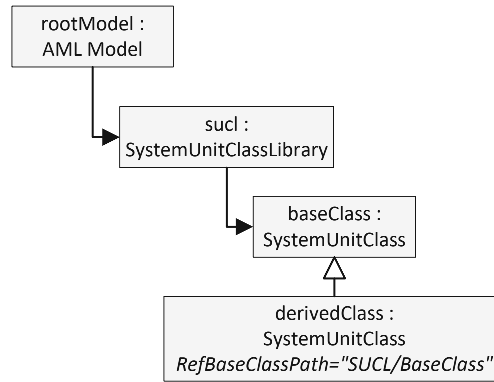

Figure 2.15: Inheritance

2.3.6.3 Class Instance Relationship

A class instance is modelled as **IE** and can be defined as part of an **IH** or a **SUC**. If an instance is instantiated based on a **SUC**, the internals of the **SUC** are copied to the **IE** and relationship between **IE** and **SUC** is specified by the attribute *RefBaseSystemUnitPath* [24]. This attribute contains the path to the originating **SUC** as described in Section 2.3.5.

In addition it is also possible to create an object based on another instance by using the "mirror concept" defined in *CAEX*. In this case, the attribute *RefBaseSystemUnitPath* contains the path of the mirrored object [24]. Figure 2.16 shows an example for a **SUC**-instance relation.

The relation of an instance and a **RC** is described in the instance's child-element *RoleRequirement* by the attribute *RefBaseRoleClassPath*. This attribute contains the main **RC** of the instance. As *AML* supports in comparison to *CAEX* multiple **RC**s for one instance, additional supported roles can be specified. This is done utilizing the child element *SupportedRoleClass* where additional **RC** can be defined via the attribute *RefRoleClassPath* [24]. Figure 2.17 shows an **IE** with a main **RC** *RoleClass1* also supporting the **RC**s *RoleClass2* and *RoleClass3*.

The relation between an **ExtI** and its **IC** is specified via the attribute *RefBaseClassPath*.

It is vital to mention, that although the class instances contain references to their originating classes a change in the source classes is **not** reflected to the instances. This is a restriction of *AML* in comparison to *CAEX*.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

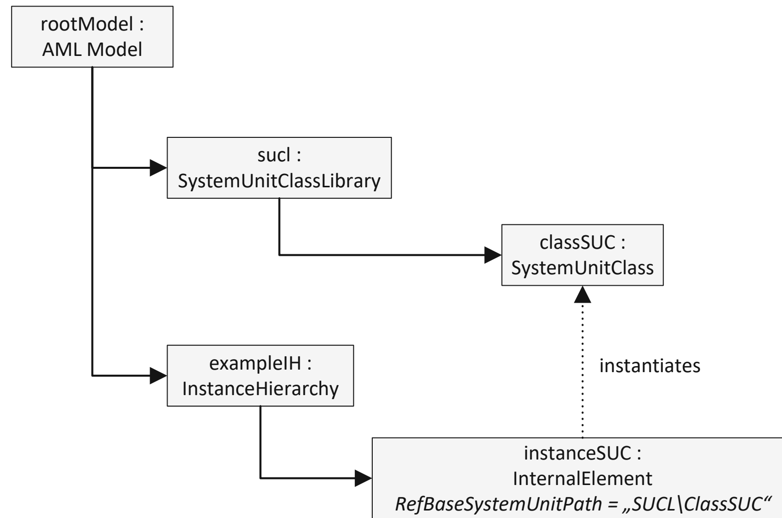

Figure 2.16: Class instantiation

2.3.6.4 Instance-Instance Relationship

To interlink **IE**s within an **IH** the **IE**s need to define **ExtI**s [24]. Based on these interfaces, it is possible to interlink them by defining an **IL**. Figure2.18 shows an exemplary instance-instance relation. The **IL** has as depicted in Listing 2.9 an attribute *Name* and contains in the attributes *RefPartnerSideA* and *RefPartnerSideB* the path of the interfaces which have to be connected.

```
<InstanceHierarchy Name="InstanceHierarchy">
  <InternalElement Name="IE_1" ID="GUID1">
  <ExternalInterface Name="Interface1" ID="GUID3" RefBaseClassPath="InterfaceClassLib/IC"/>
  <InternalLink Name="InternalLink"
    RefPartnerSideA="GUID1:Interface1" RefPartnerSideB="GUID2:Interface2"/>
  </InternalElement>
  <InternalElement Name="IE_2" ID="GUID2">
    <ExternalInterface Name="Interface2" ID="GUID3" RefBaseClassPath="InterfaceClassLib/IC"/>
  </InternalElement>
</InstanceHierarchy>
```

Listing 2.9: AML Listing Internal Links

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

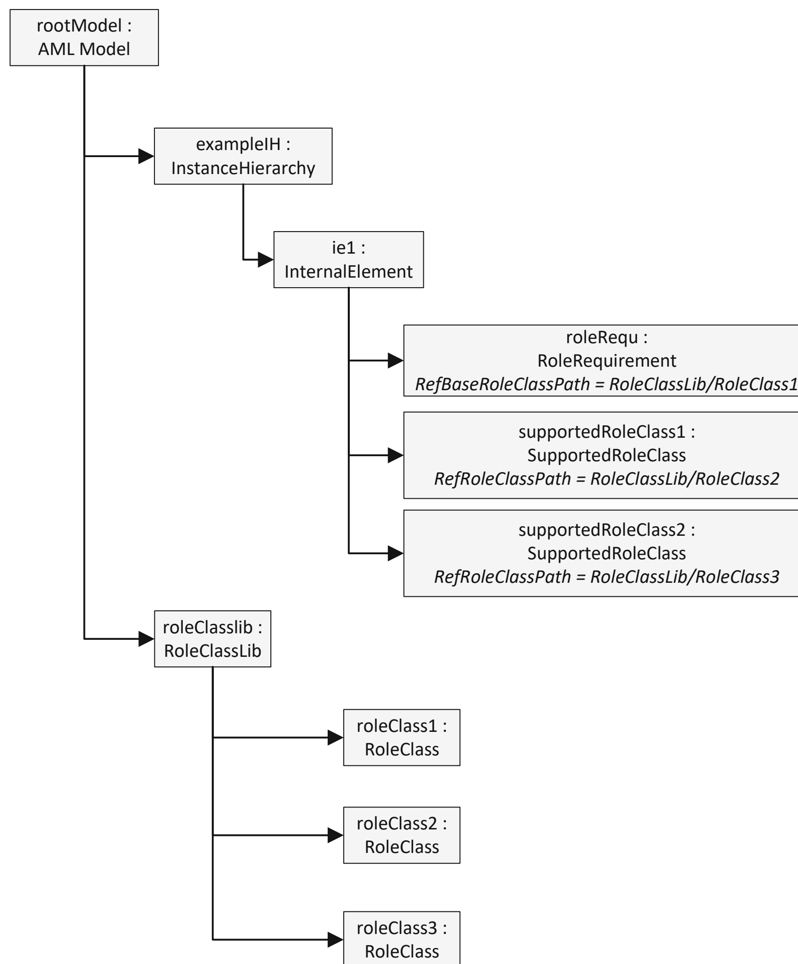

Figure 2.17: Multiple role classes

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

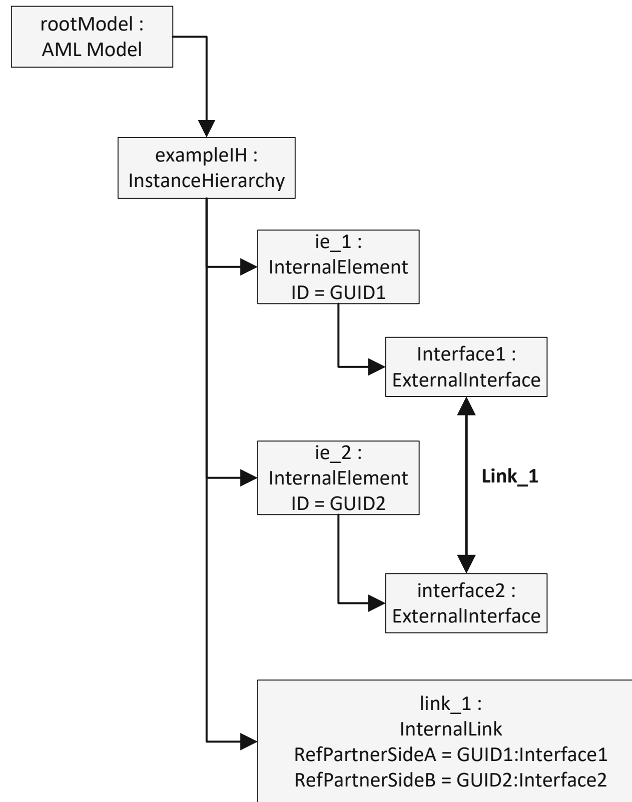

Figure 2.18: Class-Class Relation

2.4 *AML* Base Libraries

AML provides base class libraries for the modelling of **RC**s and **IC**s. User created **RC**s and **IC**s must be associated directly or indirectly to one of the provided base classes. This section presents the libraries defined in [24] and describes requirements introduced by them.

2.4.1 AML Role Class Library

In [24] the following *AML* base role classes are defined:

The *AML* representation of the *AML* **RC** base libraries can be found at Listing A.1.2. This section focuses on *AML* base role class, which define special requirements when instantiated.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

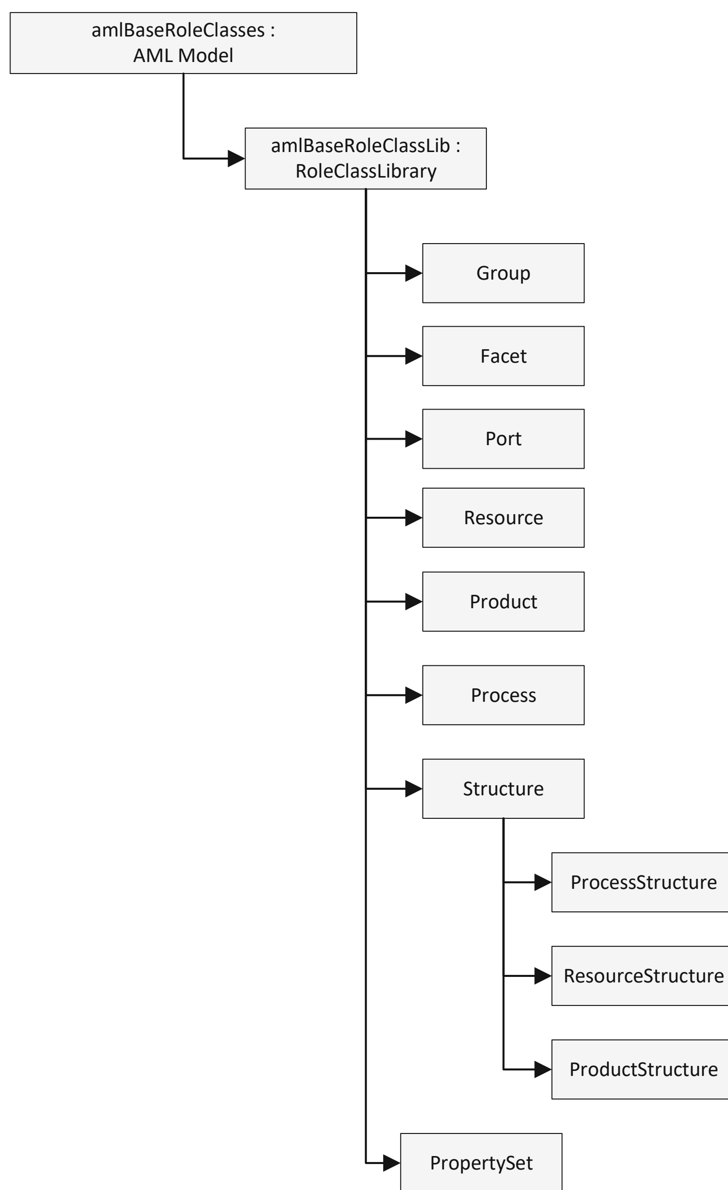

Figure 2.19: *AML* Base Role Classes [24]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.4.1.1 Role Class Port

The **RC** Port provides a possibility to simplify the connection of complex interfaces. It acts like a plug or socket and groups multiple interfaces that belong together. Instead of connecting each interface separately, Ports can be connected to other Ports based on the abstract grouping defined by them [24, 14].

Every Port instance contains an attribute *Direction* which defines whether the Port has direction “In”, “Out” or “InOut” . Ports with direction “In” can only be connected to Ports with direction “Out” or “InOut” whereas Ports with direction “Out” can be connected only to Ports with direction “In” or “InOut”. Ports with direction “InOut” can be connected to any kind of Ports.
Furthermore the attribute *Cardinality* defines the minimum and maximum number of Ports to be connected with the instance. In addition the attribute *Category* defines the type of the Port. Only Ports with the same category can be connected. The connection to other Ports is established with the **ExtI** *ConnectionPoint*.

Figure 2.20 shows an example containing two **IE**s which have two **ExtI**s each which are connected with a Port. Listing 2.10 shows the *AML* source of an exemplary Port instance.

```
1  <InternalElement Name="Port" ID="GUID1">
2      <Attribute Name="Direction" AttributeDataType="xs:string">
3        <Value>In</Value>
4      </Attribute>
5      <Attribute Name="Cardinality">
6        <Attribute Name="MinOccur" AttributeDataType="xs:unsignedInt">
7          <Value>0</Value>
8        </Attribute>
9        <Attribute Name="MaxOccur" AttributeDataType="xs:unsignedInt">
10         <Value>2</Value>
11       </Attribute>
12     </Attribute>
13     <Attribute Name="Category" AttributeDataType="xs:string">
14       <Value>MaterialFlow</Value>
15     </Attribute>
16     <ExternalInterface Name="ConnectionPoint" ID="GUID2" RefBaseClassPath="
           ↪AutomationMLInterfaceClassLib/AutomationMLBaseInterface/PortConnector" />
17     <RoleRequirements RefBaseRoleClassPath="AutomationMLBaseRoleClassLib/
           ↪AutomationMLBaseRole/Port" />
18 </InternalElement>
```

Listing 2.10: AML Listing Port

The following provision are defined in [24] for Ports:

- The Port instance must be a child element of the class or instance which should be connected
- The interfaces which need to be connected must be modelled as **ExtI**
- The Port instance must not contain child **IE**s
- The Port instance must have an **ExtI** which is derived from the *AML* base class *PortConnector*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

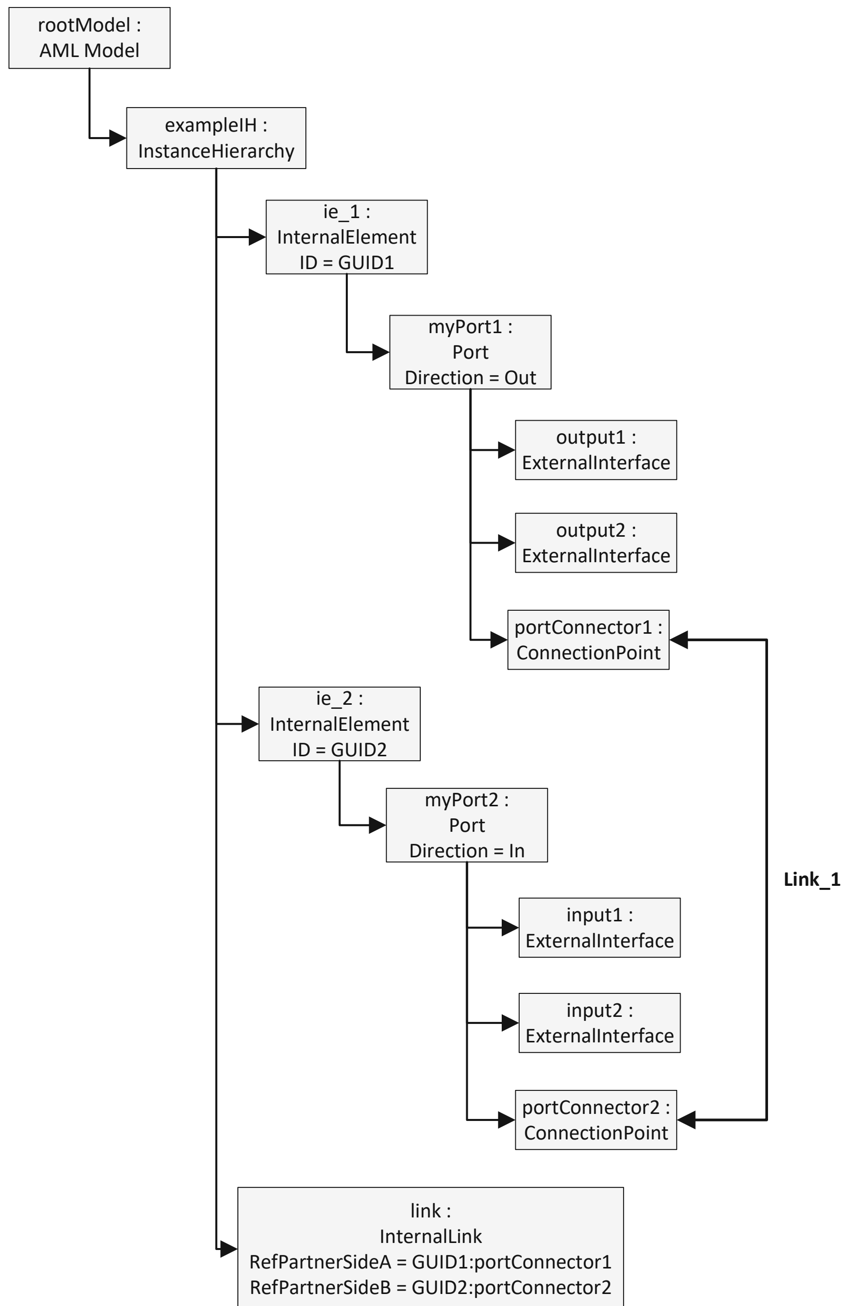

Figure 2.20: Connecting multiple interfaces with Port

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.4.1.2 Role Class Facet

The **RC** *Facet* provides a possibility to define sub-views on attributes and interfaces of *AML* objects [24, 14]. This enables the creation of engineering discipline specific views of instances. Figure 2.21 depicts an **IH** which contains an **IE** which has two attributes and two interfaces. One of every type is needed for electrical engineering, the other are related to mechanical engineering. The Facets *ElectricalFacet* and *MechanicalFacet* make this transparent and provide engineering specific views of the **IE**. Listing 2.11 shows the *AML* listing of the model.

```
1  <InstanceHierarchy Name="ExampleIH">
2    <Version>1</Version>
3    <InternalElement Name="IE_1" ID="GUID1">
4      <Attribute Name="ElectricalAttribute" AttributeDataType="xs:string" />
5      <Attribute Name="MechanicalAttribute" AttributeDataType="xs:string" />
6      <ExternalInterface Name="ElectricalInterface" ID="GUID2" />
7      <ExternalInterface Name="MechanicalInterface" ID="GUID3" />
8      <InternalElement Name="Electrical_Facet" ID="GUID4">
9        <Attribute Name="ElectricalAttribute" />
10       <ExternalInterface Name="ElectricalInterface" ID="GUID5" />
11       <RoleRequirements RefBaseRoleClassPath="AutomationMLBaseRoleClassLib/
           ↪AutomationMLBaseRole/Facet" />
12     </InternalElement>
13     <InternalElement Name="Mechanical_Facet" ID="GUID6">
14       <Attribute Name="MechanicalAttribute" />
15       <ExternalInterface Name="MechanicalInterface" ID="GUID7" />
16       <RoleRequirements RefBaseRoleClassPath="AutomationMLBaseRoleClassLib/
           ↪AutomationMLBaseRole/Facet" />
17     </InternalElement>
18   </InternalElement>
19 </InstanceHierarchy>
```

Listing 2.11: AML Listing Facet

The following provision are defined in [24] for Facets:

- The Facet instance must be a child element of the class or instance for which the view should be created
- The Facet's name must be unique among its siblings
- The Facet instance must only contain references to existing attributes and interfaces of the class or instance for which the view is created
- The Facet instance must not contain elements
- Facets must not be nested
- Facets must not modify existing attributes or interfaces of the class or instance for which the view is created

2.4.1.3 Role Class Group

The **RC** *Group* provides a possibility to define logical groups of elements independent of their concrete location in the **IH** [24, 14]. Figure 2.22 shows an exemplary **IH** which contains two **IE**s representing robots and two **IE** representing conveyors. With the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

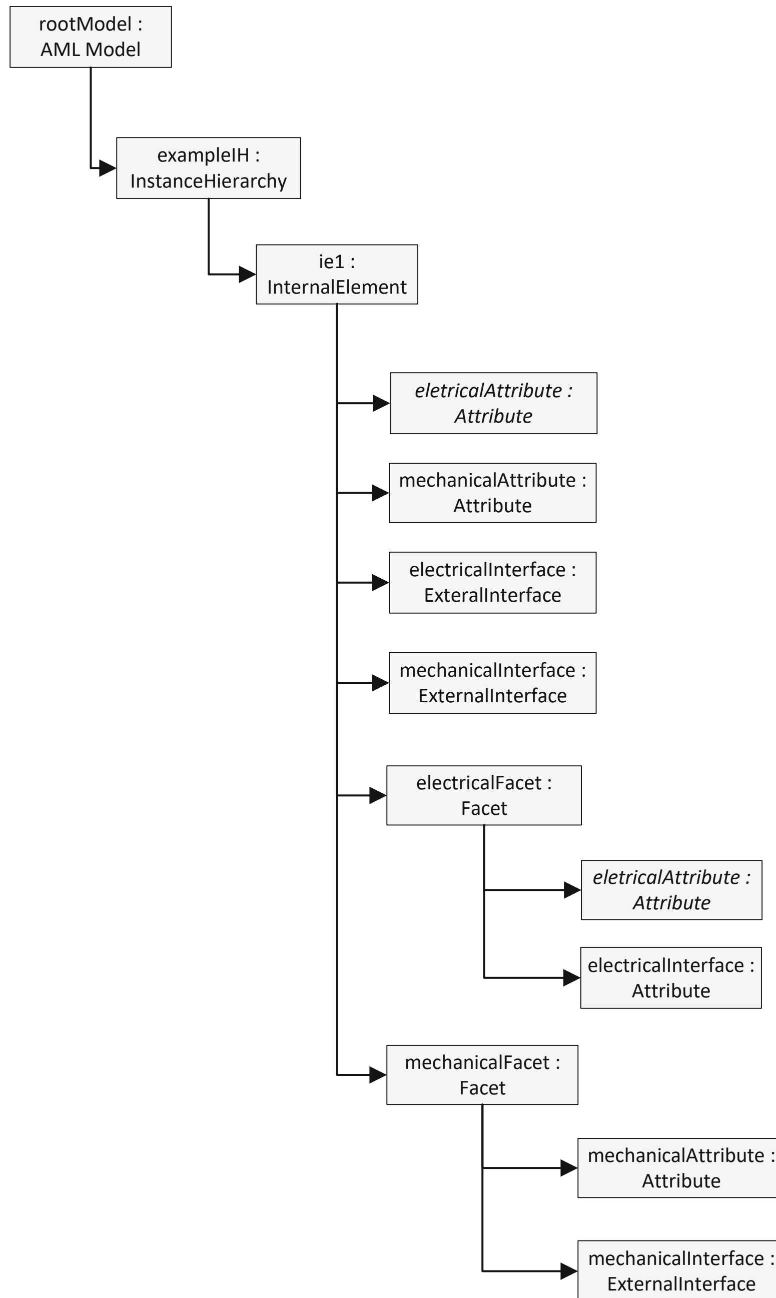

Figure 2.21: Facet Example

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

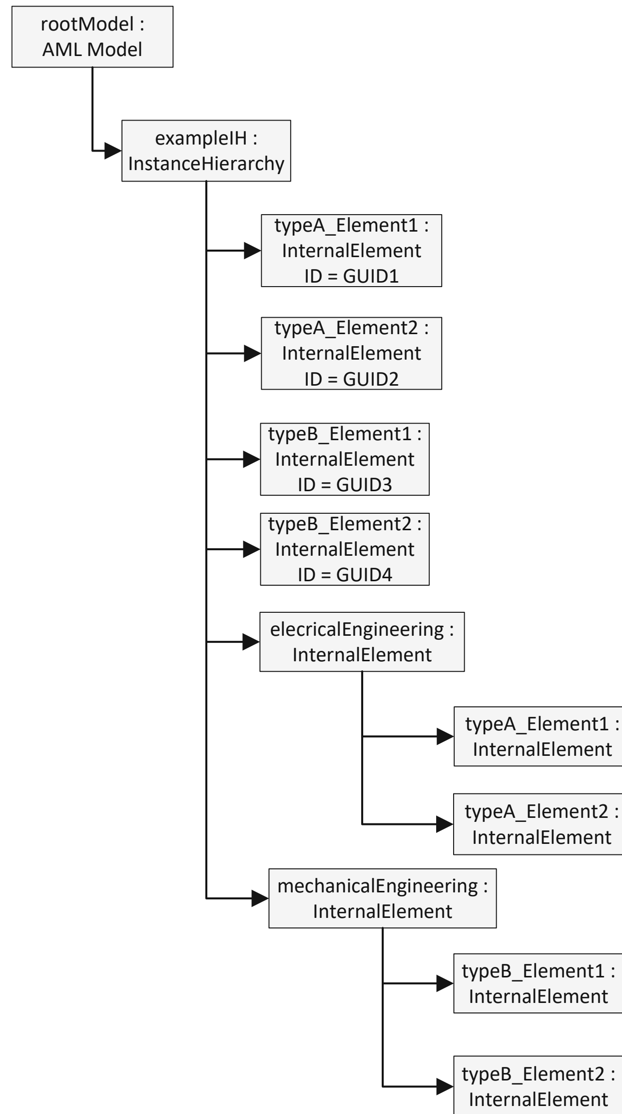

Figure 2.22: Group Example

Group **RC**, it is possible to provide two views *Robots* and *Conveyors* which define a logical grouping of instances of the same type. Listing 2.12 shows its *AML* representation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
1  <InstanceHierarchy Name="InstanceHierarchy">
2    <Version>0</Version>
3    <InternalElement Name="RobotA" ID="GUID1"/>
4    <InternalElement Name="RobotB" ID="GUID2"/>
5    <InternalElement Name="ConveyorA" ID="GUID3"/>
6    <InternalElement Name="ConveyorB" ID="GUID4"/>
7    <InternalElement Name="Robots" ID="GUID5">
8      <InternalElement Name="RobotA" ID = "GUID7" RefBaseSystemUnitPath="GUID1"/>
9      <InternalElement Name="RobotB" ID = "GUID8" RefBaseSystemUnitPath="GUID2"/>
10     <RoleRequirements RefBaseRoleClassPath="AutomationMLBaseRoleClassLib/
          ↪AutomationMLBaseRole/Group"/>
11   </InternalElement>
12   <InternalElement Name="Conveyors" ID="GUID6">
13     <InternalElement Name="ConveyorA" ID = "GUID9" RefBaseSystemUnitPath="GUID3" />
14     <InternalElement Name="ConveyorB" ID = "GUID10" RefBaseSystemUnitPath="GUID4" />
15     <RoleRequirements RefBaseRoleClassPath="AutomationMLBaseRoleClassLib/
          ↪AutomationMLBaseRole/Group" />
16   </InternalElement>
17  </InstanceHierarchy>
```

Listing 2.12: AML Listing Group

The following provisions are defined in [24] for Groups:

- A Group shall only contain mirror objects or further Group objects
- The mirrored object in the group must not be changed / enhanced in the Group
- The mirror object must have its own "ID"
- The attribute *AssociatedFacet* shall have - if used - a reference to an existing Facet

2.4.1.4 Role Class PropertySet

The **RC** *PropertySet* provides the possibility to generate a dictionary of predefined attributes for a specific domain. The intention is to ensure common understanding and standardization by definition of syntactically and semantically aligned attributes [24].
Figure 2.23 depicts an **IH** which contains an **IE** *Station*. A station has a minimum required space. In this example, this required space is described by the attributes *LengthStation* and *WidthStation*.
To provide a unified naming for areas, the model contains a **RC** *Area* which is derived from the **RC** PropertySet. The **RC** Area provides a general naming and defined the attributes *Length* and *Width*. Based on this standardized attribute namings, it is possible to create an element *MappingObject* which provides a mapping between LengthStation and Length and between WidthStation and Width. Listing 2.13 shows its *AML* representation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

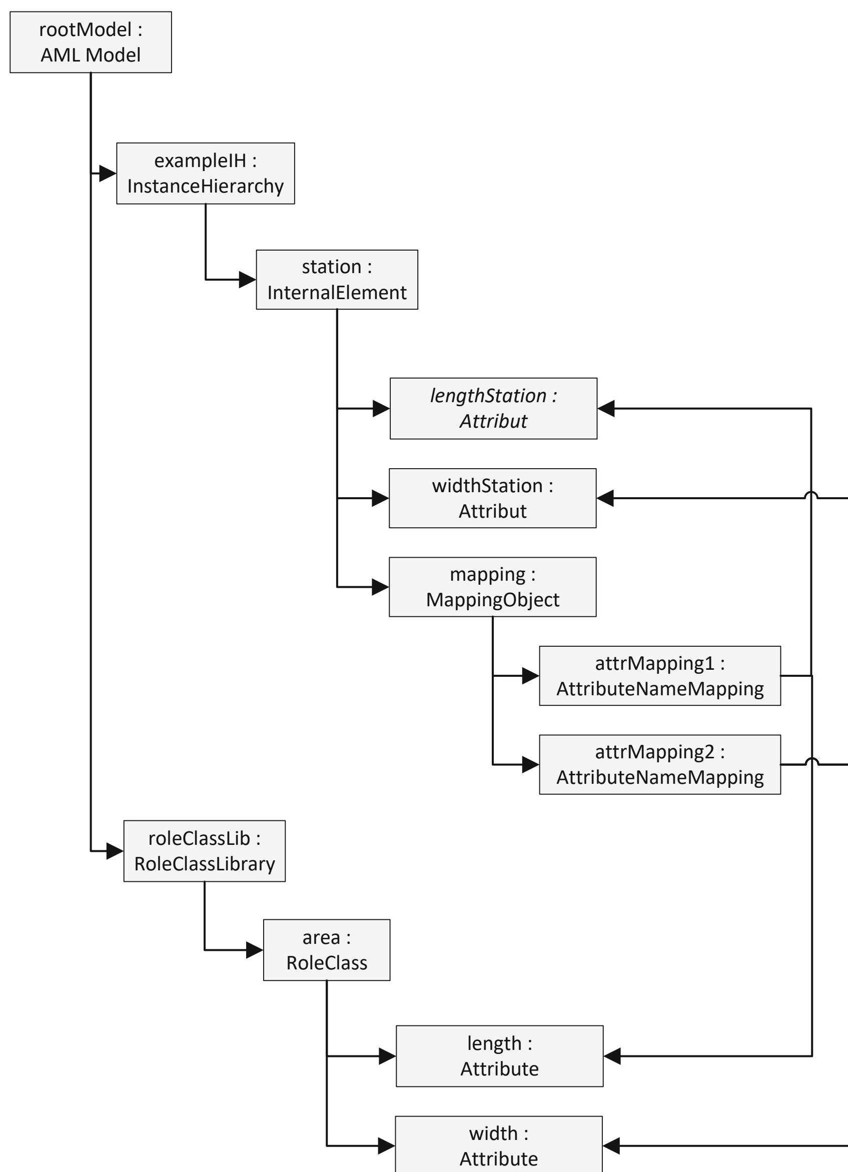

Figure 2.23: PropertySet Example

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
1  <InstanceHierarchy Name="InstanceHierarchy">
2    <InternalElement Name="Station" ID="GUID1">
3      <Attribute Name="LengthStation" AttributeDataType="xs:string" />
4      <Attribute Name="WidthStation" AttributeDataType="xs:string" />
5      <InternalElement Name="PropertySetMapping" ID="GUID2">
6        <RoleRequirements RefBaseRoleClassPath="RoleClassLib/Area" />
7        <MappingObject>
8          <AttributeNameMapping SystemUnitAttributeName="LengthStation" RoleAttributeName="
               ↪Length" />
9          <AttributeNameMapping SystemUnitAttributeName="WidthStation" RoleAttributeName="
               ↪Width" />
10       </MappingObject>
11     </InternalElement>
12   </InternalElement>
13 </InstanceHierarchy>
14 <RoleClassLib Name="RoleClassLib">
15   <Version>1</Version>
16   <RoleClass Name="Area" RefBaseClassPath="AutomationMLBaseRoleClassLib/
         ↪AutomationMLBaseRole/PropertySet">
17     <Attribute Name="Length" AttributeDataType="xs:string" />
18     <Attribute Name="Width" AttributeDataType="xs:string" />
19   </RoleClass>
20 </RoleClassLib>
```

Listing 2.13: AML Listing PropertySet

2.4.2 *AML* Interface Class Library

The *AML* representation of the *AML* **IClib** base libraries can be found at Listing A.1.1. This section focuses on *AML* base interface classes, which define special requirements when instantiated.

2.4.2.1 Order

The **IC** *Order* provides a possibility to describe interfaces dealing with orders. Instances of the **IC** Order must have an attribute *Direction*, which defines whether the interface has direction “In”, “Out” or “InOut”. Listing 2.14 shows an exemplary Order instance.

```
1 <InternalElement Name="InternalElement" ID="GUID1">
2       <ExternalInterface Name="Order" ID="GUID2" RefBaseClassPath="
            ↪AutomationMLInterfaceClassLib/AutomationMLBaseInterface/Order">
3         <Attribute Name="Direction" AttributeDataType="xs:string">
4           <Description>In</Description>
5         </Attribute>
6       </ExternalInterface>
7 </InternalElement>
```

Listing 2.14: Example instance Order

2.4.2.2 ExternalDataConnector

The **IC** *ExternalDataConnector* (**ExtCon**) acts as abstract base class for interfaces referencing external documents. It is the base class of the **IC**s *COLLADAInterface* and *PLCopenXMLInterface*. Instances of these **IC** must utilize the attribute *refURI* for storing

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

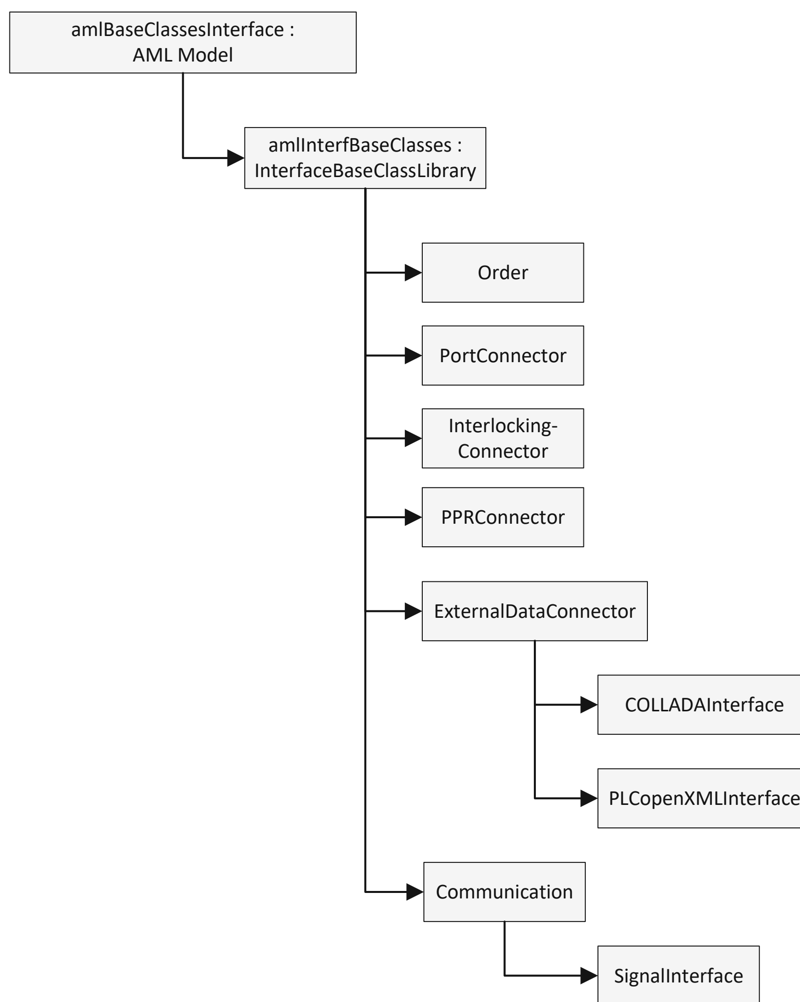

Figure 2.24: *AML* Base Interface Classes [24]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the path to the external document. Listing 2.15 shows an exemplary COLLADAInterface instance.

```
1 <InternalElement Name="InternalElement" ID="GUID1">
2       <ExternalInterface Name="COLLADAInterface" ID="GUID2" RefBaseClassPath="
          ↪AutomationMLInterfaceClassLib/AutomationMLBaseInterface/
          ↪ExternalDataConnector/COLLADAInterface">
3         <Attribute Name="refURI" AttributeDataType="xs:anyURI">
4           <Value>PathToExternalDoc</Value>
5         </Attribute>
6       </ExternalInterface>
7 </InternalElement>
```

Listing 2.15: Example instance COLLADAInterface

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Related Work

As *AML* intends to become the main standard for modelling in plant construction projects there are many topics which are addressed to extend and improve its applicability.
This chapter presents related work with focus *AML* model validation in Section 3.1. Further concepts for increasing standardization, recent developments of *AML* and modelling approaches are described in Section 3.2.

3.1 Validation Approaches

It's general consensus, that it's extremely important that *AML* models comply to their formal specification, as otherwise the main aim of *AML* to enable unambiguous data exchange would be obstructed. Therefore, multiple concepts to ensure it's validity are already available.

The work of Schleipen proposes a two step approach for validation of *AML* models [32]:

1. Syntax check
2. Semantic check

The syntax check is done based on XML schema validation. As *AML* introduces additional constraints and extensions which cannot be validated via schema validation, [32] proposes a semantic check based on a formalized set of validations.

The described approach transforms *AML* models into *UML* class diagrams and generates formalized constraints representing restrictions and extensions of *AML* in *OCL*. The work focuses on the presentation of the validation approach without implementing a full scale validation framework.
Whereas the approach from Schleipen requires the transformation of the *AML* model to an *UML* class diagram, the approach presented in this thesis does not require this

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

intermediate step. In addition, this thesis formalizes the whole *AML* specification in contrast to the prototypical approach described in [32].

Abele et. al utilize a different approach and present an ontology based validation approach for *AML* models [4]. Gruber defines an ontology as "a formal specification of a shared conceptualization of a domain of interest", which means that the abstract model of a specific domain is formalized in an common understandable way [21]. The described approach uses OWL [12] to represent ontologies.
To enable the validation, [4] defines a base ontology for representing general *AML* concepts in *OWL*. To transform *AML* models into an ontology representing the model a set of rules is required. The compliance of the *AML* model to [24] is analyzed with SPARQL [13] queries executed on the *OWL* ontology.

Sabou et. al also utilize ontologies to create the prototypical tool "AutomationML Analyzer" [31]. The application enables the user beside a graphical representation of interlinked model artefacts to execute *SPARQL* queries on *AML* models. The query functionality aims to enable the modeller to validate the consistency of the models and presents therefore a similar approach as [4].

Whereas the approach from Abele et. al and Sabou et. al provide an approach based on semantic web technologies which requires the creation of an ontology and the model's transformation, the approach presented in this thesis does not require this additional concepts.

Whereas the so far discussed work focus on static analysis of *AML* model, the next concepts also intend to inform about inconsistencies, which arise due to changes on class specifications. As *AML* class instances are created as copy of their originating classes, updates on the classes can lead to inconsistencies. Class instances can be considered as clones, whereas classes are describes as prototypes.

To address this situation Berardinelli, et. at [7] formalize multiple levels of compliance, clones can have with respect to their prototypes. In addition, a model for expressing the relationship between clones and prototypes is created.
Based on this foundation, a tool to report and (semi) automatically update clones depending on the required compliance level was implemented.

Avanieva et. al also proposes an approach to detect and correct inconsistencies due to *AML* model updates which is based on the VITRUVIUS framework [5]. In contrast to [7], which utilizes a comparison between two versions of the *AML* model, [5] uses a delta-based approach, which reacts based on the detection of atomic edit operations triggered by the user.

The concepts of Berardinelli, et. and Avanieva et. al provide in comparison to the static validation presented in this thesis also a dynamic validation. Dynamic validation enables feedback/warnings while updating a model and even (semi) automatically correct inconsistencies resulting of the update whereas the approach in this thesis only provides a statically valuation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.2 Standardization Approaches

The way towards an industry standard for data exchange is due to the involvement of many stakeholder, the need of many alignments, etc. time-consuming and complex. Therefore a step wise approach utilizing *AML* to reach this target is described in Section 3.2.1.

During the implementation of the validation framework presented in this thesis, the new *AML* version 2.1 was released. It was decided to complete the validation framework based on *AML* version 2.0, as the latter one is still an active standard and its main concepts are valid for *AML* version 2.1. The adaptions of the validation to *AML* version 2.1 is proposed as future work.
Section 3.2.2 presents the most important adaptions and improvements invented by *AML* version 2.1.
Section 3.2.3 presents a way to standardize object and attribute semantics in *AML* based on *ECLASS*.
In Section 3.2.4 a modelling approach of *AML* which enables the generation of product centric, process centric and resource centric views is presented.

3.2.1 Incremental Approach to reach Standardization

Draht proposes an approach to incrementally create an industry standard [16]. The concept depicts four maturity levels:

1. Maturity Level 1: Only syntax standardization, no standardization of semantics
2. Maturity Level 2: Mixture of semantically standardized and proprietary content
3. Maturity Level 3: Complete standardization of semantics, local application
4. Maturity Level 4: Complete standardization of semantics, global application

Especially for maturity level 1 and 2 it is essential to have a data format which enables the storage of semantically standardized and proprietary content in a syntactically standardized way. In addition, this approach requires the possibility to add metadata, e.g., the files origin to derive the used semantics, etc. As *AML* fulfills both requirements, [16] considers *AML* as the optimal data format for this approach.

3.2.2 Innovations of AutomationML 2.1

Whereas *AML* version 2.0 was based on *CAEX* version 2.15, the new *AML* version 2.1 uses the *CAEX* version 3.0 as base format [25]. The new *CAEX* version provides significant benefits, which are also utilized by *AML* 2.1 [15].

The main improvements of *AML* specified in [25] are:

- Invention of attribute type library
- Easier assignment of **RC**s on the instance level
- Possibility to define meta data on the object level

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The invention of an attribute type library enables standardization also on the attribute level and is a big step towards interoperability and machine readability [15].
Figure 3.1 depicts the *AttributeTypes* defined by *AML*.

In addition *AML* 2.1 provides the possibility to assign more than one **RC** as *RoleRequirement* for an **IE** [25]. So far, it was only possible to assign one *RoleRequirement* and further **RC** assignments had to be modelled as *SupportedRoleClass* [24]. The new approach simplifies the structure of *AML* models and is less error prone. Listing 3.1 depicts an **IE** which has two **RC**s assigned via child elements *RoleRequirements*.

```
1 <InstanceHierarchy Name="InstanceHierarchy">
2   <Version>0</Version>
3   <InternalElement Name="InternalElement" ID="GUID1">
4     <RoleRequirements RefBaseRoleClassPath="RoleClassLib/RoleClass1" />
5     <RoleRequirements RefBaseRoleClassPath="RoleClassLib/RoleClass2" />
6   </InternalElement>
7 </InstanceHierarchy>
```

Listing 3.1: Simplified **RC** assignment

Beyond that, *AML* 2.1 provides the possibility to define meta data also on the object level [15]. This improves traceability along the chain of tools which were used to update the model. Listing 3.2 depicts an **IE** which has a child element *SourceObjectInformation* which enables with the attributes *OriginID* and *SourceObjID* to provide additional meta information. The attribute *OriginID* contains the identifier of the tool the object when created [15]. It should not change over the object's lifetime. The attribute *SourceObjId* contains the proprietary identifier of the object in the source tool [15].

```
1 <CAEXFile SchemaVersion="3.0" FileName="MetaInfo.aml">
2  <SuperiorStandardVersion>AutomationML 2.10</SuperiorStandardVersion>
3  <SourceDocumentInformation OriginName="AutomationML Editor" OriginID="916578CA-FE0D-474E-
       ↪A4FC-9E1719892369" OriginVersion="6.1.3.0" LastWritingDateTime="2023-01-29T16
       ↪:07:43.2241672+01:00" OriginProjectID="unspecified" OriginProjectTitle="unspecified"
       ↪OriginRelease="6.1.3.0" OriginVendor="AutomationML e.V." OriginVendorURL="www.
       ↪AutomationML.org" />
4  <InstanceHierarchy Name="InstanceHierarchy">
5    <Version>0</Version>
6    <InternalElement Name="InternalElement" ID="GUID1">
7      <SourceObjectInformation OriginID="916578CA-FE0D-474E-A4FC-9E1719892369" SourceObjID="
          ↪GUID1" />
8    </InternalElement>
9  </InstanceHierarchy>
10 </CAEXFile>
```

Listing 3.2: InternalElement with Metainformation

3.2.3 Standardization of Semantics in *AML*

AML provides methods to describe semantics of objects and attributes but it is not the intention of *AML* to provide a standardized catalogue of semantics [20]. For this purpose *AML* utilizes *ECLASS* which represents a hierarchical semantic system for categorization of materials, products, etc [20].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

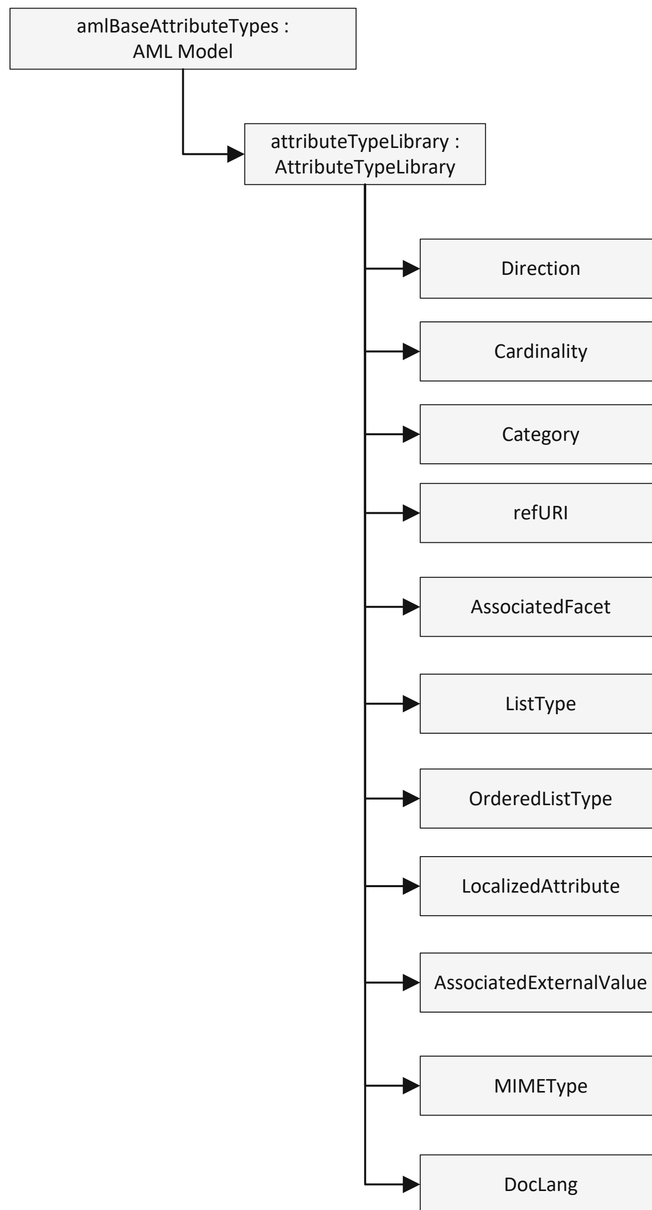

Figure 3.1: *AML* Base Attribute Types [25]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Whereas *ECLASS* is utilized in multiple use-cases, the following two are especially in the context of unambiguous data exchange important:

- Identification of object semantic
- Identification of attribute semantic

To utilize *ECLASS* for defining object semantic, [20] proposes to create a **RC** as depicted in Listing 3.3. This class should then be used as base-class for all further *ECLASS* related **RC** definitions.

```
1 <RoleClass Name="eClassClassSpecification" RefBaseClassPath="AutomationMLBaseRoleClassLib/
    ↪AutomationMLBaseRole" ID="GUID1">
2     <Attribute Name="Standard" AttributeDataType="xs:string" />
3     <Attribute Name="ClassificationClass" AttributeDataType="xs:string" />
4     <Attribute Name="IRDI" AttributeDataType="xs:string" />
5 </RoleClass>
```

Listing 3.3: RoleClass decribing *ECLASS* attributes

The attribute *Standard* is used to describe the version of the *ECLASS* catalog used [20]. To store the hierarchical classification following the 4 level structure (segment, main group, group, sub-group) of *ECLASS* the attribute *ClassificationClass* is used. The International Registration Data Identifier (*IRDI*) of the *ECLASS* class is stored in the attribute *IRDI*. Listing 3.4 depicts a **RC** of a robotic arm with the corresponding *ECLASS* attributes.

```
1 <RoleClass Name="RoboticArm" RefBaseClassPath="RoleClassLib/eClassClassSpecification">
2   <Attribute Name="Standard" AttributeDataType="xs:string">
3       <Value>ECLASS-13.0</Value>
4     </Attribute>
5     <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
6       <Value>27380107</Value>
7     </Attribute>
8     <Attribute Name="IRDI" AttributeDataType="xs:string">
9       <Value>0173-1#01-AGA572#005</Value>
10    </Attribute>
11 </RoleClass>
```

Listing 3.4: RoleClass depicting a robotic arm

Beside objects, *ECLASS* also provides an approach to define semantics for *AML* attributes utilizing the *AML* element *RefSemantic* with its attribute *CorrespondingAttributePath*. The *CorrespondingAttributePath* contains the *IRDI* of the attribute in *ECLASS*.

An exemplary *IRDI* looks as follows:
"IRDI-Path://0173-1—BASIC_1_1%2301-ABQ162#014/0173-1%2302-BAA018%23007“ [20].

For concrete structure of the *IRDI* path please refer to [20].

3.2.4 Process-Product-Resource Modelling Approach

In industrial production facilities a profound knowledge of its processes, products and resources is vital [33, 24]. But not only the view on the distinct items is important but

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

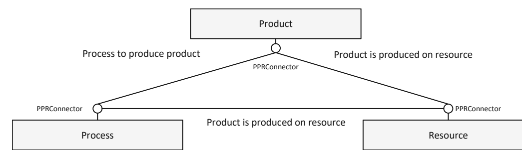

Figure 3.2: PPR Concept [24]

also the knowledge how they are connected. This is where the Process-Product-Resource (*PPR*) concepts steps in.
Industrial production processes have typically three views [33]:

- Process centric view: What processes are needed to produce products with existing resources
- Product centric view: What (sub)products are produced during execution of production processes involving resources
- Resource centric view: What resources are needed to execute various processes to generate products

Each of this view is relevant and valid and *AML* provides a way to establish the connections to be able to derive all views out of the *AML* models. Figure 3.2 shows the necessary categorizations and interfaces to enable the generation of views. For every relevant **IE** in the *AML* model one of the **RC**s *Process*, *Product* and *Resource* mentioned in Section 2.4.1 needs to be assigned. In addition, to be able to establish the necessary **ExtI** every element needs an instance of the **IC** *PPRConnector* mentioned in Section 2.4.2. Based on these categorization it is possible to define **IL**s to connect **IE** which are involved in the same step (e.g. connect the process, product and resource which have a relation during the production process).

As result of these established links in the *AML* model, individual views based on the users concrete interest can be created.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 4

Used Methods and Techniques

This chapter describes the utilized methods and tools for implementing the *AML* validation framework. The concept and the implementation is built upon is *Model-Driven Software Engineering* (*MDSE*) which is described in Section 4.1. Section 4.2 describes the 4 layer approach and presents basic modelling technologies. Section 4.3 presents the *Eclipse Modelling Framework* and describes its main functionalities. Beyond that, Section 4.4 describes model validation basics and presents two modelling validation languages.

4.1 Model-Driven Software Engineering Basics

The development of software is due to sophisticated requirements, highly interlinked systems and short timelines getting more and more complex. Also the knowledge of stakeholders is highly heterogeneous as they have typically different profiles (e.g. business responsible, software developer, etc) [9]. This increases the risk of misunderstandings, wrong implemented requirements and delayed projects.
To improve the situation *Model-Driven Software Engineering* (*MDSE*) intends to enhance the development process by utilizing the benefits of modelling in software engineering [9]. In a classical software development process, artefacts are typically created through the phases *Analysis*, *Design* and *Implementation*. The results of the particular phases are often non machine-readable and therefore cannot be directly used in the next phase [9]. In contrast *MDSE* intends to create artefacts as machine-readable models. These models are (semi)automatically refined with *model to model* (*M2M*) transformations throughout the process. On the one hand models enable a lossless handover of results to the next phase and on the other hand they can be used to improve mutual understanding between business and IT. Even code can be automatically created by *model to text* (*M2T*) transformations [9]. Figure 4.1 depicts a typical *MDSE* process.

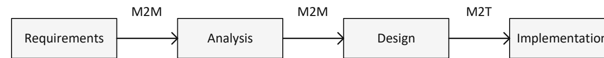

Figure 4.1: MDSE process [9]

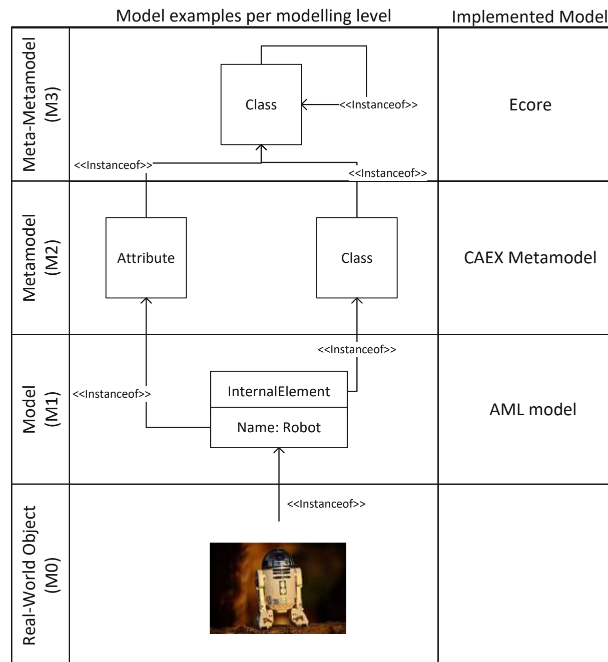

Figure 4.2: 4 Layer Metamodeling Stack [9]

4.2 Modelling Languages

As models as well as transformation need to be defined in a structured approach, *MDSE* utilizes modelling languages [9]. The used modelling languages conform to typical software language characteristics defined in [26]. As *MDSE* follows the “everything is a model” approach [10], modelling language and its representation are stored as models.
Figure 4.2 depicts the most known 4-layer approach [10] and also presents the modelling languages utilized in the implemented *AML* validation process.

Real world objects are represented in *M0*. An example is a *Robot* in a production facility. Starting from *M1*, models are used to create an abstraction of the reality. The modelling language applied in this thesis for *M1* is *AML*.
The next modelling layer *M2* is the metamodel, which is the “model of the model”. The metamodel is the formal specification of the model and defines what elements the model in *M1* can use [9]. The most popular metamodel representation is the *Unified Modelling*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

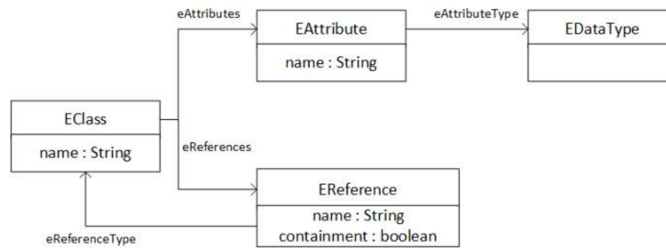

Figure 4.3: Simplified Ecore Metamodel [35]

Language (*UML*) [30] specified by the *Object Management Group* (*OMG*).
The implemented validation framework uses for this purpose the *CAEX* metamodel available at https://github.com/amlModeling/caex-workbench.
Finally, the meta-metamodel *M3* defines the meta language, which is used to build the metamodel. The best known meta-metamodel is the *Meta Object Facility* (*MOF*) [29]. As the specification of *MOF* is very complex, it was split into a comprehensive version *CMOF* (complete *MOF*) and *EMOF* (essential *MOF*).
The validation framework uses a meta-metamodel derived from *EMOF* called *Ecore*. It is the default meta-metamodel used by the Eclipse Modelling Framework [35] which is described in the next section.

4.3 Eclipse Modelling Framework

The Eclipse Modelling Framework (*EMF*) aims to provide the following functionalities [35]:

- Enabling the creation of modelling languages with *Ecore*
- Generation of model editors based on *Ecore* models
- Integration of *UML*, Java and *XML*
- Generation of Java APIs based on the *Ecore* models

The definition of modelling languages is the most important functionality of *EMF*. Is provides with its meta-meta language *Ecore* the most popular Java implementation of *EMOF* [35]. Figure 4.3 depicts a simplified *Ecore* metamodel with its main artefacts *EClass*, *EAttribute*, *EReference* and *EDataType* [35].

EClass is used to represent a modelled class, which is identified by its name. It can have attributes as well as references to other classes. *EAttribute* represents an attribute of a class, which has a name and a type, which is represented by *EDataType*. For establishing references *EReference* is utilized. It has a name and a flag to identify containment relations [35].
Figure 4.4 shows an excerpt of the *CAEX* metamodel which was created with *Ecore* and used for the implementation of the validation framework.

Based on the metamodels created with *Ecore*, *EMF* enables the creation of model editors. Figure 4.5 depicts an example of a *AML* model generated with the editor created

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

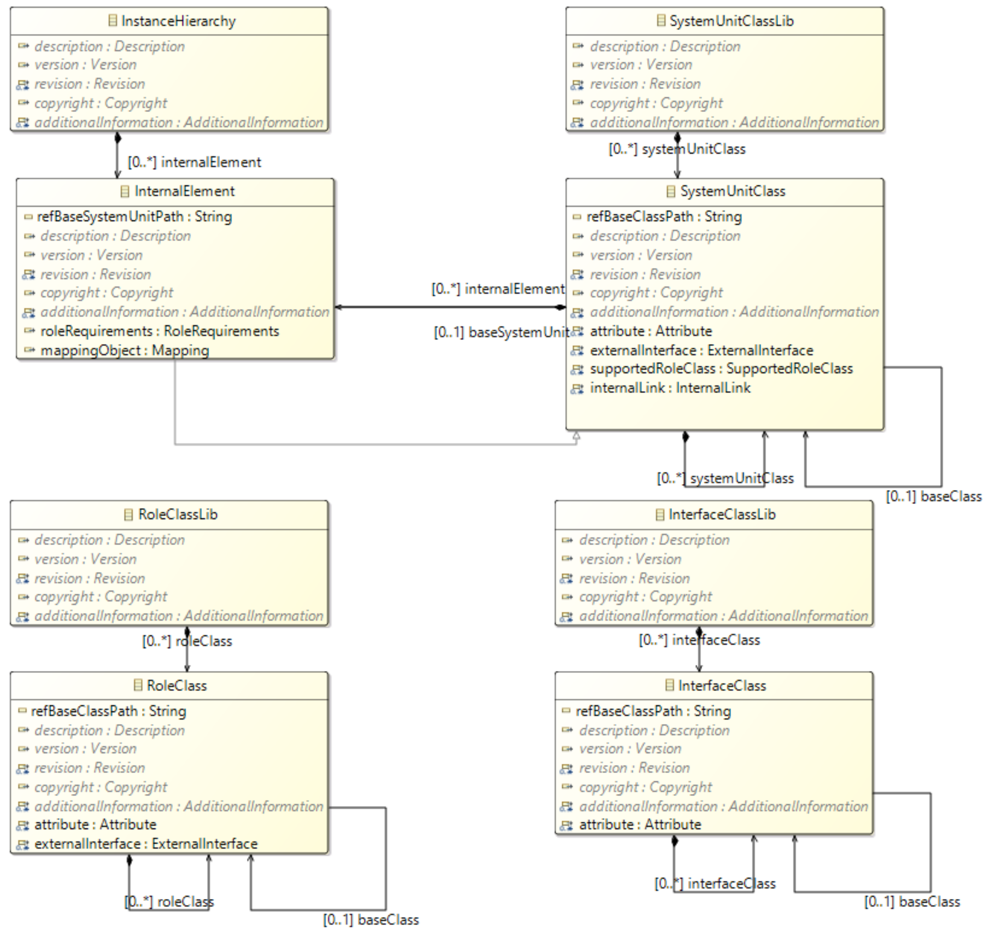

Figure 4.4: Excerpt of *CAEX* metamodel [36]

with *EMF*. These model editors can be used to create model instances based on the metamodel created with *Ecore*. The possibility to generate concrete models instances in a very early development phase prevents misunderstandings and improves common understanding [35, 9].
Another functionality of *EMF* is the integration of Java, *XML* and *UML* as depicted in Figure 4.6 [35]. It is only necessary to create e.g., a *XML* schema and *EMF* provides the functionality to create an *UML* model, an *EMF* model and even the corresponding Java API is automatically created. The user can create one of the enlisted artefacts depending on its personal preferences and skills and let *EMF* automatically generates the other artefacts. This reduces effort of standard code generation and allows software developer to focus on implementation of concrete business functionality [35].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

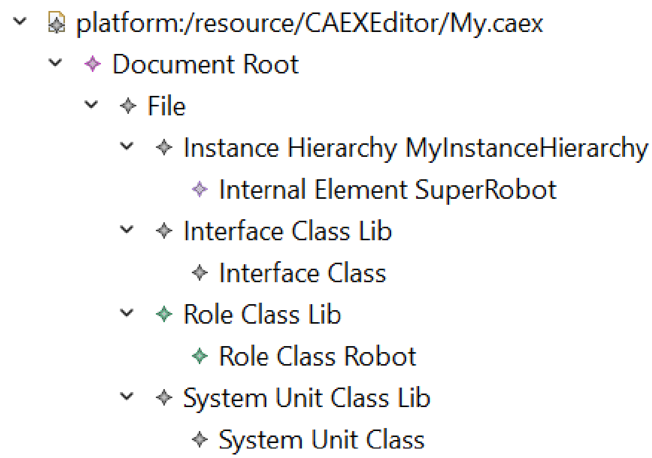

Figure 4.5: AML Model created with EMF editor

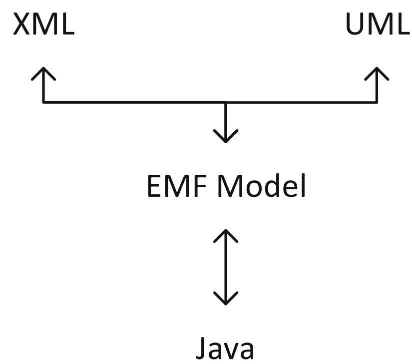

Figure 4.6: Integration of Java, XML and UML [35]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

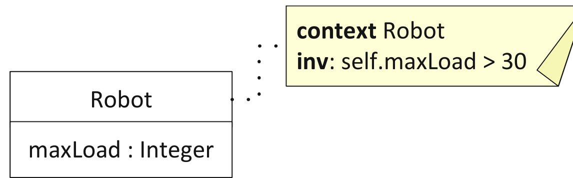

Figure 4.7: OCL Example

4.4 Model Validation

Although graphical modelling languages have many benefits they are not the best fit for every requirement e.g., model validation [11]. To not overload the expressiveness of modelling languages, model validation is typically implemented as textual language [11]. This section presents the *Object Constraint Language* which is specified by the *OMG*. In addition, it presents the *Epsilon Validation Language* which is part of the *Epsilon Platform* [2].

4.4.1 Object Constraint Language

The Object Constraint Language (*OCL*) is used to specify constraints on the model level which model instances have to conform to [11].
Figure 4.7 depicts a model representing a *Robot* and the requirement that instances of the model must be capable to exceed a maximum load of 30 as *OCL* validation.
Every *OCL* validation starts with the *context* which specifies the name of the type on which instances the constraint is executed on. The *invariant* (inv) defines the requirement the class instance has to fulfill.

Beyond this pattern for validation of object's instances, *OCL* provides basic types like *Interger*, *Real*, *String*, template types like e.g. *Set*, *Collection* or *Sequence* and various other features [11].
Anyhow, it has been recognized that *OCL* has shortcomings when it comes to e.g., multiple models, dependent constraints or the possibility to create warnings instead of errors [28, 27]. The result was the invention of the *Epsilon Validation Language* which is presented in the next section.

4.4.2 Epsilon Validation Language

The *Epsilon Validation Language* (*EVL*) [3] is part of the *Epsilon platform* [2] and therefore uses its *Epsilon Object Language* (*EOL*) as basis. *EOL* is based on *OCL* and extends its functionalities [28, 27].

In comparison to *OCL* the following main additions were made in *EVL*:

- Access of multiple models

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- Creation of custom operations
- Support for debugging

Whereas *OCL* only enables one specific model during validation, *EOL* provides the possibility to support multiple models [27]. In addition, the possibility to create custom operations is a major benefit of *EOL* to increase usability [27]. Beyond that, the possibility for debugging output was added in *EOL* which makes troubleshooting easier [27].

Listing 4.1 shows an exemplary *EVL* validation with its main parts *context definition* and *constraint definition*.

```
1  context CAEX!SystemUnitClass
2  {
3      constraint hasPortDirectionCorrectValue
4      {
5          guard: self.isPortInstance() and self.hasDirection()
6  
7          check
8          {
9              debugOutput("hasPortDirectionCorrectValue: " + self);
10 
11             var bRet : Boolean;
12             var mapPath : Map;
13             var ergMap : Map;
14             var caexFile : CAEXFile;
15             var attr : Attribute;
16 
17             caexFile = self.getCAEXFile();
18 
19             return self.checkValueDirection();
20         }
21 
22         message: ValidationExecution.addError(caexFile.filename, "", "", "
               ↪HasPortDirectionCorrectValue", "Instance of Type Port has invalid value
               ↪Instance: " + self.name)
23     }
24 }
```

Listing 4.1: Exemplary EVL listing

The context definition is initiated by the keyword **context** and is followed by the model name *CAEX*. After that, the operator **!** separates the model name from the concrete model class *SystemUnitClass*.

The constraint definition has a name and can contain the sections *guard*, *check*, *message* and *fix*. The constraint in Listing 4.1 is called *hasPortDirectionCorrectValue*.
The *guard* intends to limit the executions of constraints by specifying specific requirements which have to be fulfilled by the instance [3]. In this example, the instance needs to be to a *Port* and it the existence of the attribute *Direction* is a prerequisite.
The *check* block contains the validation of the instance. It has to return either *true* or *false* [3].
If the result of the *check* block is *false*, the *message* block provides a possibility to define an error message [3]. Beyond that, *EVL* provides with the *fix* block the possibility to execute code which automatically corrects found inconsistencies [3]. This functionality was not utilized within this thesis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Validation Framework

The validity of *AML* models is verified by the process depicted in Figure 5.1. The overall validation is split in a *CAEX validation phase* and a *AML validation phase*. The process consists of process steps which are executed sequentially. This process is orchestrated by a Java application. As validation rules depend on each other, the next process step is only executed if all prior steps were completed successfully.
The *CAEX preparation phase* ensures the models conformity to *CAEX*. The description of this phase can be found in Section 5.1.
The *AML validation phase*, the centerpiece of this thesis verifies the adherence of the *AML* model to the *AML* specification and is described in Section 5.2.
Section 5.3 presents some basic software metrics of the created validation framework.

The source of the implementation can be found at https://github.com/amlModeling/amlValidation.

5.1 *CAEX* Validation Phase

The *CAEX* validation phase ensures that the *AML* model conforms to the *CAEX* specification described in [23]. In contrast to the *AML* validation phase, the *CAEX* validation utilizes Schema validation [22]. The *CAEX* Schema CAEX_ClassModel_V2.15.xsd is provided as part of the *AML* reference editor *AutomationMLEditor* [1] and is used within the *CAEX* validation phase.
In case a violation of the specification is found, the validation is stopped and the problem is reported. In case the *AML* model conforms to the *CAEX* specification, the validation process proceeds to the *AML* validation phase.

5.2 *AML* Validation Phase

The process starts with the validation of the *AML* models document version. As the implementation supports only *AML* models which conform to *AML* version 2.0, this step

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

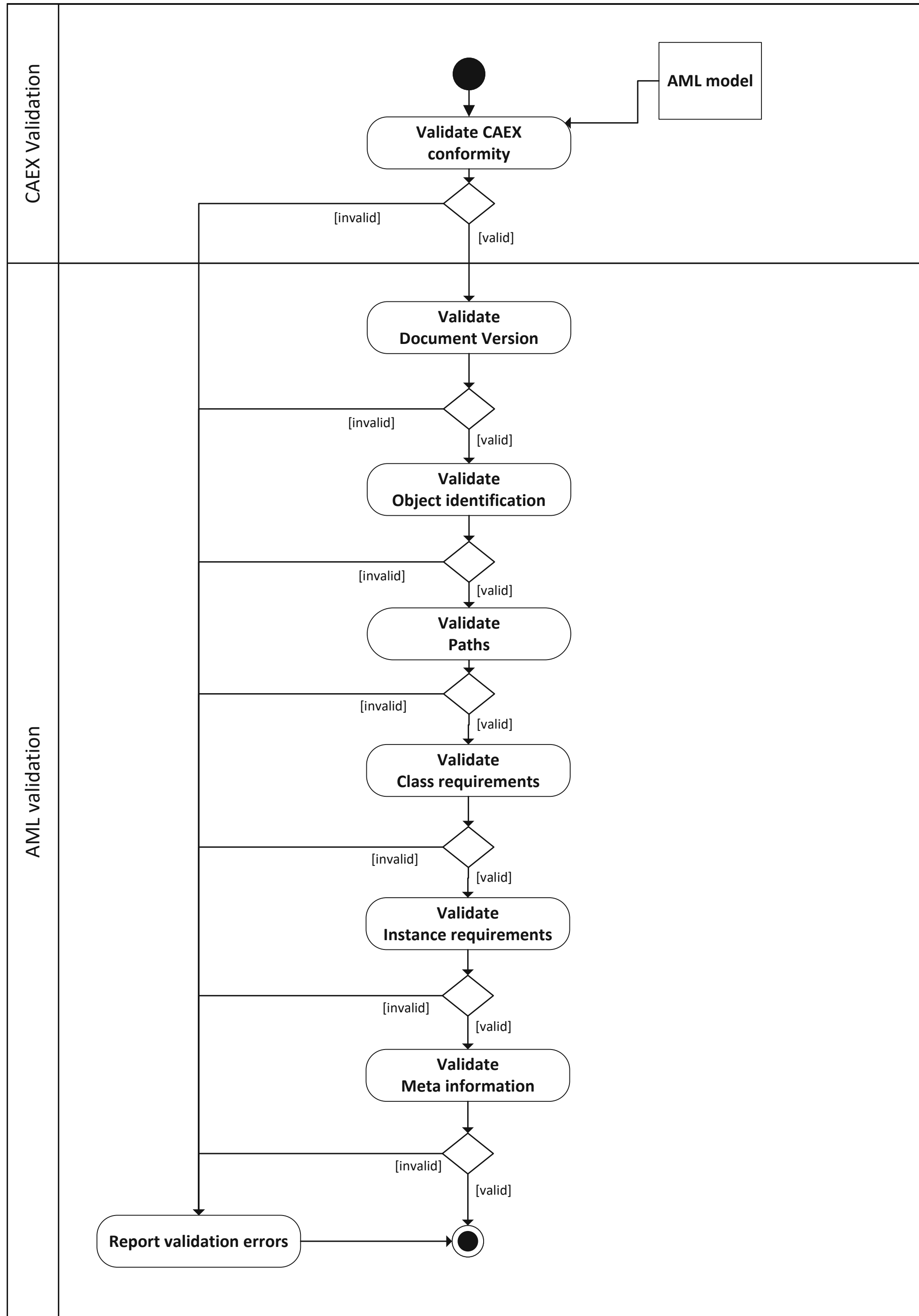

Figure 5.1: Validation framework

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

is a prerequisite for all subsequent *AML* validations and is described in Section 5.2.1. Section 5.2.2 describes how the implementation validates the uniqueness of object identifiers. As referencing of classes and instances is a common pattern in *AML* the validation of paths is essential and illustrated in Section 5.2.3.
Section 5.2.4 describes how requirements on the class level are validated. The next step towards a valid *AML* model is the verification of instance requirements, which is described in Section 5.2.5. The last artefact in the process is the validation of meta information of the model, which is explained in Section 5.2.6.

5.2.1 Validate Document Version

The first phase of the validation verifies the adherence of the *AML* model to requirements related to versioning described in Section 2.3.2.

1. Version of the *AML* model
   - constraint HasElementAdditionalInformation
   - constraint HasAttributeAutomationMLVersion
   - constraint HasCorrectAutomationMLVersion
2. Existence of library versions for **`IClib`**, **`RClib`** and **`SUClib`**
   - constraint HasLibraryVersion

The implementation verifies whether the model contains the element *AdditionalInformation* with the attribute *AutomationMLVersion*. It is essential that the model's version is "2.0" as all further validations are specific for this *AML* version.

Listing 5.1 shows the implementation of the constraint *HasCorrectAutomationMLVersion*. The validation depends on the constraint *HasAttributeAutomationMLVersion*, which is implemented as guard. This ensures that the check-block is only executed, if the constraint *HasAttributeAutomationMLVersion* is valid for the instance.
To make the constraints easier to understand, reduce maintainability efforts and provide the possibility to reuse artefacts, the implementation encapsulates validation logic in operations, which are called from the constraints. For this purpose the operations *getValidAutomationMLVersion* and *getAutomationMLVersionElements* were created.

Beside the *AML* version, also the existence of library version for every library is validated within this module. If the validated document does not conform to the version required, the validation stops and reports the miss-match as further analysis is not useful.

```
1  context CAEX!CAEXFile
2  {
3      constraint HasCorrectAutomationMLVersion
4      {
5          guard: self.satisfies("HasAttributeAutomationMLVersion")
6          check
7          {
8              var documentAMLVersion = self.getAutomationMLVersion();
9              return documentAMLVersion == getValidAutomationMLVersion();
10         }
11         message
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
12          {
13              return ValidationExecution.addError(self.filename, getValidAutomationMLVersion(),
14                  documentAMLVersion, "HasCorrectAutomationMLVersion", "");
15          }
16      }
17
18      constraint HasAttributeAutomationMLVersion
19      {
20          guard: self.satisfies("HasElementAdditionalInformation")
21          check
22          {
23              var automationMLVersions = self.getAutomationMLVersionElements();
24              return automationMLVersions.size() == 1;
25          }
26          message
27          {
28             return ValidationExecution.addError(self.filename, "", "", "
                   ↪HasAttributeAutomationMLVersion",
29              "No or multiple Attributes <AdditionalInformation>.AutomationMLVersion
                   ↪defined");
30          }
31      }
32  }
33
34  operation getValidAutomationMLVersion() : String
35  {
36    return "2.0";
37  }
38
39  operation CAEXFile getAutomationMLVersionElements() : Sequence
40  {
41      var elements : Sequence;
42
43      for(additionalInformation : Element in self.AdditionalInformation)
44      {
45          var attributes = additionalInformation.attributes.asSequence();
46          if (attributes.size <> 1)
47              continue;
48          for (attribut : Element in attributes)
49          {
50              if(attribut.name == "AutomationMLVersion")
51                  elements.add(attribut);
52          }
53      }
54      return elements;
55  }
```

Listing 5.1: Validation of *AML* version

5.2.2 Validate Object Identification

The second phase of the validation focuses on the uniqueness of library and class names and unambiguous identification of class instances as described in Section 2.3.4.

1. Uniqueness of library names of **ICLib**, **RClib** and **SUClib** within an *AML* file
    - constraint IsInterfaceClassLibUnique
    - constraint IsRoleClassLibUnique
    - constraint IsSystemUnitClassLibUnique

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2. Uniqueness of **IC**, **RC** and **SUC** names within their libraries
    - constraint IsInterfaceClassNameUnique
    - constraint IsSystemUnitClassNameUnique
    - constraint IsRoleClassNameUnique
3. Uniqueness of InternalElements and ExternalReferences IDs
    - constraint HasID
    - constraint IsIDGUID

The implementation is divided into two logical parts. On the one hand the uniqueness of identifier for libraries and classes and on the other hand the uniqueness of identifier for class instances is validated. Whereas for libraries and classes the relevant identifier which has to be unique is stored in the attribute *Name*, for class instances the identifier is stored in the attribute *ID*.

Listing 5.2 depicts the validation of the uniqueness of **IClib**s. The implementation utilizes for finding duplicates a Map as shown in the operation *getDuplicateCAEXObjects*. It is vital to recognize that the context for this constraint is a specific *AML* file.

```
1
2  context CAEX!CAEXFile
3  {
4      constraint IsInterfaceClassLibUnique
5      {
6          /* Same libraries of different versions are forbidden to be stored in the same
               ↪AML file.*/
7
8          check
9          {
10             var duplicateLibs : Sequence;
11             duplicateLibs = getDuplicateCAEXObjects(self.interfaceClassLib);
12             return duplicateLibs.isEmpty();
13         }
14         message
15         {
16             return ValidationExecution.addError(self.filename, "", "", "
                   ↪IsInterfaceClassLibUnique", "InterfaceClassLibNames not unique:
                   ↪Duplicates: " + duplicateLibs.getAsCommaSeparatedString());
17         }
18     }
19
20     operation getDuplicateCAEXObjects(objects : OrderedSet) : Sequence
21     {
22         var duplicates : Sequence;
23         var keyMap : Map;
24         var objName : String;
25
26         for(obj : CAEXObject in objects)
27         {
28             objName = obj.name;
29             if(not keyMap.containsKey(objName))
30             {
31                 keyMap.put(objName, objName);
32             }
33             else
34             {
35                 duplicates.add(objName);
36             }
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
37         }
38         return duplicates;
39     }
40 }
```

Listing 5.2: Validation of **IClib** uniqueness

On the contrary, the validation of identifiers for class instances checks if they are globally unique. Therefore all instances - including referenced *AML* models - are included during this validation step as shown in Listing 5.3.

In this implementation the *EVL* functionality of a “Pre-block” is utilized. The code located in a “Pre-block” is executed before all constraints and is in this scenario used to store the number of appearances of each identifier in a Map. The constraint *IsIDGUID* checks via the operation *getAnzForID* if the used identifier is globally unique.

```
1
2  pre
3  {
4      var colExternalReferences : Collection;
5      var colInternalElements : Collection;
6
7      colInternalElements = CAEX!InternalElement.getAllOfType();
8      colExternalReferences = CAEX!ExternalInterface.getAllOfType();
9
10     for (internalElement : InternalElement in colInternalElements)
11         updateIDMap(mapIDs, internalElement.ID);
12     for (externalInt : ExternalInterface in colExternalReferences)
13         updateIDMap(mapIDs, externalInt.ID);
14 }
15
16 context CAEX!InternalElement
17 {
18     constraint IsIDGUID
19     {
20         guard: self.satisfies("HasID")
21         check
22         {
23             var caexFile : CAEXFile;
24             caexFile = self.getCAEXFile();
25             return getAnzForID(self.ID) == 1;
26         }
27         message: ValidationExecution.addError(caexFile.filename, "", "", "IsIDGUID", "
               ↪InternalElement " + self.name + " has a non unique ID: " + self.ID)
28     }
29 }
30
31 operation updateIDMap(mapIDs : Map, id : String)
32 {
33     var anz_aktID : Integer;
34
35     if (mapIDs.containsKey(id))
36     {
37         anz_aktID = mapIDs.get(id);
38         anz_aktID = anz_aktID +1;
39         mapIDs.put(id, anz_aktID);
40     }
41     else
42         mapIDs.put(id, 1);
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
43     }
44 }
45
46 operation getAnzForID(id : String) : Integer
47 {
48     return mapIDs.get(id);
49 }
```

Listing 5.3: Validation of ID uniqueness

If the naming of classes, libraries or identifier of instances are not unique the validation stops and reports the violation of the *AML* specification.

5.2.3 Validate Paths

The third phase of the validation focuses on the validity of paths. It follows the description of the concepts described in Sections 2.3.6.2, 2.3.6.3, 2.3.6.4 and 2.3.5.

1. Validity of paths used for inheritance
   - constraint IsRefBaseClassPathValid
2. Validity of paths used for **SUC**-instance relations
   - constraint IsRefBaseSystemUnitPathValid
3. Validity of paths used for **RC**-instance relations
   - constraint IsRefRoleClassPathValid
   - constraint IsRefBaseRoleClassPathValid
4. Validity of paths used for establishing **IL**s between interfaces
   - ArePartnerSidePathsValid

The validation of paths within *AML* models is separated based on the concrete use-case of the path. This results out of the fact, that *AML* uses for different use-cases different attribute names and concepts how the path is assembled. The implementation approach for all path validations is to check if the path can be successfully resolved.

For the validation of paths in context of inheritance relations, the implementation searches for the class specified in the attribute "RefBaseClassPath". In case the path does not contain any library name, the validation checks as described in Section 2.3.6.2 the direct parent of the child class. This validation is executed for **SUC**s, **RC**s and **IC**s. The implementation of the constraint *IsRefBaseClassPathValid* is depicted in Listing 5.4.

The validation of paths in context of class-instance relations searches for the **SUC** specified in the attribute "RefBaseSystemUnitPath". In this scenario, also the special case "mirror object" (a object is a copy of another instance exclusive it's child objects) is validated. This case is recognized based on the fact that the attribute "RefBaseSystemUnitPath" contains instead of a **SUC** the "ID" of another **IE**.

The validation of paths in context of an **IE** supporting **RC**s checks one specific **RC** as well as multiple **RC**s. In case of one **RC** the validation searches for the **RC** specified in the attribute "RefBaseRoleClassPath" of the element "RoleRequirement". In case additional

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

RCs are specified via “SupportedRoleClass” elements, the validation also checks the existence of these **RC**s specified in the attribute “RefRoleClassPath”.

The path validation for **IL**s checks the existence of of the referenced **IC**s. Therefore it searches for the **IC**s specified in the attributes “RefPartnerSideA” and “RefPartnerSideB”.

If during the validation an error is found, the validation process is stopped and the error reported.

```
1  context CAEX!CAEXObject
2  {
3      constraint IsRefBaseClassPathValid
4      {
5          guard: self.isAMLBaseClass()
6          check
7          {
8              debugOutput("IsRefBaseClassPathValid: " + self);
9              if (not self.refBaseClassPath.isDefined())
10                 return true;
11             var mapPath : Map;
12             var validationError : String;
13             var caexFile : CAEXFile;
14             var libType : String;
15 
16             caexFile = self.getCAEXFile();
17             mapPath = parsePath(self.refBaseClassPath);
18             libType = self.getLibType();
19             validationError = self.isRefBaseClassPathValid(mapPath, libType);
20             return validationError.length() == 0;
21         }
22         message
23         {
24             return ValidationExecution.addError(caexFile.filename, "", "", "
                   ↪IsRefBaseClassPathValid", "RefBaseClassPath not valid: " +
                   ↪validationError);
25         }
26     }
27 }
28 operation CAEXObject isRefBaseClassPathValid(pathMap : Map, libType : String) : String
29 {
30     var searchParent : Boolean = true;
31     var mirrorSearch : Boolean = false;
32     var ret : Boolean;
33     var retMap : Map;
34     var obj : CAEXObject;
35     var errTxt : String;
36 
37     ret = self.searchItem(pathMap, searchParent, mirrorSearch, retMap, libType);
38 
39     if (!ret)
40     {
41         errTxt = retMap.get("ErrorMsg");
42     }
43 
44     return errTxt;
45 }
```

Listing 5.4: Validation of Baseclass path

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

5.2.4 Validate Class Requirements

The fourth phase of the validation focuses on requirements of *AML* classes defined in Section 2.4.

1. Requirements for InterfaceClasses
   - constraint isDerivedFromAMLBaseClass
2. Requirements for RoleClasses
   - constraint isDerivedFromAMLBaseRoleClass
3. Requirements for SystemUnitClasses
   - isRelatedToAMLBaseRoleClass

The validation of **ICs** recursively checks based on the attribute “RefBaseClassPath” if it is direct or indirect derived from one of the base interface classes defined in Section 2.4.2. Listing 5.5 shows the constraint *isDerivedFromAMLBaseClass* including the operation *isClassChildOf*, which is used for the recursion through the hierarchy.

```
1  context CAEX!InterfaceClass
2  {
3      constraint isDerivedFromAMLBaseClass
4      {
5          guard: self.isTypeOf(InterfaceClass)
6
7          check
8          {
9              debugOutput("isDerivedFromAMLBaseClass: " + self);
10
11             var caexFile : CAEXFile;
12             var ergMap : Map;
13             var bRet : Boolean;
14             var objName : String;
15
16             caexFile = self.getCAEXFile();
17
18             if (not self.refBaseClassPath.isDefined() and not isBaseClassDerivedFrom("
                   ↪AutomationMLBaseInterfaces", self.name))
19             {
20                 objName = self.name;
21                 bRet = false;
22             }
23             else
24             {
25                 self.isClassChildOf(ergMap, "AutomationMLBaseInterfaces", "InterfaceClass");
26                 objName = ergMap.get("MissingObject");
27                 bRet = not objName.isDefined();
28             }
29             return bRet ;
30         }
31         message: ValidationExecution.addError(caexFile.filename, "", "", "
               ↪IsDerivedFromAMLBaseClass", "Class not derived from AMLInterfaceclass: " +
               ↪objName)
32     }
33 }
34
35 operation CAEXObject isClassChildOf(erg : Map, searchCategory : String, libType : String) : Boolean
36 {
37     var baseClass : InterfaceClass = null;
38     var mapPath : Map;
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
39
40     if (isBaseClassDerivedFrom(searchCategory, self.name))
41     {
42         return true;
43     }
44     if(self.refBaseClassPath.isDefined())
45     {
46         var searchParent : Boolean = true;
47         var mirrorSearch : Boolean = false;
48         var ret : Boolean;
49         var retMap : Map;
50
51         mapPath = parsePath(self.refBaseClassPath);
52         ret = self.searchItem(mapPath, searchParent, mirrorSearch, retMap, libType);
53         if (ret == false)
54         {
55             erg.put("ErrorMsg", retMap.get("ErrorMsg"));
56             erg.put("MissingObject", retMap.get("MissingObject"));
57             return false;
58         }
59         else
60             return retMap.get("obj").isClassChildOf(erg, searchCategory, libType);
61     }
62 }
```

Listing 5.5: Validation if **IC** is derived from AML baseclass

For **RC** the same logic is applied to check if the **RC** is direct or indirectly derived from one of the base role classes defined in Section 2.4.1. For **SUC** the implementation verifies whether the class is assigned direct or indirect to a base role class via the attribute "RefRoleClassPath" of the element "SupportedRoleClass". Again, if any of the mentioned validations fails, the execution is stopped and the violation of the *AML* specification is reported.

5.2.5 Validate Instance Requirements

The fifth phase of the validation focuses on requirements of *AML* instances defined in Sections 2.4.2.1, 2.4.2.2, 2.4.1.1, 2.4.1.2, 2.4.1.3 and 2.4.1.4.

1. Requirements for Order Interface instances
    - constraint HasOrderInstanceDirection
    - constraint HasOrderInstanceDirectionCorrectValue
2. Requirements for ExternalDataConnector Interfaces
    - constraint hasExternalDataConnectorRefURI
    - constraint hasExternalDataConnectorRefUIIValue
3. Requirements for Ports
    - hasPortDirectionCorrectValue
    - hasPortCardinalityCorrectValue
    - hasPortExternalInterface
    - hasPortConnector
    - hasPortNoChildInternalElements
    - AreConnectedPortOfSameCategory

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

4. Requirements for Facets
   - AreFacetAttributesValid
   - AreFacetInterfacesValid
   - NoAdditionalChildElementsInFacets
   - FacetNameIsUniqueWithinSiblings
5. Requirements for Groups
   - AreGroupChildrenValid
6. Requirements for PropertySets
   - ArePropertySetMappingsValid

This validation step is split based on the concrete instance type and consists of one or more checks.

For instances of the **IC** *Order* the validation checks whether the attribute *Direction* exists and has one of the values "In", "Out" or "InOut". Listing 5.6 shows the implementation of the validation.

```
1
2  context CAEX!ExternalInterface
3  {
4      constraint HasOrderInstanceDirectionCorrectValue
5      {
6          guard: self.satisfies("HasOrderInstanceDirection")
7          check
8          {
9              debugOutput("HasOrderInstanceDirectionCorrectValue: " + self);
10
11             var bRet : Boolean;
12             var ergMap : Map;
13             var caexFile : CAEXFile;
14             var attr : Attribute;
15             caexFile = self.getCAEXFile();
16             return self.checkValueDirection();
17         }
18         message: ValidationExecution.addError(caexFile.filename, "", "", "
               ↪HasOrderInstanceDirectionCorrectValue", "Instance of Type Order has invalid
               ↪ value Instance: " + self.name)
19     }
20 }
21
22 operation CAEXObject checkValueDirection() : Boolean
23 {
24     var ergMap : Map;
25     var attr : Attribute;
26
27     if (self.isKindOf(SystemUnitClass) or self.isTypeOf(ExternalInterface))
28     {
29         searchAttribute(self.attribute, "Direction", false, ergMap);
30         attr = ergMap.get("Attr");
31
32         if (attr.value == "In" or attr.value == "Out" or attr.value == "InOut")
33             return true;
34     }
35     return false;
36 }
```

Listing 5.6: Validation if Direction of Order Instance has correct value

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

The validation checks for instances of the **IC** *ExternalDataConnector* the existence of the mandatory attribute *RefURI*. In addition, it validates, if it has a value. The validity of the value is due to the fact, that the validation only checks the *CAEX AML* part of *AML*, not validated.

For instances which have an direct or indirect relation to the **RC** *Port* multiple aspects are validated. Beside correct values of the attributes *Direction* and *Cardinality* the existence of **ExtI**s which should be connected is verified. As a *Port* is supposed to be connected to another *Port* the existence of a **IC** representing the *PortConnector* is checked. As only *Ports* of the same category are allowed to be connected, also this condition is checked. Beyond that, the implementation checks if the *Port* instance does not contain any **IE** child elements. Listing 5.7 shows the validation which verifies if the *Port* contains **ExtI**s to connect.

```
1
2  context CAEX!SystemUnitClass
3  {
4  constraint hasPortExternalInterface
5      {
6          guard: self.isPortInstance()
7
8          check
9          {
10             debugOutput("hasPortExternalInterface: " + self);
11             var caexFile : CAEXFile;
12             caexFile = self.getCAEXFile();
13
14             for (extInt : ExternalInterface in self.externalInterface)
15             {
16                 if(not extInt.isDerivedFromPortConnector())
17                     return true;
18             }
19
20             return false;
21
22         }
23         message: ValidationExecution.addError(caexFile.filename, "", "", "
               ↪HasPortExternalInterface", "Port without ExternalInterface (exkl
               ↪PortConnector): " + self.name)
24     }
25 }
26
27 operation SystemUnitClass isPortInstance() : Boolean
28 {
29     var boolIEPort = self.isTypeOf(InternalElement) and self.roleRequirements.isDefined() and
           ↪self.hasRoleRequirementPort();
30     var boolSUCPort = self.isTypeOf(SystemUnitClass) and self.supportedRoleClass.isDefined()
           ↪and self.hasSupportedRoleClassPort();
31     var isPort = boolIEPort or boolSUCPort;
32
33     return isPort;
34 }
```

Listing 5.7: Validation if Port has ExternalInterfaces

For instances related to the **RC** *Facet* the existence of the attributes and interfaces of the *Facet* are checked in its parent object. In addition an unambiguous naming of the *Facet*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

within its siblings is validated. As the *Facet* intends to only provide a sub-view of the object, the implementation also checks for invalid **IE** within the instance.

For instances with relation to the **RC** *Group* the existence of all attributes of the *Group* is checked in its parent object.

For instances related to the **RC** *PropertySet* the existence of attributes in the **IE** and **RC** as well as in the *RoleRequirement* is validated.

If any of the mentioned validations fails, the execution is stopped and the violation of the *AML* specification is reported.

5.2.6 Validate Meta Information

The sixth phase of the validation focuses on requirements of the *AML* model's meta data described in Section 2.3.3.

- constraint HasElementWriterHeader
- constraint ValidateWriterHeader

As the intention of the meta data is to have a history of all tools which were used to update the model, a model can contain multiple entries *WriterHeader* which represent the information. Therefore the implementation gathers first all relevant objects and then checks their validity as described in Listing 5.8.

```
1 context CAEX!CAEXFile
2 {
3     constraint ValidateWriterHeader
4     {
5         guard: self.satisfies("HasElementWriterHeader")
6 
7         check
8         {
9             var writerHeaders = self.getWriterHeaderElements();
10            var bValid : Boolean = true;
11            var validationErrors : Sequence;
12 
13            for (writerHeader : Element in writerHeaders)
14            {
15                validationErrors.addAll(writerHeader.validateWriterHeader(self.filename));
16            }
17            return validationErrors.isEmpty();
18        }
19        message: ValidationExecution.addErrors(self.filename, "ValidateWriterHeader",
              ↪validationErrors)
20    }
21 }
22 
23 operation CAEXFile getWriterHeaderElements() : Sequence
24 {
25   var elements : Sequence;
26 
27   for(additionalInformation : Element in self.AdditionalInformation)
28   {
29     for (nestedElement : Element in additionalInformation.nestedElements)
30     {
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

```
31       if (nestedElement.name == "WriterHeader")
32         elements.add(nestedElement);
33     }
34   }
35   return elements;
36 }
37
38 operation Element validateWriterHeader(filename : String) : Sequence
39 {
40   var elementMap : Map;
41   var validationErrors : Sequence;
42   var iPosition : Integer = 0;
43   var bCorrectOrder : Boolean = true;
44   var bElementMissing : Boolean = false;
45   var bElementDuplicate : Boolean = false;
46   var duplicateElements : Sequence;
47   var value : String;
48   var elementNamesOrder : Sequence = Sequence {"WriterName", "WriterID", "WriterVendor", "
       ↪WriterVendorURL", "WriterVersion",
49         "WriterRelease", "LastWritingDateTime", "WriterProjectTitle", "WriterProjectID"};
            ↪
50   var elementNamesMandatory: Sequence = Sequence {"WriterName", "WriterID", "WriterVendor", "
       ↪WriterVendorURL", "WriterVersion",
51         "WriterRelease", "LastWritingDateTime"};
52
53   for (writeHeaderElement : Element in self.nestedElements)
54   {
55     if(not elementMap.containsKey(writeHeaderElement.name))
56       elementMap.put(writeHeaderElement.name, writeHeaderElement.value);
57     else
58       bElementDuplicate = true;
59
60     bCorrectOrder = bCorrectOrder and writeHeaderElement.name == elementNamesOrder.at(
         ↪iPosition);
61     iPosition++;
62   }
63
64   for (elementName : String in elementNamesMandatory)
65   {
66     if(not elementMap.containsKey(elementName))
67     {
68       validationErrors.add(elementName + " not defined");
69       bElementMissing = true;
70     }
71     else
72     {
73       value = elementMap.get(elementName);
74       if (value == null)
75         validationErrors.add(elementName + " empty");
76       else
77       {
78         if (elementName == "LastWritingDateTime")
79         {
80           if (not value.matches("(\\d{4})-(\\d{2})-(\\d{2})T(\\d{2}):(\\d{2}):(\\d{2}).*"))
81             validationErrors.add(elementName + " '" + value + "' is not of type DateTime");
                ↪
82         }
83       }
84     }
85   }
86   if (bElementDuplicate and not bElementMissing)
87     validationErrors.add("Duplicate elements detected");
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
88   if (!bCorrectOrder and not bElementMissing)
89     validationErrors.add("Order of <WriterHeader> elements not correct");
90   return validationErrors;
91 }
```

Listing 5.8: Validation of WriterHeader objects

5.3 Software Metrics of the Validation Framework

This section gives an overview of some basic software metrics of the implemented validation framework.

The implementation is for the purpose of metric calculations split into three parts:

- Environment Setup and process control (**P1**)
- *CAEX* validation phase (**P2**)
- *AML* validation phase (**P3**)

The code necessary for setup of the *EVL* environment, loading of *AML* models, overall validation process control etc. is aggregated in **P1**.
Source code, which is specifically necessary for the *CAEX* schema validation is considered in **P2**.
The *EVL* implementation, which is the centerpiece of the validation framework is aggregated in **P3**.

Table 5.1 depicts some key metrics for implementation of the validation framework:

Part	Metric	Value
P1	Lines Of Code	720
P2	Lines Of Code	50
P3	Lines Of Code	2804
P3	Number of *EVL* constraints	42
P3	Number of *EVL* operations	78

Table 5.1: Basic software metrics of the validation framework

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Evaluation

Whereas the previous chapter focused on the verification of *AML* models, this chapter presents the evaluation of the implementation itself. It is crucial that the validation process is error free and does not report false positives (validation errors for correct models) or false negatives (not detecting validation errors). The evaluation is implemented as JUnit project and follows the modular structure of the validation's implementation and consists of 152 *AML* models.
The source of the evaluation can be found at https://github.com/amlModeling/amlValidation.

6.1 General Evaluation Approach

To enable also a partial testing of the implementation the following categories were implemented:

- Evaluation of *AML* document version constraints
- Evaluation of *AML* object identification constraints
- Evaluation of *AML* path constraints
- Evaluation of *AML* class requirements constraints
- Evaluation of *AML* instance requirements constraints
- Evaluation of *AML* meta information constraints

The intention of the evaluation is to validate the correctness of every implemented *EVL* constraint based on a model, which violates the conditions checked by the constraint. If a constraint is applicable on multiple types, its functionality is validated on all possible types by multiple models. In addition, as constraints might be triggered by multiple constellations for the same type, for specific constraints multiple test model scenarios were created. To ensure that a correct model is not reported as faulty, also valid models are tested during the evaluation. To improve readability, *AML* listings in the following sections only contain the relevant aspects of the model.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The following sections describe the evaluation categories, present the number of models used per constraint and instance type and show how many of the inconsistent *AML* models were recognized as inconsistent. Beyond that, also the number of valid models, tested as valid is shown. To get a better understanding of the utilized tests, exemplary test models and their unit tests are shown.

6.2 Evaluation of *AML* Document Version Validation

To verify the correct functionality of the approach described in 5.2.1, the models in Table 6.1 were created.
Whereas the correct functionality of detecting invalid *AML* versions only needs to be evaluated on the *AML* model level, the correctness with respect to library versions is examined based on **SUC**s, **IC**s and **RC**s.

	Test artefact	Invalid models	Tested invalid	Valid Models	Tested valid
Version of *AML* model	*AML* model	2	2	1	1
Library Version	**SUC**	2	2	1	1
Library Version	**IC**	2	2	1	1
Library Version	**RC**	2	2	1	1

Table 6.1: *AML* Document Version Validation

An exemplary model utilized for the testing of the *AML* model's version is depicted in Listing 6.1. The corresponding JUnit test is shown in Listing 6.2.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <CAEXFile FileName="Testcase.aml" SchemaVersion="2.15" xsi:noNamespaceSchemaLocation="
      ↪CAEX_ClassModel_V2.15.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3   <AdditionalInformation AutomationMLVersion="3.0"/>
4 </CAEXFile>
```

Listing 6.1: Invalid *AML* version

```
1 @Test
2 public void WrongAutomationMLVersion() throws Exception
3 {
4     String modelPath = TestModelPath + "AMLVersion\\WrongAutomationMLVersion/";
5     addExpectedTestResult("Testcase.aml", "HasCorrectAutomationMLVersion", "3.0", "2.0", "")
        ↪;
6     Assertions.assertTrue(executeAndValidateTest(modelPath));
7 }
```

Listing 6.2: JUnit test WrongAutomationMLVersion

6.3 Evaluation of *AML* Object Identification Validation

To verify the correct functionality of the approach described in 5.2.2, the models in Table 6.2 were created.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

	Test artefact	Invalid models	Tested invalid	Valid Models	Tested valid
Unique library names	**IC**	1	1	1	1
Unique library names	**RC**	1	1	1	1
Unique library names	**SUC**	1	1	1	1
Unique class names	**IC**	1	1	1	1
Unique class names	**RC**	1	1	1	1
Unique class names	**SUC**	1	1	1	1
Unique instance IDs	**IE**	8	8	1	1
Unique instance IDs	**ExtI**	19	19	1	1

Table 6.2: *AML* Object Identification Validation

It has to be emphasized, that especially for evaluation of the validation of unique instance IDs, various *AML* model constellations (one *AML* file, multiple *AML* files, multiple **IH**s, etc.) were considered in the evaluation.
Listing 6.3 and Listing 6.4 show an exemplary testcase consisting of a root *AML* model with an external reference to another *AML* model. Both *AML* models have two **IH**s with multiple **IE**s sharing the same IDs. Listing 6.5 shows the corresponding JUnit testcase.

```
1  <?xml version="1.0" encoding="utf-8" standalone="no"?>
2  <CAEXFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" FileName="Testcase.aml"
       ↪SchemaVersion="2.15" xsi:noNamespaceSchemaLocation="./Source/CAEX_ClassModel_V2.15.
       ↪xsd">
3    <ExternalReference Alias="ReferencedLib" Path="ReferencedLib.aml"/>
4    <InstanceHierarchy Name="InstanceHierarchy1">
5      <Version>1.0.0</Version>
6      <InternalElement Name="InternalElement11" ID="9c99111a-937d-402e-8941-a390a0b5bb3e">
7        <InternalElement Name="InternalElement12" ID="8e362e89-2c88-4bb6-827a-a26f48e29ada">
8          <InternalElement Name="InternalElement13" ID="7a16a017-350d-478f-8f75-6d471c25a278
             ↪" />
9        </InternalElement>
10     </InternalElement>
11     <InternalElement Name="InternalElement21" ID="e56339ec-1ddf-41b0-8d70-139c606ce916">
12       <InternalElement Name="InternalElement22" ID="f5973236-6c1f-43d1-ac7f-cf8b34f33021" /
           ↪>
13     </InternalElement>
14     <InternalElement Name="InternalElement31" ID="3e402fb2-ca24-4016-8771-ae5f17a482a4" />
15   </InstanceHierarchy>
16   <InstanceHierarchy Name="InstanceHierarchy2">
17     <Version>1.0.0</Version>
18     <InternalElement Name="InternalElement51" ID="1a51459a-539e-4197-b506-e304ee51e5c4">
19       <InternalElement Name="InternalElement52" ID="c9be9315-5a25-4cf3-aea6-859db743636d">
20         <InternalElement Name="InternalElement53" ID="72ae2ad5-312e-42a1-b90f-b208f1e9aadd
             ↪" />
21       </InternalElement>
22     </InternalElement>
23     <InternalElement Name="InternalElement61" ID="98fe215b-ecb0-48b8-938b-084686663a9a" />
24   </InstanceHierarchy>
25 </CAEXFile>
```

Listing 6.3: Non unique IDs with external reference - root model

```
1  <?xml version="1.0" encoding="utf-8" standalone="no"?>
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
2  <CAEXFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" FileName="ReferencedLib.
     ↪aml" SchemaVersion="2.15" xsi:noNamespaceSchemaLocation="CAEX_ClassModel_V2.15.xsd">
3    <InstanceHierarchy Name="InstanceHierarchy51">
4      <Version>1.0.0</Version>
5      <InternalElement Name="InternalElement511" ID="9c99111a-937d-402e-8941-a390a0b5bb3e">
6        <InternalElement Name="InternalElement512" ID="8e362e89-2c88-4bb6-827a-a26f48e29ada"
           ↪>
7          <InternalElement Name="InternalElement513" ID="7a16a017-350d-478f-8f75-6
             ↪d471c25a278" />
8        </InternalElement>
9      </InternalElement>
10     <InternalElement Name="InternalElement521" ID="e56339ec-1ddf-41b0-8d70-139c606ce916">
11       <InternalElement Name="InternalElement522" ID="f5973236-6c1f-43d1-ac7f-cf8b34f33021"
           ↪ />
12     </InternalElement>
13     <InternalElement Name="InternalElement531" ID="3e402fb2-ca24-4016-8771-ae5f17a482a4" /
         ↪>
14   </InstanceHierarchy>
15   <InstanceHierarchy Name="InstanceHierarchy52">
16     <Version>1.0.0</Version>
17     <InternalElement Name="InternalElement551" ID="1a51459a-539e-4197-b506-e304ee51e5c4">
18       <InternalElement Name="InternalElement552" ID="c9be9315-5a25-4cf3-aea6-859db743636d"
           ↪>
19         <InternalElement Name="InternalElement553" ID="72ae2ad5-312e-42a1-b90f-
             ↪b208f1e9aadd" />
20       </InternalElement>
21     </InternalElement>
22     <InternalElement Name="InternalElement561" ID="98fe215b-ecb0-48b8-938b-084686663a9a" /
         ↪>
23   </InstanceHierarchy>
24 </CAEXFile>
```

Listing 6.4: Non unique IDs with external reference - referenced model

```
1  @Test
2  public void InternalElements_MultipleModelsDuplicateID() throws Exception
3  {
4     String modelPath = TestModelPath + "InstanceID\\
        ↪InternalElements_MultipleModelsDuplicateID\\";
5
6     addExpectedTestResult("Testcase.aml", "IsIDGUID", "", "", "InternalElement
        ↪InternalElement11 has a non unique ID: 9c99111a-937d-402e-8941-a390a0b5bb3e");
7     addExpectedTestResult("Testcase.aml", "IsIDGUID", "", "", "InternalElement
        ↪InternalElement12 has a non unique ID: 8e362e89-2c88-4bb6-827a-a26f48e29ada");
8     addExpectedTestResult("Testcase.aml", "IsIDGUID", "", "", "InternalElement
        ↪InternalElement13 has a non unique ID: 7a16a017-350d-478f-8f75-6d471c25a278");
9     addExpectedTestResult("Testcase.aml", "IsIDGUID", "", "", "InternalElement
        ↪InternalElement21 has a non unique ID: e56339ec-1ddf-41b0-8d70-139c606ce916");
10    addExpectedTestResult("Testcase.aml", "IsIDGUID", "", "", "InternalElement
        ↪InternalElement22 has a non unique ID: f5973236-6c1f-43d1-ac7f-cf8b34f33021");
11    addExpectedTestResult("Testcase.aml", "IsIDGUID", "", "", "InternalElement
        ↪InternalElement31 has a non unique ID: 3e402fb2-ca24-4016-8771-ae5f17a482a4");
12    addExpectedTestResult("Testcase.aml", "IsIDGUID", "", "", "InternalElement
        ↪InternalElement51 has a non unique ID: 1a51459a-539e-4197-b506-e304ee51e5c4");
13    addExpectedTestResult("Testcase.aml", "IsIDGUID", "", "", "InternalElement
        ↪InternalElement52 has a non unique ID: c9be9315-5a25-4cf3-aea6-859db743636d");
14    addExpectedTestResult("Testcase.aml", "IsIDGUID", "", "", "InternalElement
        ↪InternalElement53 has a non unique ID: 72ae2ad5-312e-42a1-b90f-b208f1e9aadd");
15    addExpectedTestResult("Testcase.aml", "IsIDGUID", "", "", "InternalElement
        ↪InternalElement61 has a non unique ID: 98fe215b-ecb0-48b8-938b-084686663a9a");
16
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
17     addExpectedTestResult("ReferencedLib.aml", "IsIDGUID", "", "", "InternalElement
       ↪InternalElement511 has a non unique ID: 9c99111a-937d-402e-8941-a390a0b5bb3e")
       ↪;
18     addExpectedTestResult("ReferencedLib.aml", "IsIDGUID", "", "", "InternalElement
       ↪InternalElement512 has a non unique ID: 8e362e89-2c88-4bb6-827a-a26f48e29ada")
       ↪;
19     addExpectedTestResult("ReferencedLib.aml", "IsIDGUID", "", "", "InternalElement
       ↪InternalElement513 has a non unique ID: 7a16a017-350d-478f-8f75-6d471c25a278")
       ↪;
20     addExpectedTestResult("ReferencedLib.aml", "IsIDGUID", "", "", "InternalElement
       ↪InternalElement521 has a non unique ID: e56339ec-1ddf-41b0-8d70-139c606ce916")
       ↪;
21     addExpectedTestResult("ReferencedLib.aml", "IsIDGUID", "", "", "InternalElement
       ↪InternalElement522 has a non unique ID: f5973236-6c1f-43d1-ac7f-cf8b34f33021")
       ↪;
22     addExpectedTestResult("ReferencedLib.aml", "IsIDGUID", "", "", "InternalElement
       ↪InternalElement531 has a non unique ID: 3e402fb2-ca24-4016-8771-ae5f17a482a4")
       ↪;
23     addExpectedTestResult("ReferencedLib.aml", "IsIDGUID", "", "", "InternalElement
       ↪InternalElement551 has a non unique ID: 1a51459a-539e-4197-b506-e304ee51e5c4")
       ↪;
24     addExpectedTestResult("ReferencedLib.aml", "IsIDGUID", "", "", "InternalElement
       ↪InternalElement552 has a non unique ID: c9be9315-5a25-4cf3-aea6-859db743636d")
       ↪;
25     addExpectedTestResult("ReferencedLib.aml", "IsIDGUID", "", "", "InternalElement
       ↪InternalElement553 has a non unique ID: 72ae2ad5-312e-42a1-b90f-b208f1e9aadd")
       ↪;
26     addExpectedTestResult("ReferencedLib.aml", "IsIDGUID", "", "", "InternalElement
       ↪InternalElement561 has a non unique ID: 98fe215b-ecb0-48b8-938b-084686663a9a")
       ↪;
27
28     Assertions.assertTrue(executeAndValidateTest(modelPath));
29 }
```

Listing 6.5: JUnit test InternalElements_MultipleModelsDuplicateID

6.4 Evaluation of *AML* Path Validation

To verify the correct functionality of the approach described in 5.2.3, the models in Table 6.3 were created.

	Test artefact	**Invalid models**	**Tested invalid**	**Valid Models**	**Tested valid**
Inheritance	**IC**	10	10	2	2
Inheritance	**RC**	9	9	2	2
Inheritance	**SUC**	9	9	2	2
Class Instantiation	**SUC**	8	8	1	1
Class Instantiation	**IE** mirror	2	2	1	1
Role-Instance Relation	**SuppRole**	1	1	1	1
Role-Instance Relation	**RoleReq**	1	1	1	1
Internal Links	**ExtI**	4	4	1	1

Table 6.3: *AML* Path Validation

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The evaluation of the path related constraints distinguishes based on the use-case the path is utilized for and the instance type.
As a path can point to various locations in the hierarchy (e.g., direct parent, class in same library in same *AML* file, class in different library in same *AML* file, class in different library in different *AML* file, etc.) various test cases representing different scenarios were created. As the functionality for validating the correctness of paths is universal applicable and used in all scenarios, the full variety of tests was only created for inheritance and class instantiation.

Listing 6.6 and Listing 6.7 show an exemplary testcase with a faulty inheritance path for an **IC**.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <CAEXFile FileName="Testcase.aml" SchemaVersion="2.15" xsi:noNamespaceSchemaLocation="
      ↪CAEX_ClassModel_V2.15.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3    <ExternalReference Alias="AMLBaseClasses" Path="AMLBaseclasses.aml" />
4    <InterfaceClassLib Name="InterfaceClassLib1">
5      <Version>1.0.0</Version>
6      <InterfaceClass Name="InterfaceClass1" RefBaseClassPath="
          ↪AMLBaseClasses@AutomationMLInterfaceClassLib/AutomationMLBaseInterface" />
7    </InterfaceClassLib>
8    <InterfaceClassLib Name="InterfaceClassLib2">
9      <Version>1.0.0</Version>
10     <InterfaceClass Name="InterfaceClass1" RefBaseClassPath="InterfaceClassLib1/
          ↪NotExistingIC" />
11   </InterfaceClassLib>
12 </CAEXFile>
```

Listing 6.6: Not valid path for inheritance

```
1  @Test
2  public void IC_NotExisting_SameFile() throws Exception
3  {
4    String modelPath = TestModelPath + "Inheritance\\IC_NotExisting_SameFile\\";
5
6    addExpectedTestResult("Testcase.aml", "IsRefBaseClassPathValid", "", "", "
          ↪RefBaseClassPath not valid: InterfaceClass NotExistingIC not found");
7    Assertions.assertTrue(executeAndValidateTest(modelPath));
8  }
```

Listing 6.7: JUnit test IC_NotExisting_SameFile

Another test scenario is depicted in the Listings 6.8, 6.9 and 6.10. These test models ensure, that the correct application of the mirroring concept does not report a false positive test result.

```
1  <CAEXFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" FileName="Testcase.aml"
      ↪SchemaVersion="2.15" xsi:noNamespaceSchemaLocation="./Source/CAEX_ClassModel_V2.15.
      ↪xsd">
2    <ExternalReference Alias="ReferencedLib" Path="ReferencedLib.aml" />
3    <InstanceHierarchy Name="InstanceHierarchy">
4      <Version>0</Version>
5      <InternalElement Name="InternalElement" ID="59e3e7b3-f23a-4907-94a4-d24de87e6a3b"
          ↪RefBaseSystemUnitPath="REFERENCEDGUID" />
6    </InstanceHierarchy>
7  </CAEXFile>
```

Listing 6.8: Valid mirroring example consisting of two *AML* models - root model

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
1 <?xml version="1.0" encoding="utf-8" standalone="no"?>
2 <CAEXFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" FileName="ReferencedLib.
    ↪aml" SchemaVersion="2.15" xsi:noNamespaceSchemaLocation="CAEX_ClassModel_V2.15.xsd">
3   <InstanceHierarchy Name="InstanceHierarchy">
4     <Version>0</Version>
5     <InternalElement Name="InternalElement" ID="REFERENCEDGUID"/>
6   </InstanceHierarchy>
7 </CAEXFile>
```

Listing 6.9: Valid mirroring example consisting of two *AML* models - referenced model

```
1 @Test
2 public void ValidIE_MirrorExisting_DifferentFile() throws Exception
3 {
4     String modelPath = TestModelPath + "ClassInstance\\
        ↪ValidIE_MirrorExisting_DifferentFile\\";
5     Assertions.assertTrue(executeAndValidateTest(modelPath));
6 }
```

Listing 6.10: JUnit test ValidIE_MirrorExisting_DifferentFile

6.5 Evaluation of *AML* Class Requirements Validation

To verify the correct functionality of the approach described in Section 5.2.4, the models in Table 6.4 were created.

	Test artefact	Invalid models	Tested invalid	Valid Models	Tested valid
Derived from BaseClass	**IC**	2	2	1	1
Derived from BaseClass	**RC**	1	1	1	1
Assigned to RoleClass	**SUC**	1	1	1	1

Table 6.4: *AML* Class Requirements Validation

As **IC**s and **RC**s require to be derived from an *AML* base class, the correct functionality was evaluated for every class separately.

An exemplary testcase for **IC**s is depicted in the Listings 6.11, 6.12 and 6.13.
The concrete scenario shows an *AML* model consisting of a root model and a referenced model. The root model's **IC** *InterfaceClass1* is derived from the **IC** *ReferencedICLChild* which is defined in the referenced model. As *ReferencedICLChild* is not derived from an *AML* base class, *ReferencedICLChild* and *InterfaceClass1* is required to be found as violation of the *AML* specification as depicted in Listing 6.13.

```
1 <CAEXFile FileName="Testcase.aml" SchemaVersion="2.15" xmlns:xsi="http://www.w3.org/2001/
    ↪XMLSchema-instance">
2   <ExternalReference Alias="AMLBaseClasses" Path="AMLBaseclasses.aml" />
3   <ExternalReference Alias="RefLib" Path="ReferencedLib.aml" />
4   <InterfaceClassLib Name="InterfaceClassLib">
5     <Version>0</Version>
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
6      <InterfaceClass Name="InterfaceClass" RefBaseClassPath="
           ↪AMLBaseClasses@AutomationMLInterfaceClassLib/AutomationMLBaseInterface/
           ↪PortConnector"/>
7    <InterfaceClass Name="InterfaceClass1" RefBaseClassPath="RefLib@ReferecedICL/
         ↪ReferencedICParent/ReferencedICLChild"/>
8    </InterfaceClassLib>
9  </CAEXFile>
```

Listing 6.11: **IC** without relation to *AML* base class - root model

```
1  <CAEXFile FileName="ReferencedLib.aml" SchemaVersion="2.15" xmlns:xsi="http://www.w3.org
       ↪/2001/XMLSchema-instance">
2    <InterfaceClassLib Name="ReferecedICL">
3      <Version>0</Version>
4      <InterfaceClass Name="ReferencedICParent" RefBaseClassPath="
           ↪AMLBaseClasses@AutomationMLInterfaceClassLib/AutomationMLBaseInterface/
           ↪PortConnector">
5      <InterfaceClass Name="ReferencedICLChild"/>
6    </InterfaceClass>
7    </InterfaceClassLib>
8  </CAEXFile>
```

Listing 6.12: **IC** without relation to *AML* base class- referenced model

```
1  @Test
2  public void IE_WithoutAMLBaseClass_MultipeFiles() throws Exception
3  {
4      String modelPath = TestModelPath + "InterfaceClass\\
           ↪IE_WithoutAMLBaseClass_MultipeFiles\\";
5
6      addExpectedTestResult("Testcase.aml", "IsDerivedFromAMLBaseClass", "", "", "Class not
           ↪derived from AMLInterfaceclass: InterfaceClass1");
7      addExpectedTestResult("ReferencedLib.aml", "IsDerivedFromAMLBaseClass", "", "", "Class
           ↪not derived from AMLInterfaceclass: ReferencedICLChild");
8
9      Assertions.assertTrue(executeAndValidateTest(modelPath));
10 }
```

Listing 6.13: JUnit test IE_WithoutAMLBaseClass_MultipeFiles

6.6 Evaluation of *AML* Instance Requirements Validation

The evaluation is split into the following categories described in 5.2.5.

Instance	Invalid models	Tested invalid	Valid Models	Tested valid
Instance requirements Order	4	4	1	1
Instance requirements Facet	4	4	1	1
Instance requirements Ports	10	10	1	1
Instance requirements Groups	1	1	1	1
Instance requirements PropertySet	3	3	1	1

Table 6.5: *AML* Instance Requirements Validation

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

As the validation of instance requirements is implemented per instance type, the evaluation follows the same pattern. For every instance type listed in Table 6.5 at least one model which violates its requirements was created.

Listings 6.14 and 6.15 depict a testcase for evaluating the correct implementation of the validation for the attribute *Direction* of an instance of the class *Order*. Listing 6.14 shows that the value of the attribute *Direction* contains the value *WrongValue* instead of the three possible values *In*, *Out*, or *InOut*. Listing 6.15 represents the corresponding JUnit test, verifying the correct validation.

```
1  <CAEXFile FileName="Testcase.aml" SchemaVersion="2.15" xmlns:xsi="http://www.w3.org/2001/
       ↪XMLSchema-instance">
2    <ExternalReference Alias="AMLBaseClasses" Path="AMLBaseclasses.aml" />
3    <InstanceHierarchy Name="InstanceHierarchy">
4      <Version>0</Version>
5      <InternalElement Name="InternalElement" ID="43601a05-aca0-4f46-bc22-1349efe72e9b">
6        <ExternalInterface Name="InterfaceClass" ID="bb8e14a1-3488-4135-9ce6-420847a88317"
             ↪RefBaseClassPath="InterfaceClassLib/InterfaceClass">
7          <Attribute Name="Direction" AttributeDataType="xs:string">
8            <Value>WrongValue</Value>
9          </Attribute>
10       </ExternalInterface>
11     </InternalElement>
12   </InstanceHierarchy>
13   <InterfaceClassLib Name="InterfaceClassLib">
14     <Version>0</Version>
15     <InterfaceClass Name="InterfaceClass" RefBaseClassPath="
           ↪AMLBaseClasses@AutomationMLInterfaceClassLib/AutomationMLBaseInterface/Order"
           ↪/>
16   </InterfaceClassLib>
17 </CAEXFile>
```

Listing 6.14: Order instance with wrong attribut value for direction

```
1 @Test
2 public void IE_OrderWithWrongDirectionAttribute() throws Exception
3 {
4    String modelPath = TestModelPath + "Order\\IE_OrderWithWrongDirectionAttribute\\";
5
6    addExpectedTestResult("Testcase.aml", "HasOrderInstanceDirectionCorrectValue", "", "",
         ↪"Instance of Type Order has invalid value Instance: InterfaceClass");
7    Assertions.assertTrue(executeAndValidateTest(modelPath));
8 }
```

Listing 6.15: JUnit test IE_OrderWithWrongDirectionAttribute

Listings 6.16 and 6.17 show one of the evaluations for *Facets*. This testcase evaluates the correct implementation responsible for detecting attributes in the *Facet* not defined in its parent object.

```
1  <CAEXFile FileName="Testcase.aml" SchemaVersion="2.15" xmlns:xsi="http://www.w3.org/2001/
       ↪XMLSchema-instance" xsi:noNamespaceSchemaLocation="CAEX_ClassModel_V2.15.xsd">
2    <ExternalReference Alias="AMLBaseClasses" Path="AMLBaseclasses.aml" />
3    <SystemUnitClassLib Name="SystemUnitClassLib">
4      <Version>0</Version>
5      <SystemUnitClass Name="Conveyor1" ID="bb706881-f9d0-4a66-9a40-4e2834712f31">
6        <Attribute Name="A" AttributeDataType="xs:string" />
7        <Attribute Name="B" AttributeDataType="xs:string" />
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
8      <ExternalInterface Name="X" ID="54b398d0-fd17-4a4e-a54c-02103fb65c9f" />
9      <ExternalInterface Name="Y" ID="9441e370-9fcc-40d4-9cd2-9bbd3cb556bf" />
10     <InternalElement Name="PLCFacet" ID="cd1f6943-4c05-45a6-8f1a-7aea5192de7e">
11       <Attribute Name="WRONG ATTRIBUTE"/>
12       <ExternalInterface Name="X" ID="a4a543c2-8112-4e6d-bc7b-a4bacc2d0c03" />
13       <RoleRequirements RefBaseRoleClassPath="AMLBaseClasses@AutomationMLBaseRoleClassLib/
             ↪AutomationMLBaseRole/Facet" />
14     </InternalElement>
15     <InternalElement Name="HMIFacet" ID="e437443c-d069-436e-9517-ead6d56ff5b3">
16       <Attribute Name="A" />
17       <Attribute Name="B" />
18       <ExternalInterface Name="Y" ID="baf8ba7a-b388-42d1-ac1b-ef5a07827031" />
19       <RoleRequirements RefBaseRoleClassPath="AMLBaseClasses@AutomationMLBaseRoleClassLib/
             ↪AutomationMLBaseRole/Facet" />
20     </InternalElement>
21     <SupportedRoleClass RefRoleClassPath="AMLBaseClasses@AutomationMLBaseRoleClassLib/
           ↪AutomationMLBaseRole" />
22   </SystemUnitClass>
23 </SystemUnitClassLib>
24 </CAEXFile>
```

Listing 6.16: Facet instance with wrong facet attribut

```
1 @Test
2 public void AMLFacetAttributInvalid() throws Exception
3 {
4    String modelPath = TestModelPath + "Facet\\AMLFacetAttributInvalid\\";
5
6    addExpectedTestResult("Testcase.aml", "AreFacetAttributesValid", "", "", "Facet
         ↪Attributes not defined: WRONG ATTRIBUTE ");
7    Assertions.assertTrue(executeAndValidateTest(modelPath));
8 }
```

Listing 6.17: JUnit test AMLFacetAttributInvalid

6.7 Evaluation of *AML* Meta information Validation

The evaluation approach of the implementation described in Section 5.2.6 is depicted in the following table.

Instance	Invalid models	Tested invalid	Valid Models	Tested valid
Meta information	12	12	1	1

Table 6.6: *AML* Meta information Validation

Listings 6.18 and 6.19 depict one of the evaluations relating to *AML* metadata. This testcase evaluates the correct implementation responsible checking of the existence of the element *WriterName*.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <CAEXFile FileName="Testcase.aml" SchemaVersion="2.15" xsi:noNamespaceSchemaLocation="
      ↪CAEX_ClassModel_V2.15.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3   <AdditionalInformation AutomationMLVersion="2.0"/>
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
4      <AdditionalInformation>
5          <WriterHeader>
6              <!--<WriterName>AutomationML Editor</WriterName> -->
7              <WriterID>916578CA-FE0D-474E-A4FC-9E1719892369</WriterID>
8              <WriterVendor>AutomationML e.V.</WriterVendor>
9              <WriterVendorURL>www.AutomationML.org</WriterVendorURL>
10             <WriterVersion>4.3.11.0</WriterVersion>
11             <WriterRelease>4.3.11.0</WriterRelease>
12             <LastWritingDateTime>2016-04-11T23:57:35.5278227+02:00</LastWritingDateTime>
13             <WriterProjectTitle>unspecified</WriterProjectTitle>
14             <WriterProjectID>unspecified</WriterProjectID>
15         </WriterHeader>
16     </AdditionalInformation>
17 </CAEXFile>
```

Listing 6.18: WriterHeader without WriterName

```
1 @Test
2 public void NoWriterName() throws Exception
3 {
4     String modelPath = TestModelPath + "NoWriterName\\";
5
6     addExpectedTestResult("Testcase.aml", "ValidateWriterHeader", "", "", "WriterName not
        ↪defined");
7
8     Assertions.assertTrue(executeAndValidateTest(modelPath));
9 }
```

Listing 6.19: JUnit test NoWriterName

6.8 Evaluation Result

For evaluation purposes for every constraint implemented in Chapter 5 at least one *AML* model which contradicts the restrictions and extensions specified in [24] was created. The implemented evaluation provides with its *152 JUnit tests* a 100% constraint coverage, what implies that every implemented *EVL* constraint successfully reports for at least one model the desired validation result.

Anyhow, based on the fact that the models were created manually, some threats to validity remain. Although the *AML* models were created with the intention to evaluate as much as possible of the *EVL* constraints execution paths, a full path coverage cannot be guaranteed. As *AML* provides a wide range of possibilities for modeling, further evaluation steps should utilize real world *AML* models to ensure that all possible *AML* constellations are covered.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 7

Conclusion and Future work

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

7.1 Conclusion

Whereas *AML* was invented to provide a data exchange format which enables standardization of syntax and semantics level [24, 34, 14] available tools to validate its correctness are vague. That involves the risk that incorrect models prohibit the original fundamental idea of *AML* to enable an efficient data exchange. A flawless and unambiguous data exchange is especially for construction of automation systems of industrial plants due to the involvement of multiple engineering disciplines essential [8, 18].
As *AML* utilizes a wide range of techniques (e.g., inheritance, class instantiation, multiple role classes, splitting of models into multiple files, etc.) an in depth validation of models is required to ensure the models consistency.

This thesis implemented based on the *AML* specification a tool which enables the in depth validation of *AML* models.
The validation is implemented as a sequential process and checks the following model aspects:

1. Versioning
2. Object identification
3. Paths
4. Class requirements
5. Instance requirements
6. Meta information

To ensure the implementations correctness a JUnit project was setup which follows the validation's modular approach. Based on various correct and faulty models the correct functionality of the implementation was shown.

7.2 Future Work

As during the implementation of this thesis a new version of *CAEX* and *AML* was released, the implementation should be updated based on the new specification. As the *CAEX* metamodel is already available at https://github.com/amlModeling/caex-workbench, the next step is to update the *AML* specific implementation in *EVL* based on the new specification. In addition, the evaluation project implemented as JUnit project should be updated based on the adaptions in the *AML* specification.
Besides that, the evaluation method applied in this thesis should be extended from artificial models created specific for the validation of the implementation to *AML* models originating from real world projects.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

APPENDIX A

Appendix

A.1 AML Base Libraries

Section A.1.1 and A.1.2 depict the *XML* representations of the *AML* base classes for **ICs** and **RCs**. As mentioned in Section 2.4, user created classes must be associated directly or indirectly to one of the presented base classes. The depicted listings are part of [24].

A.1.1 AutomationMLInterfaceClassLib

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <CAEXFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
     ↪noNamespaceSchemaLocation="CAEX_ClassModel_V2.15.xsd" FileName="
     ↪AutomationMLInterfaceClassLib.aml" SchemaVersion="2.15">
3    <AdditionalInformation AutomationMLVersion="2.0" />
4    <AdditionalInformation>
5      <WriterHeader>
6        <WriterName>IEC SC65E WG 9</WriterName>
7        <WriterID>IEC SC65E WG 9</WriterID>
8        <WriterVendor>IEC</WriterVendor>
9        <WriterVendorURL>www.iec.ch</WriterVendorURL>
10       <WriterVersion>1.0</WriterVersion>
11       <WriterRelease>1.0.0</WriterRelease>
12       <LastWritingDateTime>2013-03-01</LastWritingDateTime>
13       <WriterProjectTitle>Automation Markup Language Standard Libraries</WriterProjectTitle>
14       <WriterProjectID>Automation Markup Language Standard Libraries</WriterProjectID>
15     </WriterHeader>
16   </AdditionalInformation>
17   <InterfaceClassLib Name="AutomationMLInterfaceClassLib">
18     <Description>Standard Automation Markup Language Interface Class Library</Description>
19     <Version>2.2.0</Version>
20     <InterfaceClass Name="AutomationMLBaseInterface">
21       <InterfaceClass Name="Order" RefBaseClassPath="AutomationMLBaseInterface">
22         <Attribute Name="Direction" AttributeDataType="xs:string" />
23       </InterfaceClass>
24       <InterfaceClass Name="PortConnector" RefBaseClassPath="AutomationMLBaseInterface" />
25       <InterfaceClass Name="InterlockingConnector" RefBaseClassPath="
           ↪AutomationMLBaseInterface" />
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
26      <InterfaceClass Name="PPRConnector" RefBaseClassPath="AutomationMLBaseInterface" />
27      <InterfaceClass Name="ExternalDataConnector" RefBaseClassPath="
            ↪AutomationMLBaseInterface">
28        <Attribute Name="refURI" AttributeDataType="xs:anyURI" />
29        <InterfaceClass Name="COLLADAInterface" RefBaseClassPath="ExternalDataConnector" />
30        <InterfaceClass Name="PLCopenXMLInterface" RefBaseClassPath="ExternalDataConnector" />
31      </InterfaceClass>
32      <InterfaceClass Name="Communication" RefBaseClassPath="AutomationMLBaseInterface">
33        <InterfaceClass Name="SignalInterface" RefBaseClassPath="Communication" />
34        </InterfaceClass>
35      </InterfaceClass>
36    </InterfaceClassLib>
37  </CAEXFile>
```

Listing A.1: *AML* Interface class library

A.1.2 AutomationMLBaseRoleClassLib

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <CAEXFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
       ↪noNamespaceSchemaLocation="CAEX_ClassModel_V2.15.xsd" FileName="
       ↪AutomationMLBaseRoleClassLib.aml" SchemaVersion="2.15">
3    <AdditionalInformation AutomationMLVersion="2.0" />
4    <AdditionalInformation>
5      <WriterHeader>
6        <WriterName>AutomationML e.V.</WriterName>
7        <WriterID>AutomationML e.V.</WriterID>
8        <WriterVendor>AutomationML e.V.</WriterVendor>
9        <WriterVendorURL>www.AutomationML.org</WriterVendorURL>
10       <WriterVersion>1.0</WriterVersion>
11       <WriterRelease>1.0.0</WriterRelease>
12       <LastWritingDateTime>2012-02-17</LastWritingDateTime>
13       <WriterProjectTitle>AutomationML Standard Libraries</WriterProjectTitle>
14       <WriterProjectID>AutomationML Standard Libraries</WriterProjectID>
15     </WriterHeader>
16   </AdditionalInformation>
17   <ExternalReference Path="../InterfaceClass Libraries/AutomationMLInterfaceClassLib.aml"
         ↪Alias="AutomationMLInterfaceClassLib" />
18   <RoleClassLib Name="AutomationMLBaseRoleClassLib">
19     <Description>AutomationML base role library</Description>
20     <Version>2.1.2</Version>
21     <RoleClass Name="AutomationMLBaseRole">
22       <RoleClass Name="Group" RefBaseClassPath="AutomationMLBaseRole">
23         <Attribute Name="AssociatedFacet" AttributeDataType="xs:string" />
24       </RoleClass>
25       <RoleClass Name="Facet" RefBaseClassPath="AutomationMLBaseRole" />
26       <RoleClass Name="Port" RefBaseClassPath="AutomationMLBaseRole">
27         <Attribute Name="Direction" AttributeDataType="xs:string" />
28         <Attribute Name="Cardinality">
29           <Attribute Name="MinOccur" AttributeDataType="xs:unsignedInt" />
30           <Attribute Name="MaxOccur" AttributeDataType="xs:unsignedInt" />
31         </Attribute>
32         <Attribute Name="Category" AttributeDataType="xs:string" />
33         <ExternalInterface Name="ConnectionPoint" ID="1c6a2bb9-8f93-4394-8fae-ef0e0074716a"
               ↪RefBaseClassPath="
               ↪AutomationMLInterfaceClassLib@AutomationMLInterfaceClassLib/
               ↪AutomationMLBaseInterface/PortConnector" />
34       </RoleClass>
35       <RoleClass Name="Resource" RefBaseClassPath="AutomationMLBaseRole" />
36       <RoleClass Name="Product" RefBaseClassPath="AutomationMLBaseRole" />
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
37       <RoleClass Name="Process" RefBaseClassPath="AutomationMLBaseRole" />
38       <RoleClass Name="Structure" RefBaseClassPath="AutomationMLBaseRole">
39         <RoleClass Name="ProductStructure" RefBaseClassPath="Structure" />
40         <RoleClass Name="ProcessStructure" RefBaseClassPath="Structure" />
41         <RoleClass Name="ResourceStructure" RefBaseClassPath="Structure">
42           <RoleClass Name="Cell" RefBaseClassPath="ResourceStructure" />
43           <RoleClass Name="MainGroup" RefBaseClassPath="ResourceStructure" />
44           <RoleClass Name="FunctionGroup" RefBaseClassPath="ResourceStructure" />
45           <RoleClass Name="SubFunctionGroup" RefBaseClassPath="ResourceStructure" />
46           <RoleClass Name="MechatronicAssembly" RefBaseClassPath="ResourceStructure" />
47           <RoleClass Name="MechanicalAssembly" RefBaseClassPath="ResourceStructure" />
48           <RoleClass Name="MechanicalPart" RefBaseClassPath="ResourceStructure" />
49           <RoleClass Name="Device" RefBaseClassPath="ResourceStructure" />
50         </RoleClass>
51       </RoleClass>
52       <RoleClass Name="PropertySet" RefBaseClassPath="AutomationMLBaseRole" />
53     </RoleClass>
54   </RoleClassLib>
55 </CAEXFile>
```

Listing A.2: *AML* Role class library

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

1.1 Plant planning process . . . 1
1.2 Standard IEC CAEX vs AML CAEX . . . 2

2.1 Planned vs real project progression [6] . . . 5
2.2 Planned vs real project progression with handover efforts [6] . . . 6
2.3 Planned vs real project progression with data loss[6] . . . 6
2.4 Pure **IH** . . . 8
2.5 Pure **RClib** . . . 9
2.6 **IH** with **RClib** . . . 9
2.7 Pure **IClib** . . . 10
2.8 **IH** with **RClib** and **IClib** . . . 10
2.9 Pure **SUClib** . . . 11
2.10 **IH** with **RClib**, **IClib** and **SUClib** . . . 12
2.11 **IH** external **SUClib** . . . 13
2.12 Paths . . . 16
2.13 Object Parent-Child Relationship . . . 17
2.14 Class Parent-Child Relationship . . . 18
2.15 Inheritance . . . 19
2.16 Class instantiation . . . 20
2.17 Multiple role classes . . . 21
2.18 Class-Class Relation . . . 22
2.19 *AML* Base Role Classes [24] . . . 23
2.20 Connecting multiple interfaces with Port . . . 25
2.21 Facet Example . . . 27
2.22 Group Example . . . 28
2.23 PropertySet Example . . . 30
2.24 *AML* Base Interface Classes [24] . . . 32

3.1 *AML* Base Attribute Types [25] . . . 39
3.2 PPR Concept [24] . . . 41

4.1 MDSE process [9] . . . 44
4.2 4 Layer Metamodeling Stack [9] . . . 44
4.3 Simplified Ecore Metamodel [35] . . . 45

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.4 Excerpt of *CAEX* metamodel [36] 46
4.5 AML Model created with EMF editor 47
4.6 Integration of Java, XML and UML [35] 47
4.7 OCL Example 48

5.1 Validation framework 52

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Tables

2.1 AML metadata information [24] 14
2.2 AML path examples [24] 15

5.1 Basic software metrics of the validation framework 65

6.1 *AML* Document Version Validation 68
6.2 *AML* Object Identification Validation 69
6.3 *AML* Path Validation 71
6.4 *AML* Class Requirements Validation 73
6.5 *AML* Instance Requirements Validation 74
6.6 *AML* Meta information Validation 76

Listings

2.1 *AML* document versioning 11
2.2 *AML* library versioning 12
2.3 *AML* meta information 14
2.4 *AML* object identification 15
2.5 *AML* class identification 15
2.6 *AML* object parent-child relationship 17
2.7 *AML* class parent-child relationship 18
2.8 AML Listing Inheritance 18
2.9 AML Listing Internal Links 20
2.10 AML Listing Port 24
2.11 AML Listing Facet 26
2.12 AML Listing Group 29
2.13 AML Listing PropertySet 31

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.14 Example instance Order 31
2.15 Example instance COLLADAInterface 33
3.1 Simplified **RC** assignment 38
3.2 InternalElement with Metainformation 38
3.3 RoleClass decribing *ECLASS* attributes 40
3.4 RoleClass depicting a robotic arm 40
4.1 Exemplary EVL listing 49
5.1 Validation of *AML* version 53
5.2 Validation of **IClib** uniqueness 55
5.3 Validation of ID uniqueness 56
5.4 Validation of Baseclass path 58
5.5 Validation if **IC** is derived from AML baseclass 59
5.6 Validation if Direction of Order Instance has correct value 61
5.7 Validation if Port has ExternalInterfaces 62
5.8 Validation of WriterHeader objects 63
6.1 Invalid *AML* version 68
6.2 JUnit test WrongAutomationMLVersion 68
6.3 Non unique IDs with external reference - root model 69
6.4 Non unique IDs with external reference - referenced model 69
6.5 JUnit test InternalElements_MultipleModelsDuplicateID 70
6.6 Not valid path for inheritance 72
6.7 JUnit test IC_NotExisting_SameFile 72
6.8 Valid mirroring example consisting of two *AML* models - root model . 72
6.9 Valid mirroring example consisting of two *AML* models - referenced model 73
6.10 JUnit test ValidIE_MirrorExisting_DifferentFile 73
6.11 **IC** without relation to *AML* base class - root model 73
6.12 **IC** without relation to *AML* base class- referenced model 74
6.13 JUnit test IE_WithoutAMLBaseClass_MultipeFiles 74
6.14 Order instance with wrong attribut value for direction 75
6.15 JUnit test IE_OrderWithWrongDirectionAttribute 75
6.16 Facet instance with wrong facet attribut 75
6.17 JUnit test AMLFacetAttributInvalid 76
6.18 WriterHeader without WriterName 76
6.19 JUnit test NoWriterName 77
A.1 *AML* Interface class library 81
A.2 *AML* Role class library 82

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Bibliography

[1] AutomationML Editor V.6.1.4. https://www.automationml.org/wp-content/uploads/2023/01/AMLEditor_6.1.4.zip. Accessed: 2023-02-07.

[2] Epsilon platform. https://www.eclipse.org/epsilon/. Accessed: 2023-02-07.

[3] Epsilon Validation Language. https://www.eclipse.org/epsilon/doc/evl/. Accessed: 2023-02-07.

[4] Lisa Abele, Christoph Legat, Stephan Grimm, and Andreas W. Müller. Ontology-based validation of plant models. In *11th IEEE International Conference on Industrial Informatics*, 2013.

[5] Sofia Ananieva, Erik Burger, and Christian Stier. Model-Driven Consistency Preservation in AutomationML. In *IEEE 14th International Conference on Automation Science and Engineering*, 2018.

[6] Mike Barth, Rainer Drath, Alexander Fay, Florian Zimmer, and Karin Eckert. Evaluation of the openness of automation tools for interoperability in engineering tool chains. In *Proceedings of IEEE 17th International Conference on Emerging Technologies and Factory Automation*, 2012.

[7] Luca Berardinelli, Stefan Biffl, Emanuel Maetzler, Tanja Mayerhofer, and Manuel Wimmer. Model-based co-evolution of production systems and their libraries with AutomationML. In *Proceedings of IEEE 20th International Conference on Emerging Technologies and Factory Automation*, 2015.

[8] Stefan Biffl, Luca Berardinelli, Emanuel Mätzler, Manuel Wimmer, Arndt Lueder, and Nicole Schmidt. Model-Based Risk Assessment in Multi-disciplinary Systems Engineering. In *41st Euromicro Conference on Software Engineering and Advanced Applications*, 2015.

[9] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice, Second Edition*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2017.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[10] Jean Bézivin. In Search of a Basic Principle for Model Driven Engineering. *UPGRADE, European Journal for the Informatics Professional*, 5(2), 2004.

[11] Jordi Cabot and Martin Gogolla. Object Constraint Language (OCL): A Definitive Guide. In *Formal Methods for Model-Driven Engineering, Lecture Notes in Computer Science*. Springer, 2012.

[12] World Wide Web Consortium. OWL 2 Web Ontology Language – Version 11 December 2012. `http://www.w3.org/TR/2012/REC-owl2-overview-20121211/`.

[13] World Wide Web Consortium. SPARQL Query Language for RDF – Version 15 January 2008. `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`.

[14] Rainer Draht. *Datenaustausch in der Anlagenplanung mit AutomationML: Integration von CAEX, PLCopen XML und COLLADA*. Springer, 2010.

[15] Rainer Drath. *AutomationML – A Practical Guide*. De Gruyter, 2021.

[16] Rainer Drath and Mike Barth. Concept for managing multiple semantics with AutomationML — Maturity level concept of semantic standardization. In *Proceedings of IEEE 17th International Conference on Emerging Technologies and Factory Automation*, 2012.

[17] Rainer Drath, Alexander Fay, and Mike Barth. Interoperabilität von Engineering-Werkzeugen. *Automatisierungstechnik*, 59(7), 2011.

[18] Rainer Drath, Arndt Lüder, Joern Peschke, and Lorenz Hundt. AutomationML - the glue for seamless Automation Engineering. In *Proceedings of IEEE 13th International Conference on Emerging Technologies and Factory Automation*, 2008.

[19] Sebastian Faltinski, Oliver Niggemann, Natalia Moriz, and André Mankowski. AutomationML: From data exchange to system planning and simulation. In *IEEE International Conference on Industrial Technology*, 2012.

[20] Olaf Graeser, Lorenz Hundt, Michael John, Alexander Krahner, Gerald Lobermeier, Arndt Lüder, Stefan Mülhens, and Josef Schmelter. AutomationML and ECLASS integration. `https://www.automationml.org/wp-content/uploads/2021/11/WP_AutomationML_and_ECLASS_integration_V2.0.pdf`, 2021. Accessed: 2023-02-07.

[21] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 1993.

[22] Elliotte Harold and Scott Means. *XML in a nutshell*. O'Reilly, 2002.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[23] IEC. *IEC 62424:2008: Darstellung von Aufgaben der Prozessleittechnik – Fließbilder und Datenaustausch zwischen EDV-Werkzeugen zur Fließbilderstellung und CAE-Systemen*. IEC, 2008.

[24] IEC. *IEC 62714-1:2014: Engineering data exchange format for use in industrial automation systems engineering - Automation markup language - Part 1: Architecture and general requirements*. IEC, 2014.

[25] IEC. *IEC 62714-1:2018 : Engineering data exchange format for use in industrial automation systems engineering - Automation Markup Language - Part 1: Architecture and general requirements*. IEC, 2018.

[26] Anneke Kleppe. The Field of Software Language Engineering. In *Software Language Engineering, Lecture Notes in Computer Science*. Springer, 2009.

[27] Dimitrios Kolovos, Richard Paige, and Fiona Polack. The Epsilon Object Language (EOL). In *Model Driven Architecture - Foundations and Applications, Lecture Notes in Computer Science*. Springer, 2006.

[28] Dimitrios Kolovos, Richard Paige, and Fiona Polack. On the Evolution of OCL for Capturing Structural Constraints in Modelling Languages. In *Rigouros Methods for Software Construction and Analysis, Lecture Notes in Computer Science*. Springer, 2009.

[29] Object Management Group. OMG Object Constraint Language (OCL) – Version 2.4. https://www.omg.org/spec/OCL/2.4, 2014.

[30] Object Management Group. OMG Unified Modeling Language – Version 2.5.1. https://www.omg.org/spec/UML/2.5.1, 2017.

[31] Marta Sabou, Fajar Ekaputra, Olga Kovalenko, and Stefan Biffl. Supporting the engineering of cyber-physical production systems with the AutomationML analyzer. In *1st International Workshop on Cyber-Physical Production Systems*, 2016.

[32] Miriam Schleipen. A concept for conformance testing of AutomationML models by means of formal proof using OCL. In *Proceedings of IEEE 15th International Conference on Emerging Technologies and Factory Automation*, 2010.

[33] Miriam Schleipen and Rainer Drath. Three-view-concept for Modeling Process or Manufacturing Plants with AutomationML. In *Proceedings of IEEE 14th International Conference on Emerging Technologies and Factory Automation*, 2009.

[34] N. Schmidt and A. Lüder. AutomationML in a Nutshell. `https://www.automationml.org/wp-content/uploads/2021/06/AutomationML-in-a-Nutshell_151104.pdf`, 2015. Accessed: 2023-02-07.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[35] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0 (2nd Edition)*. Addison Wesley, 2009.

[36] Manuel Wimmer and Tanja Mayerhofer. ECore CAEX Metamodel . https://github.com/amlModeling/caex-workbench/. Accessed: 2023-02-10.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub