Karlsruhe Institute of Technology

Intelligent Process
Automation and
Robotics Lab

Methods for Semantic Interoperability in AutomationML-based Engineering

Ph.D. Thesis

of

Yingbing Hua

At the KIT Department of Informatics
Institute for Anthropomatics and Robotics (IAR) -
Intelligent Process Automation and Robotics Lab (IPR)

First reviewer:	Prof. Dr.-Ing. habil. Björn Hein
Second reviewer:	Prof. Dr. York Sure-Vetter

December 31, 2020

Institute for Anthropomatics and Robotics (IAR) -
Intelligent Process Automation and Robotics Lab (IPR)
KIT Department of Informatics
Karlsruhe Institute of Technology
Engler-Bunte-Ring 8
76131 Karlsruhe

Yingbing Hua
Schänzlestr. 14
79104 Freiburg im Breisgau
yingbing.hua@kit.edu

Methods for Semantic Interoperability in AutomationML-based Engineering

Zur Erlangung des akademischen Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)
genehmigte

DISSERTATION

von

Yingbing Hua

Tag der mündlichen Prüfung: 04.12.2020

1. Referent: Prof. Dr.-Ing. habil. Björn Hein

2. Referent: Prof. Dr. York Sure-Vetter

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, December 31, 2020

. . . .

(**Yingbing Hua**)

Acknowledgements

Writing this Ph.D. thesis was one of the most challenging things in my life. Without help from many sides, I would not have been able to complete this work.

I would like to give my sincere thank to my supervisor, Prof. Björn Hein, who allowed me to follow my own research direction, which is, in fact, quite perpendicular to the main research fields of our research group. Back in the year 2014, I started with the research project ReApp and have been fascinated since then by the ideas of Industry 4.0 and the modern Semantic Web Technologies. Prof. Björn Hein encouraged me to pursue my research ideas and supported me by providing financial resources and academic advice to my work.

This thesis focuses on the challenge of semantic interoperability in AutomationML-based engineering. During the last 6 years, I have been regularly attending the workshops and conferences of the AutomationML association and met many excellent researchers and practitioners who develop and utilize AutomationML in the context of Industry 4.0. I am very thankful for the discussions with them, which gave me feedback on my research ideas and insights into requirements and solutions in real engineering processes. Specifically, I want to thank Prof. Rainer Drath from Hochschule Pforzheim, Prof. Arndt Lüder from OVGU Magdeburg, Prof. Alexander Fay from HSU Hamburg, Joachim Burlein from Daimler AG, and Michael John from Siemens AG.

I also would like to thank all my colleagues at the Institute for Anthropomatics and Robotics - Intelligent Process Automation and Robotics Lab (IAR-IPR). I am particularly grateful for the exciting discussions with David Puljiz, Michael Mende, and Zunchao Zhen, which have inspired me with new research ideas.

During the final thesis preparation, I received fruitful inputs from the professors at the Informatics faculty of KIT, including Prof. Veit Hagenmeyer, Prof. Jürgen Beyerer, and Prof. Anne Koziolek. Specifically, I want to thank my second supervisor, Prof. York Sure-Vetter, who gave me in-depth discussions about the scientific aspects of the thesis and provided professional suggestions to improve the thesis's quality.

Finally, I want to thank my wife, Wei, and my daughter Alice for bringing so much joy to my life, so that I could overcome the hard times during my research.

Abstract

Methods for Semantic Interoperability in AutomationML-based Engineering

Industrial engineering is an interdisciplinary activity that involves human experts from various technical backgrounds working with different engineering tools. In the era of digitization, the engineering process generates a vast amount of data. To store and exchange such data, dedicated international standards are developed, including the XML-based data format AutomationML (AML). While AML provides a harmonized syntax among engineering tools, the semantics of engineering data remains highly heterogeneous. More specifically, the AML models of the same domain or entity can vary dramatically among different tools that give rise to the so-called semantic interoperability problem. In practice, manual implementation is often required for the correct data interpretation, which is usually limited in reusability.

Efforts have been made for tackling the semantic interoperability problem. One mainstream research direction has been focused on the semantic lifting of engineering data using Semantic Web technologies. However, current results in this field lack the study of building complex domain knowledge that requires a profound understanding of the domain and sufficient skills in ontology building. This thesis contributes to this research field in two aspects. First, machine learning algorithms are developed for deriving complex ontological concepts from engineering data. The induced concepts encode the relations between primitive ones and bridge the semantic gap between engineering tools. Second, to involve domain experts more tightly into the process of ontology building, this thesis proposes the AML concept model (ACM) for representing ontological concepts in a native AML syntax, i.e., providing an AML-frontend for the formal ontological semantics. ACM supports the bidirectional information flow between the user and the learner, based on which the interactive machine learning framework AMLLEARNER is developed.

Another rapidly growing research field devotes to develop methods and systems for facilitating data access and exchange based on database theories and techniques. In particular, the so-called Query By Example (QBE) allows the user to construct queries using data examples. This thesis adopts the idea of QBE in AML-based engineering by introducing the AML Query Template (AQT). The design of AQT has been focused on a native AML syntax, which allows constructing queries with conventional AML tools. This thesis studies the theoretical foundation of AQT and presents algorithms for the automated generation of query programs. Comprehensive requirement analysis shows that the proposed approach can solve the problem of semantic interoperability in AutomationML-based engineering to a great extent.

Keywords: *Interoperability, AutomationML, Concept Learning, Query By Example*

Contents

1. Introduction

Industrial engineering (or engineering in short) is an interdisciplinary task that comprises "*all activities required to design, test, and commission complex manufacturing lines or process plants*" [1]. One major challenge of engineering is the interoperability between the engineering tools used by human experts from various technical backgrounds in different activities [2]. The VDI/VDE guidelines 3695 [3] defines four interoperability levels[1] that could be achieved in engineering. Suppose a simple engineering environment with two tools, that is, a **source** tool that generates data and a **target** tool that consumes data, one could differentiate between the following cases:

Level 1 (electronic data) The source tool provides data electronically, such as PDF documents or images, from which the necessary information for the target tool can be extracted. However, because neither the syntax nor the semantics of the data is formally defined for machine processing, a human expert is usually responsible for the manual data extraction and transformation.

Level 2 (shared meta-model) In the next level, the source tool exports data into a machine-readable format, e.g., Excel spreadsheets or XML files. For the correct usage on the target side, a shared meta-model needs to be established for both tools. First, the syntax for data representation and storage needs to be defined, which could be a stable structure of the Excel tables or a schema of the XML files. Second, the semantics of the meta-model must be agreed upon the tools, i.e., the intended meaning of each table column or the XML schema elements. In this level, data exchange or synchronization can be fully automated because the same meta-model is used by both tools.

Level 3 (importer-based) Similar to level 2, the source data is exported and ready for machine processing. Moreover, the target tool implements a data importer that manifests the logic of data interpretation and provides functionalities to load or transform the source data into the target tool. In contrast to level 2, a comprehensive agreement of the meta-model is not necessary.

Level 4 (integrated tool-suite) In the highest level of interoperability, tools may use their proprietary models, and no human intervention is required for ensuring consistency. According to Drath et al. [2], the 4th level is only realized in integrated tool-suites. Indeed, level 4 shares the same semantic foundation with level 3, and dedicated software modules are provided for achieving interoperability inside the tool-suite.

For a heterogeneous engineering landscape consisting of tools from various software providers, integrated tool-suites (level 4) are less flexible, while a common meta-model (level 2) is hard to achieve [4]. Therefore, research on interoperability mainly focused on level 3. In particular, standardization efforts have been undertaken worldwide.

[1]In the original formulation, they are called the *goal status* of interoperability.

One of the most remarkable progress during the last decade was the development of the XML-based data format AutomationML[2] (AML, IEC 62714[3]). The motivation of AML was to reduce the diversity of data formats employed in different engineering tasks and environments, such as PDF documents, images, Excel tables, and proprietary XML formats. Therefore, AML is designed as an intermediate layer between the engineering tools. The core of AML is a neutral object-oriented meta-model that satisfies the modeling needs of common engineering tasks. Driving by leading companies in the manufacturing industry[4], AML has become a de-facto standard for data modeling and exchange in engineering. For example, the company Daimler AG has introduced the *integra Engineering Studio* for company-wide data integration based on a common AML model of the production lines [5]. The company ABB HVDC (High-Voltage Direct Current) Sweden applied AML as a middleware to bridge various tools and platforms that are used to engineer HVDC systems [1]. Ongoing work in other domains includes the digitization of System Control Diagrams (SCDs) by the Norwegian company Equinor in the Oil&Gas industry [6] and exchanging field-bus topology between the German company Balluff and the Japanese company Mitsubishi Electric [7]. Numerous results from the research community can also be observed. For example, a query with the keyword "AutomationML" in Google Scholar[5] retrieves more than 1800 articles, among which more than 1100 ones are published since the year 2015. The advances of AML also caught the attention of Platform Industry 4.0[6]. For example, the Reference Architecture Model Industrie 4.0 [8] and the Implementation Strategy Industrie 4.0 [9] consider AML as an approach for implementing a functional and information layer in the end-to-end engineering, while the German Standardization Roadmap Industrie 4.0 [10] and the Details of the Asset Administration Shell [11] suggest using AML for the description of automation systems and components.

In terms of the interoperability levels described above, AML provides a universal XML schema as the underlying syntax for storing engineering data and proposes guidelines for modeling engineering artifacts using this syntax. Essential parts of the guidelines have become formal IEC specifications, while new features are first released as Best-Practice Recommendations (BPR) or Application Recommendations (AR)[7]. Furthermore, AML is characterized by its **mixed model** principle, which allows the coexistence of both neutral and proprietary data in one document [12]. The motivation behind the mixed model principle is the so-called "semantic standardization deadlock," which claims that a one-world model that satisfies the needs of all engineering disciplines and stakeholders would not appear in the near future [12]. Consequently, data exchange with AML follows the export-import methodology, as described in the interoperability level 3.

Fig. 1.1 illustrates the AML-based data exchange. One the one hand, neutral data of the source tool is equipped with the standardized semantics and can be unambiguously interpreted by the importer of the target tool (Source-Neutral and Neutral-Target). On the other hand, private data of the source tool is wrapped into AML objects (Source-Private) without commonly agreed semantics and cannot be understood by the importer. Therefore, Private-Target mappings need to be established. These mappings bridge the

[2]https://www.automationml.org/
[3]https://webstore.iec.ch/publication/32339
[4]https://www.automationml.org/o.red.c/mitglieder.html
[5]https://scholar.google.de/scholar?q=automationml&hl=de&as_sdt=0,5
[6]www.plattform-i40.de
[7]https://www.automationml.org/o.red.c/publications.html

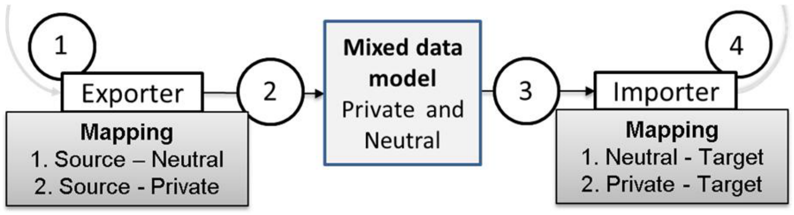


Figure 1.1.: Data exchange based on a mixed model [4].

semantic gap between the engineering tools and empower the tools with the so-called **semantic interoperability** [13]. However, in practice, the potential complexity of these mappings could lead to considerable implementation efforts that limit the applicability of AML in a wider scope [1]. One evidence of this issue is the continuously increasing endeavor of the AutomationML e.V. towards several comprehensive sub-standards (or dialects). For example, the component working group aims at providing a general meta-model for describing automation components [14], and the recently established toolchain working group focuses on a meta-model for virtual commissioning.

1.1. Motivation

This thesis aims at providing methods to improve the semantic interoperability in AML-based engineering, under the assumption that the participants in the engineering toolchain maintain a private subset of the mixed model. Formally:

Definition 1.1 (semantic interoperability in AML-based engineering)**.** Let A and B be two engineering tools and suppose that B needs data from A, i.e., A is the source tool and B is the target tool. Assume that both A and B have some private concepts. Given an AML document S from A, semantic interoperability from A to B is the ability that

1. allows B to retrieve raw data (i.e., XML elements or values) from S that semantically belongs to a private concept of B;
2. allows B to construct an AML document T using the raw data extracted from S. $\square$

In Definition 1.1, case 1 refers to the scenario of data access where the required data can be semantically described with a private concept of B, and case 2 refers to data exchange where B needs to materialize data instances from the extracted data of A. In both cases, B needs to deal with private concepts of A using Private-Target mappings as shown in Fig. 1.1. It is worth noting that the semantic interoperability from A to B does not imply semantic interoperability in the opposite direction, which needs to be additionally established by demand. However, the research results of the unidirectional semantic interoperability can be applied to bidirectional use cases.

During the last decades, several research communities have shown great interest in solutions towards semantic interoperability, including the Semantic Web, Industry 4.0, database theory, and machine learning. In particular, the following two mainstream research directions can be observed:

RD1 Harmonization of various industrial standards using ontology languages from the Semantic Web[8], and transforming (lifting) engineering data into ontologies.

RD2 Theories, methods, and systems for facilitating data access, integration, and exchange in engineering processes.

The motivation behind RD1 was that many industrial standards share a set of basic technical concepts, which might be represented differently because of varying interests or target applications [15]. For AML-based engineering, RD1 becomes even more important since AML is a meta-format that incorporates other existing standards. RD1 thus devotes to solve the mismatch or resolve the overlap between those standards. In the engineering domain, the majority of work in RD1 employs the ontology languages from the Semantic Web, benefiting from their rich modeling features and the available software tools. Representative results include the early work on the semantic lifting of CAEX [16, 17, 18] and their extensions for AML [19, 20, 21, 15], which build the foundation of **ontology-based approaches** for tackling concrete interoperability problems in RD2 [22]. Most notably, the so-called ontology-based data access (OBDA) and ontology-based data integration (OBDI) show promising results for accessing and integrating engineering data [23][24].

Nevertheless, current results from RD1 mostly focus on simple domain knowledge using primitive classes and relations. For more advanced use cases, such as the so-called GAV (Global-as-View) OBDI [24], complex domain knowledge is necessary to encode the relation between the primitive ones, which may lead to scalability problems in practice [13, 25]. Recently, Hildebrandt et al. showed that ontology building should be domain-expert-centric since the semantics expert might not be (effectively) available [26]. To better involve domain experts into the knowledge engineering procedure, Sabou et al. suggested to develop domain-specific knowledge acquisition tools such as the Siemens-Oxford Model Manager [27], and Nilsson et al. consider state-of-the-art machine learning methods for assisting ontology building [13].

In the scope of RD2, another related yet independently growing research field devotes to develop **database-based approaches** for accessing and exchanging data in relational, semi-structured (including XML), and graph databases [28, 29, 30]. In contrast to the ontology-based approaches as described above, database-based approaches are based on the foundation of database theory and work directly with the raw data using dedicated query languages. Consequently, no additional data transformation routine is required. One particularly interesting approach is the so-called Query By Example (QBE) that facilitates query construction for users who are unskilled in programming [31, 32, 33]. Using a QBE system, the user only needs to supply examples for describing the intention of the query, while an interpreter translates them into executable query programs. Because AML is an XML-based data format, adopting these approaches into AML-based engineering seems promising. However, several challenges arise due to the conflicts in the problem setting. First, the objective of database-based approaches targets at the interoperability between different database schemas, while AML-based engineering relies on the common XML schema. Second, AML defines domain-specific modeling rules which are not considered by general-purpose QBE systems. Finally, the target user group of the thesis are the domain experts who are unfamiliar with the concepts of databases.

[8]`https://www.w3.org/2001/sw/wiki`

The challenges in both ontology-based and database-based approaches encouraged the study of the following two research questions:

RQ1: How to facilitate ontology building in industrial environments and incorporate domain experts into the modeling procedure?

RQ2: How to adapt database-based approaches for AML-based engineering so that they become AML-centric while adhering to the foundations of database theory?

To answer these research questions within a reasonable scope, this thesis restricts the complexity of the semantic heterogeneity that may occur in engineering processes. According to Kovalenko and Euzenat [34], seven classes of semantic relations can be evidenced during the integration of engineering systems. The first four classes are referred to as *value processing relations* that describe data transformation rules between engineering tools. For example, one tool might store a date value as DD-MM-YYYY, while the other one prefers YYYY:MM:DD. This thesis does not consider these relations, but covers all the three structure relations as follows:

- Granularity: for example, one tool could use the class `Robot` for all kinds of robots, while another tool additionally has the term `LightWeightRobot` to refer to robots with limited weights and payload. This granularity difference can be captured by stating conditions over the correspondences between concepts. In database research, this refers to ambiguities that are caused by the class inheritance in different tools [35, 36].
- Schematic differences: it is ubiquitous that one concept is represented differently in two engineering tools. For example, one tool A might have a private class for `SaftySensor`, while another tool B only contains the class `Sensor`, which has the property `hasSafetyCertification`. Thus, a `SafetySensor` from tool A is semantically equivalent to a `Sensor` in tool B with `hasSafetyCertification=`$true$. In terms of database theory, this difference is reported as context information or categorical attributes [37, 38].
- Grouping and aggregation: this relation is relevant for cases where the data produced from a tool needs to be combined in another tool. For example, one tool produces a list of automation components that are aggregated to compose an automation system in another tool.

Under this restriction, the contributions of this thesis are threefold:

Contribution 1 (RQ1): *Developing machine learning algorithms for inducing OWL complex class expressions.* While lifting primitive domain knowledge into an ontology is well-studied, building sophisticated domain concepts remains challenging. *Concept learning* algorithms can be adopted for assisting knowledge engineering by inducing ontological concepts from labeled data [39]. This thesis reveals the drawbacks of the state-of-the-art OWL learners and presents the novel algorithm Rapid Restart Hill Climbing (RRHC). Furthermore, AML-specific modeling constraints are utilized to accelerate learning from AML data.

Contribution 2 (RQ1): *Designing an AutomationML interface for representing OWL complex classes*. Since OWL class expressions are argued to be non-intuitive for domain experts, this thesis proposes the *AML concept model* (ACM) that can (partially) preserve ontological semantics in a native AML syntax. Therefore, ACMs can be easily perceived, modified, and even generated by AML users. To support the bidirectional information flow between the user and the OWL learner, algorithms are developed for the translation between OWL complex classes and ACMs. Finally, the interactive machine learning framework AMLLEARNER is designed with a graphical user interface embedded in the conventional AML editor.

Contribution 3 (RQ2): *Developing a framework for semi-automated data access and exchange in AML-based engineering*. Inspired by the QBE paradigm, this thesis introduces the *AutomationML Query Template* (AQT) for accessing and exchanging AutomationML data using query templates. The motivation of AQT is to facilitate query construction for domain experts who are familiar with AutomationML but unskilled in programming. Therefore, AQT has an AML-based syntax, which allows constructing queries with conventional AutomationML tools. To utilize existing database technologies, the semantics of AQT is defined based on the notion of *tree pattern queries*. Finally, algorithms are presented for generating XPath [40] and XQuery [41] programs from AQTs.

The following summarizes the published papers that build the basis of the thesis:

1. Y. Hua and B. Hein. *Concept Learning in AutomationML with Formal Semantics and Inductive Logic Programming*. In 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), pages 1542–1547, August 2018. doi: 10.1109/COASE.2018.8560541. © 2018 IEEE

2. Y. Hua and B. Hein. *Concept learning in engineering based on refinement operator*. In Up-and-Coming and Short Papers of the 28th International Conference on Inductive Logic Programming (UCS-ILP 2018) Ferrara, September 2-4, 2018, volume 2206 of CEUR Workshop Proceedings, pages 76–83. RWTH, Aachen, 2018.

3. Y. Hua and B. Hein. *Interactive Learning Engineering Concepts in AutomationML*. In 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pages 1248–1251, September 2019. © 2019 IEEE

4. Y. Hua and B. Hein. *Rapid Restart Hill Climbing for Learning Description Logic Concepts*. In 29th International Conference on Inductive Logic Programming (ILP 2019), Plovdiv, September 2019.

5. Y. Hua and B. Hein. *Interpreting OWL Complex Classes in AutomationML based on Bidirectional Translation*. In 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pages 79–86, September 2019. © 2019 IEEE

6. Y. Hua and B. Hein. *AQT - A Query Template for AutomationML*. IEEE Transaction on Industrial Informatics, April 2020. doi: 10.1109/TII.2020.2989125. © 2020 IEEE

1.2. Chapter Overview

The rest of the thesis is organized as follows. Chapter 2 introduces the basic notions of AML and the detailed workflow of AML-based engineering. For describing the challenges caused by the mixed-model principle, Section 2.2 presents a layered view of the required semantic mappings (Private-Target mappings as shown in Fig. 1.1), based on which related work towards semantic interoperability in AML is discussed.

The main part of the thesis begins with ontology-based approaches. In Chapter 3, a brief introduction of Semantic Web languages is introduced, and the related work on the semantic lifting of AML is discussed. This chapter continues with the foundations of concept learning and describes the utility of concept learning in AML-based engineering.

After the basics, Chapter 4 first conducts a comparative study of the well-known algorithm CELOE in Section 4.1 and then develops algorithms for concept learning in OWL (Section 4.2) and AML (Section 4.3.1). In Section 4.4, experiments are carried out for evaluating the performance of the proposed algorithms. For representing the learned OWL class expressions in AML, Chapter 5 proposes the AML Concept Model (ACM) and presents the bidirectional translation between ACM and OWL. Then, Section 5.3 introduces the AMLLEARNER framework, and Section 5.4 discusses the interactive features of AMLLEARNER according to the literature of interactive machine learning systems.

The second part of the thesis aims at developing a QBE-like approach for accessing and exchanging AML data. Chapter 6 begins with the introduction of the standard XML Query languages and tree patterns. Then, foundations of XML data exchange are described in Section 6.3, and the related work on QBE-based approaches are discussed in Section 6.4.

Based on the notion of tree patterns, Chapter 7 proposes the AML Query Template (AQT) for accessing engineering data stored in AML files and presents algorithms for translating AQTs into XPath and XQuery programs. To enable semi-automated data exchange, Chapter 8 extends the AQT-based approach for the materialization of a target AML document. In particular, requirements and theoretical foundations for AML data exchange are studied in Section 8.1 and 8.2, which provide the necessary notions and guidelines for the extension of AQT. Section 8.5 compares AQT with general-purpose QBE systems and shows that AQT is more suitable for data exchange with AML.

Finally, Chapter 9 concludes the thesis and outlines ideas for future work.

2. AutomationML-based Engineering

The development of AutomationML (AML) was driven by the strong need of a neutral, cross-domain data format for seamless data exchange between engineering tools [42]. With the slogan "*the glue for seamless automation engineering*," AML is designed as a meta-format that integrates several existing standards. Syntactically, an AML document is an XML file that adheres to the XML schema of the Computer-Aided Engineering Exchange (CAEX, IEC 62424) [43]. To describe geometry and kinematic models, AML provides an interface to the data format COLLADA (as ISO/PAS17506). For specifying logic behavior of automation components and systems, PLCopen (as IEC 61131) models can be referenced. Fig. 2.1 shows the relations between AML and those cooperating standards. Recently, a joint working group with the OPC foundation extends the scope of AML to model on-line information like process data and diagnostic information [44]. The integration of eCl@ss into AML enables the usage of the standardized object and property semantics [45], e.g., the definition of the class `ArticulatedRobot` and the attribute `ManufacturerName`.

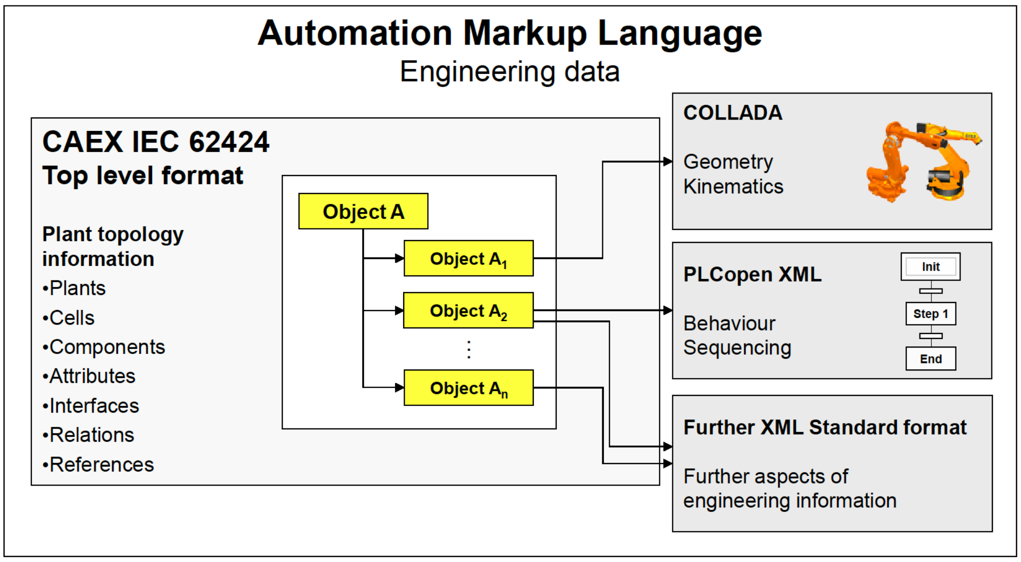


Figure 2.1.: The relations between AutomationML and its cooperating standards [46].

2.1. AutomationML modeling facilities

For providing the basic understanding of data modeling in AML, the CAEX schema elements are explained first. Fig. 2.2 shows the class diagram of the CAEX schema in

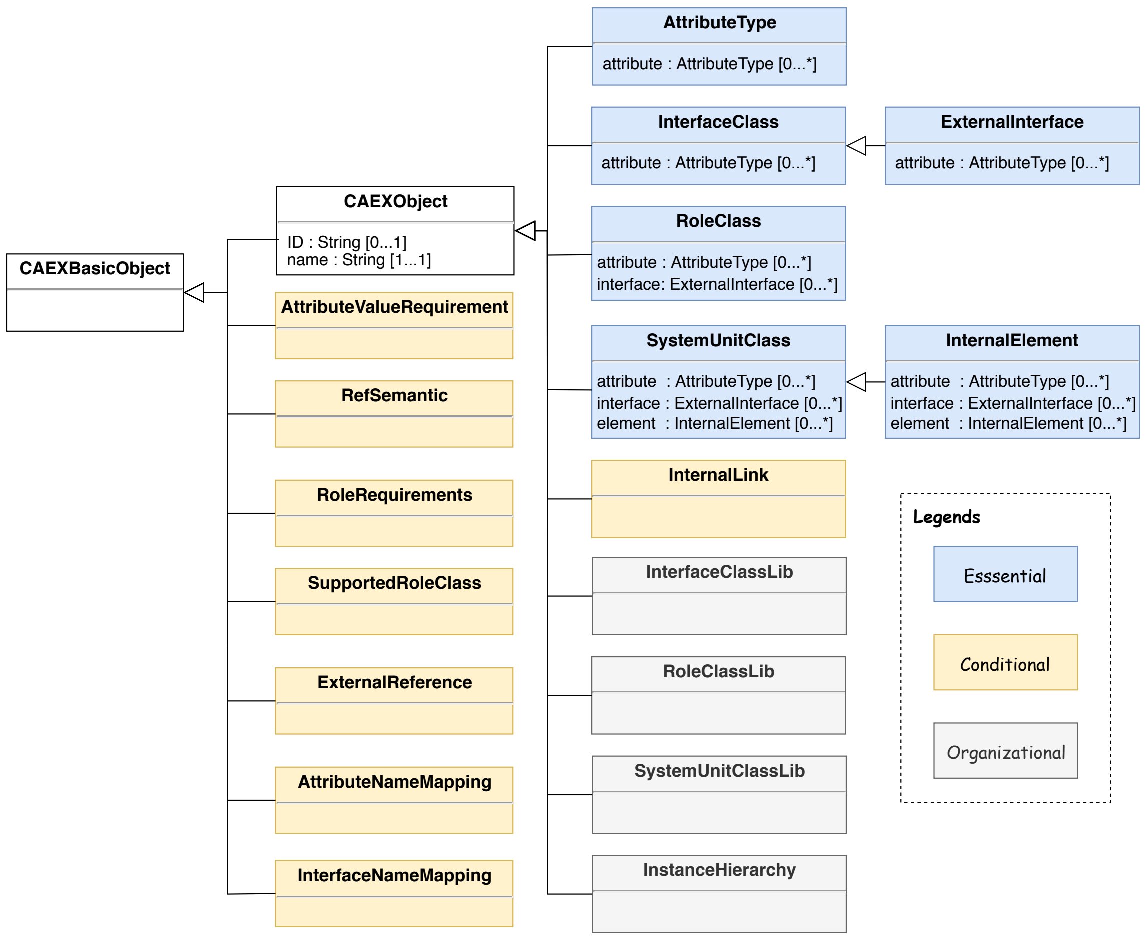

AttributeType
attribute : AttributeType [0...*]
InterfaceClass
attribute : AttributeType [0...*]
ExternalInterface
attribute : AttributeType [0...*]
CAEXObject
ID : String [0...1]
name : String [1...1]
RoleClass
attribute : AttributeType [0...*]
interface: ExternalInterface [0...*]
CAEXBasicObject
AttributeValueRequirement
SystemUnitClass
attribute : AttributeType [0...*]
interface : ExternalInterface [0...*]
element : InternalElement [0...*]
InternalElement
attribute : AttributeType [0...*]
interface : ExternalInterface [0...*]
element : InternalElement [0...*]
RefSemantic
InternalLink
RoleRequirements
Legends
Esssential
SupportedRoleClass
InterfaceClassLib
Conditional
ExternalReference
RoleClassLib
Organizational
AttributeNameMapping
SystemUnitClassLib
InterfaceNameMapping
InstanceHierarchy

Figure 2.2.: The class diagram of the CAEX schema V2.15. Essential concepts are marked as blue, while conditional and organization ones are marked as yellow and grey respectively.

version 2.15, while Fig. 2.3 illustrates the architecture of a typical AML document. One can observe the following four modules that constitute an AML document:

Role Class Libraries: a role class library collects shared concepts in engineering. Each concept is modeled as an AML **role class**, which can be extended with properties and interfaces. AML supports the modeling of class-class relations (i.e., inheritance) using the keyword RefBaseClassPath. The first part[1] of IEC 62714 specifies the role class library `AutomationMLBaseRoleClassLib` that contains abstract concepts in engineering, such as `Product`, `Process`, `Resource`, etc. This base library is refined into a series of domain-specific ones in the second part[2] of IEC 62714. For example, the `AutomationMLDMIRoleClassLib` comprises concepts for discrete manufacturing, e.g., `Robot` and `Tool`, and the `AutomationCSRoleClassLib` contains basic object types for control systems, e.g., `Controller` and `Sensor`. Fig. 2.4a shows the inheritance hierarchy of some standardized AML role classes. Finally, private role classes can be derived from standardized ones.

Interface Class Libraries: an interface class library is a collection of shared concepts about interfaces. Each concept is modeled as an **interface class** between which

[1]`https://webstore.iec.ch/publication/32339`
[2]`https://webstore.iec.ch/publication/22030`

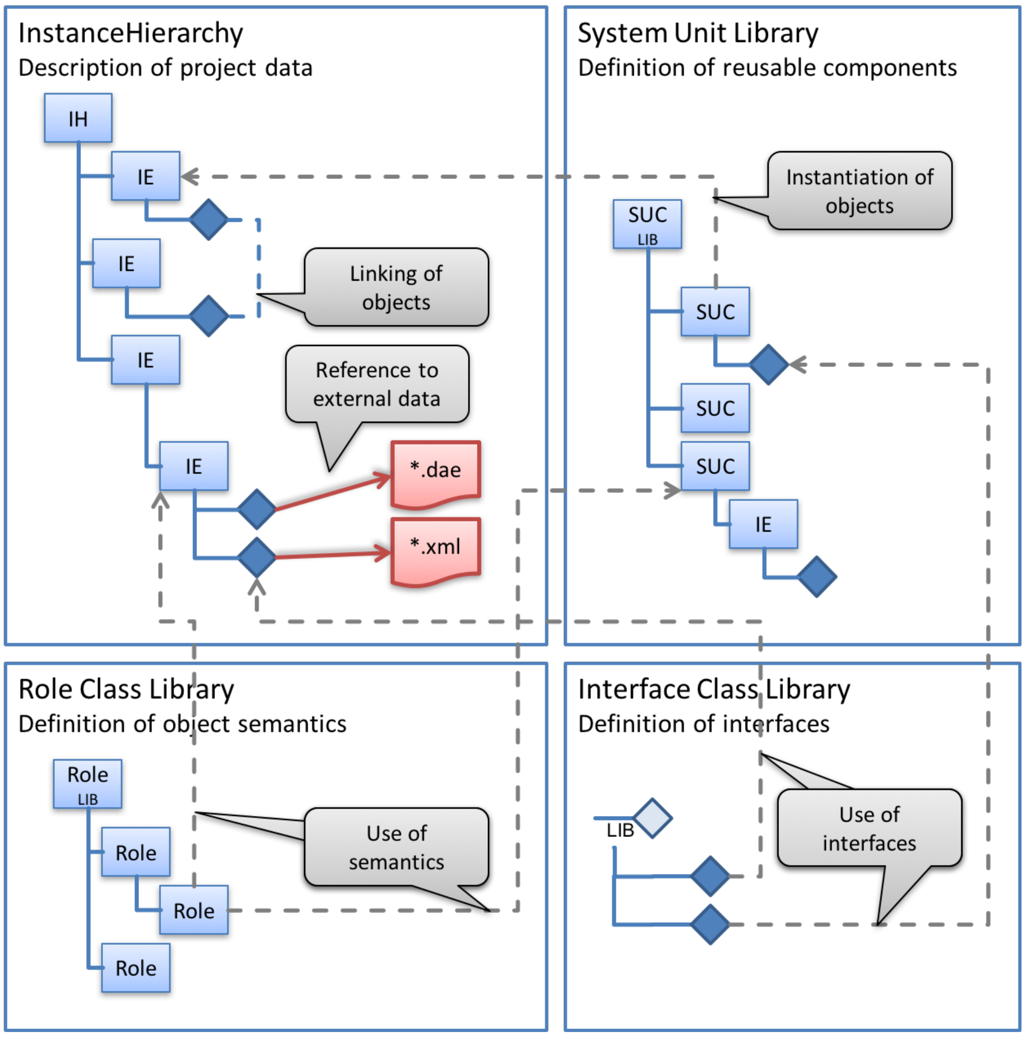
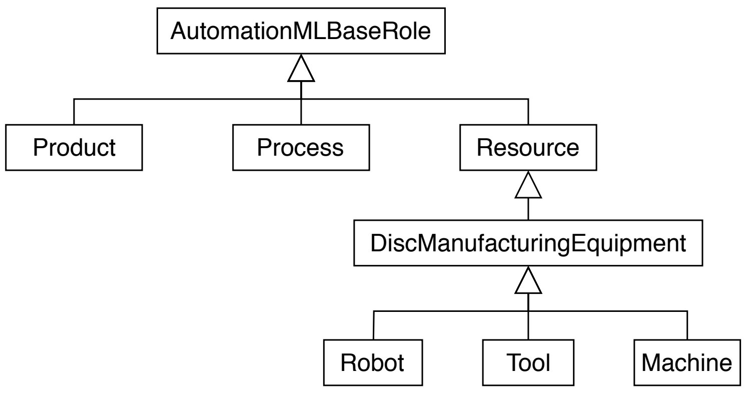
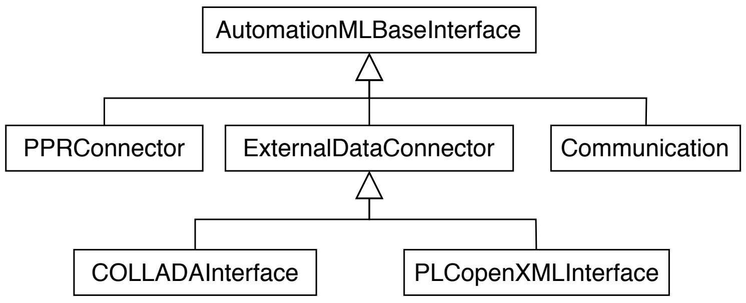


Figure 2.3.: The architecture of an AutomationML document [47]. For sake of brevity, instance hierarchy is written as IH, internal element is written as IE, system unit class is written as SUC, role class is written as Role, and librariy is written as LIB.



(a) Examples of standardized AML role classes.



(b) Examples of standardized AML interface classes.

Figure 2.4.: The inheritance hierarchy of some AML role and interface classes.

inheritance relations can be specified. Again, a basic set of interface classes has been standardized (cf. Fig. 2.4b). For example, `COLLADAInterface` is designed for connecting geometry or kinematics description in COLLADA files and is modeled as a subclass of `ExternalDataConnector`, and the class `Communication` stands for communication interfaces between automation objects. Similar to the role classes, interface classes can be refined by the user for covering specific modeling requirements.

System Unit Class Libraries: while the previous two modules are used for shared domain concepts, a system unit class library includes tool-specific models of engineering components or systems. Those models are referred to as **system unit classes** and may comprise subordinated object structures and interfaces that are represented as nested AML **internal elements** and **external interfaces**, respectively. The semantics of a system unit class or an internal element is defined by the associated role classes using class-instance relations. AML provides two mechanisms for this purpose: the SupportedRoleClass can be used for both system unit classes and internal elements, and the RoleRequirements is only applicable to internal elements but allows a more comprehensive description of the requirements of the referenced role class. An external interface represents logical or physical connection points on automation objects. The semantics of an external interface can be specified with a class-instance relation to an interface class. Connections between external interfaces can be modeled as instance-instance relations using AML InternalLinks. Finally, internal elements and external interfaces are uniquely identified by their universally unique identifiers (UUID).

Instance Hierarchies: an AML instance hierarchy represents the topological structure of a plant or an automation system, where individual objects and interfaces are modeled as internal elements and external interfaces, respectively. An internal element can either be constructed from scratch or instantiated from system unit classes using a copy operation. However, the copy operation distinguishes the conventional instantiation in the object-oriented paradigm, since the internal element can modify, add, or delete data from the original system unit class.

The properties of classes, interfaces, and objects are described using CAEX **attributes**. The adjective "CAEX" is necessary for attributes since the notion *AML attribute* is reserved for all CAEX attributes that are standardized by the AML association. For example, `direction` is an AML attribute that is defined for the interface class `Order`. Syntactically, a CAEX attribute is a complex XML element that may include the name, the data type, the unit, and the value or AttributeValueRequirements. AML mainly supports two types of value requirements: the *nominal scaled type* that contains a list of permitted values, and the *ordinal scaled type* that specifies the minimum and maximum of the value. The semantics of a CAEX attribute can be defined using AML RefSemantic. For example, an eClss property can be linked to a CAEX attribute following the rules described in [45]. Finally, CAEX attributes can be nested for representing complex data structures. For example, a `coordinate` attribute can have three sub-attributes $(\mathtt{x}, \mathtt{y}, \mathtt{z})$.

In addition to the CAEX schema elements, IEC 62424 standardizes a set of XML attributes (not CAEX attributes) for the serialization of a CAEX document, most notably including:

- `Name`: name of one object, interface, class, or attribute.
- `ID`: UUID of one object or interface.
- `Unit` and `AttributeDataType`: unit and data type of a CAEX attribute.
- `RefBaseClassPath`: the path of a role, interface, or system unit class w.r.t. its super class.
- `RefBaseRoleClassPath`: the path of the referenced role class in role requirements.
- `RefRoleClassPath`: the path of the reference role class in a supported role class.
- `RefPartnerSideA` and `RefPartnerSideB`: the UUID of the connected external interfaces in an internal link.

2.2. AutomationML-based Engineering

The CAEX schema, as described in the last section, defines the syntax of a valid AML document. However, as reported by Hua et al. [48], AML only supports human-interpretable semantics, which is described in its IEC and supplementary technical specifications. For example, the XML attribute `RefBaseClassPath` is defined as "*If inheritance is required, the parent class shall be specified using the CAEX tag "RefBaseClassPath" comprising the full path of the class according to IEC 62424:2008, A.2.7.*" This information declares the intended meaning of `RefBaseClassPath`, but only for human interpretation. A machine could not process this information without programming effort from human experts. One would argue that the role and interface class concept is designed for machine interpretation since once the XML document is parsed, each engineering object can be semantically identified by its role class. However,

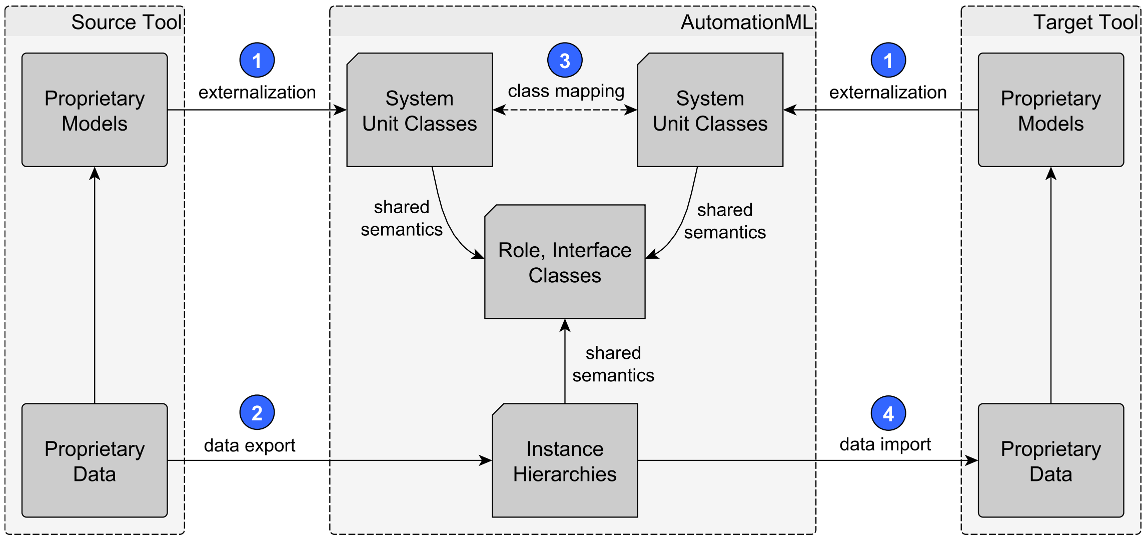


Figure 2.5.: The “standard” data exchange process between two engineering tools while using AutomationML as an intermediate layer.

besides the standardized role classes, which are themselves too general, the mixed model principle of AML allows private concepts that are defined by individual tools and are not semantically interchangeable per se (cf. Fig. 1.1).

As a consequence, the standard data exchange pipeline suggested by the AutomationML e.V. follows an exporter-importer-based approach [49]. Fig. 2.5 illustrates the data flow from a source to a target engineering tool via the intermediate AML layer.

The first step in the process is the *externalization* of the proprietary models from each engineering tool to AML system unit classes. These proprietary models are the abstraction of the private data in each engineering tool.

The second step is called data *export*, which only happens on the source tool side. In this step, proprietary data objects in the source tool are serialized to an AML-conform format, following the modeling principles described in Section 2.1. The result of the export can be one or several AML instance hierarchies that contain concrete data objects from the source tool as AML internal elements. Ideally, each internal element is an instantiation of a system unit class that emerges during the externalization step.

Next, *mappings* between the system unit classes of the source and target tool are defined. Let S be a source system unit class, T be a target system unit class, and s be a source data object (an internal element) that is instantiated from S. Let R_S be the set of role classes referenced in S and R_T the set of role classes referenced in T. One distinguishes between the following two scenarios for the mapping:

- In the simplest case, $R_S \subseteq R_T$, which means that all data elements in s can be directly transferred to the target tool.
- If $R_S \supset R_T$ or $R_S - R_T \neq \emptyset$, the mapping shall cover the part of s which has no direct correspondence in T. In this case, either the remaining data is ignored, or it has to be stored under additional strategies.

In the last step, the target tool *imports* the source instance hierarchies using the mappings generated in the last step.

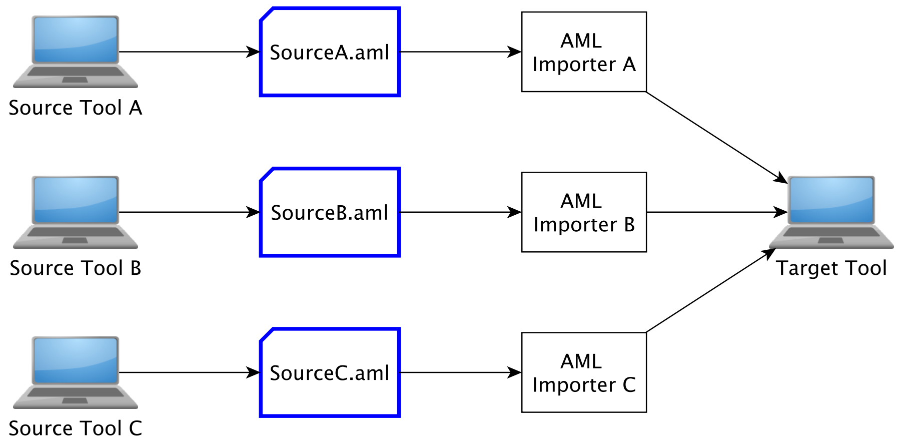


Figure 2.6.: One tool might need dedicated AML importers for different source tools.

The key prerequisite of the standard data exchange pipeline is the agreement of a set of shared role and interface classes between the tools[3]. This is a strong assumption in practice, because,+ on the one hand, such an agreement requires intensive negotiations between different stakeholders in an engineering project. Moreover, if $R_S \subseteq R_T$, the standard data exchange pipeline can be categorized to the second level of the VDI/VDE guideline 3695, as described in Chapter 1. On the other hand, if $R_S \supset R_T$ or $R_S - R_T \neq \emptyset$, private concepts of the source tool needs to be treated individually via class mappings, which can be rather complex since semantic alignments between the tools need to be established [34]. Fig. 2.6 illustrates this dilemma when a target tool needs data from various source tools. For each source tool, a dedicated AML importer[4] is required as the class mappings may vary among different sources. To understand the scale of these mappings, we compare AML-based engineering with the meta-model hierarchy proposed by the Object Management Group (OMG[5]), which distinguishes between the following four levels:

- M3: this level contains the *meta-meta-models* which serve as a language for defining meta-models. Examples of M3 level languages include the OMG Meta Object Facility (MOF[6]) and the EMF core (Ecore[7]) from the Eclipse framework.
- M2: this level defines the language, or the *meta-models*, for constructing models. In model-driven engineering, meta-models are often referred to as a platform-independent specification of a domain.
- M1: once the meta-model is defined, a platform-dependent *model* can be generated that describes the concrete data.
- M0: this layer contains concrete instance data of a model.

The CAEX schema is a meta-model (M2) that specifics the engineering domain. Nevertheless, the role and interface classes cannot be easily categorized because they use the CAEX schema as their modeling language while being platform-independent.

[3]In reality, they also need to agree with the attributes, which are not the primary concern of this thesis.
[4]Or at least a dedicated extension of the basic AML importer if implemented properly.
[5]`https://www.omg.org/`
[6]`https://www.omg.org/mof/`
[7]`https://www.eclipse.org/modeling/emf/`

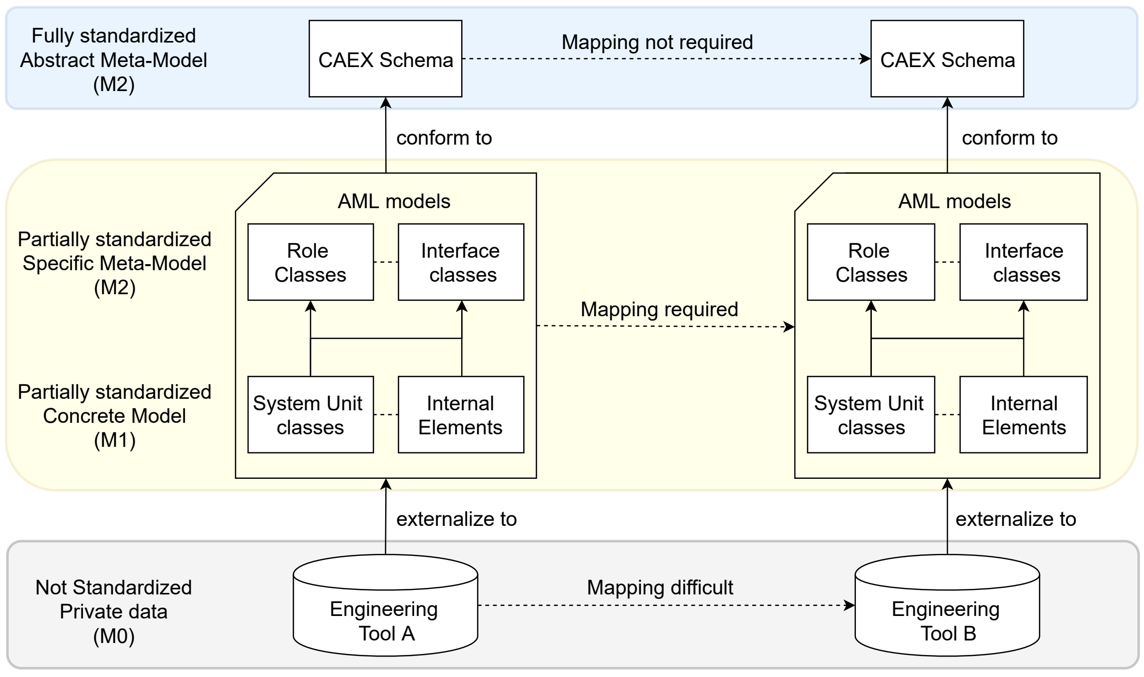


Figure 2.7.: A layered view of the mappings between two tools that support AML-based engineering. The blue layer denotes the fully standardized part of AML, the yellow layer refers to the partially standardized part of AML, and the gray layer represents non-standardized private data.

Therefore, a distinction is needed within the M2 level that separates the class definitions in AML from the underlying schema, as illustrated in Fig. 2.7:

- The CAEX schema itself is called an *abstract meta-model* (M2). There is no mapping required in this layer because tools use the same CAEX schema.
- The role and interface classes are called *specific meta-models* (M2). Since the mixed-model principle allows deriving private domain concepts from the standardized ones, mappings are required between the privately defined role and interface classes.
- System unit classes and internal elements belong to the M1 level as they are tool-specific. Because their associated role and interface classes can be partially standardized, mappings are also required within the M1 level.
- The proprietary data of each engineering tool belong to the M0 level. It is difficult to formulate mappings at this level because there is no agreement on the syntax. In practice, human experts are responsible for transferring data from one tool to another (cf. interoperability level 1 in Chapter 1).

In conclusion, mappings are required in the layers M1 and M2. Currently, these mappings are often embedded into the AML importer of the target tool. From a model-based engineering perspective, the complexity of the importer can be significantly reduced if the mapping specification is separated from its implementation. However, while AML has a rich vocabulary to model relations between engineering model artifacts [47], the standard provides no guidelines for describing these mappings.

Recently, researches have been conducted to tackle this problem. Bigvant et al. first introduced the *semantic data hub* concept that manages the mappings in an intermediate mapping library [50]. This approach mainly focuses on the aspects of the novel engineering process and system architectures and is less concerned about the implementation. Bihani and Drath proposed to embed 1-1 mappings between CAEX attributes into the AML file of the source tool [1]. Each mapping consists of the source tool attribute, the desired target tool, and the identifier of the target tool attribute. This work is extended in [51] by allowing more complex mapping types, e.g., mathematical functions, between the values of CAEX attributes. These approaches, however, do not provide sufficient expressiveness for complex data exchange scenarios, e.g., class and object structure mappings. In the field of semantic technologies, several studies have attempted to transform AML data into ontologies and describe the mappings using expressive ontology languages. These approaches will be discussed in Section 3.1.4.

Part I.

Ontology-based Approach

3. Preliminaries and Related Work

This chapter gives a brief introduction to Semantic Web technologies and discusses the related work on inductive learning in description logics.

3.1. Semantic Web Languages

In 2001, Tim Berners-Lee and others envisioned the future of the World Wide Web in the article "The Semantic Web" [52]. The main emphasis of this article was to bring machine-understandable meaning to the documents in the world wide web and to represent knowledge using tools from artificial intelligence, e.g., ontologies and rules. The Semantic Web thus brings logical structures to the web content and fosters automated inference that can answer complex questions or help decision making. Since then, various working groups are established within the World Wide Web Consortium (W3C), which have led to a set of standards. The most notable ones include the Resource Description Framework (RDF) and its schema language (RDFS)[1], the SPARQL query language, and the Web Ontology Language (OWL). In addition, numerous implementations emerged, e.g., inference engines, open-source libraries, and ontology editors. These efforts promoted the usage of Semantic Web technologies in other fields, including engineering in the industrial context [53].

3.1.1. RDF(S) and SPARQL

This section gives a brief overview of RDF(S) and SPARQL that is necessary for understanding the thesis. For more details about these languages, the readers are referred to the corresponding W3C specifications and the book "Foundations of Semantic Web Technologies" [54].

The Resource Description Framework (RDF) is a language for expressing information about resources. Each statement in RDF is a *triple* in the form of *subject-predicate-object*, and an RDF document is a collection of such triples. Informally, a triple describes the relation, denoted by the predicate, between two resources denoted by the subject and object. RDF employs the Uniform Resource Identifiers (URI) for the global identification of resources or relations and uses *literals* for concrete values. Syntactically, URIs can present at any position in a triple, while literals can only appear as an object. If a resource does not require a global identification, it is represented as a *black node*. An RDF document is also called an *RDF graph*, where subjects and objects correspond to the graph nodes, and each predicate connects a pair of subject and object. A resource with a URI is illustrated by an ellipse, and a literal is

[1]In the literature, they are often written together as RDF(S).

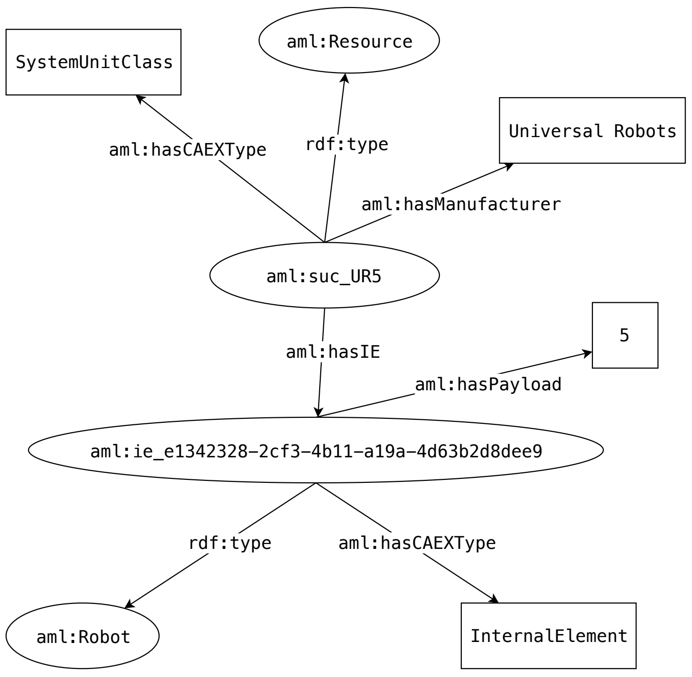


Figure 3.1.: The RDF graph of the robot `UR5`.

shown as a box. Fig. 3.1 shows the RDF graph of the robot `UR5` that is originally modeled in AML as a system unit class. A URI, e.g., `aml:suc_UR5`, consists of the namespace `aml`, defined as `xmlns:aml="http://www.ipr.kit.edu/aml#"`, and the resource identifier `suc_UR5`.

RDF supports several serialization formats. The RDF/XML format[2] was introduced in 2004 to store RDF data in XML. The following XML code describes the robot `UR5` in Fig. 3.1:

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:aml="http://www.ipr.kit.edu/aml#"

    <aml:Resource rdf:about="aml:suc_UR5">
        <aml:hasCAEXType>SystemUnitClass</aml:hasCAEXType>
        <aml:hasManufacturer>Universal Robots</aml:hasManufacturer>
        <aml:hasIE rdf:resource="aml:ie_e1342328-2cf3-4b11-a19a-4d63b2d8dee9"/>
    </aml:Resource>

    <aml:Robot rdf:about="aml:ie_e1342328-2cf3-4b11-a19a-4d63b2d8dee9">
        <aml:hasCAEXType>InternalElement</aml:hasCAEXType>
        <aml:hasPayload>5</aml:hasPayload>
    </aml:Robot>
</rdf:RDF>
```

While RDF can describe the relations between resources, RDF Schema (RDFS) is used to model the so-called terminological knowledge. The essential part of the terminological knowledge is a vocabulary that specifies the concepts and their relations in a domain of interest. For example, `aml:Resource` in Fig. 3.1 is a concept in the engineering domain that refers to the class of all engineering resources. RDFS is capable of building a hierarchy of such domain concepts by means of subclass relations. For example, `aml:Actuator` can be modeled as a subclass of `aml:Resource`, which denotes that all instances of `aml:Actuator` is also an instance of `aml:Resource`. This modeling capability makes RDFS a (lightweight) ontology language.

[2] `https://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/`

With data stored in an RDF document or a *triple store*, one might be interested in extracting knowledge from it. SPARQL is a query language designed for retrieving data from RDF graphs. Each SPARQL query contains a set of triple patterns, in which subjects, objects, or relations can be replaced by variables (represented by a leading ?). A SPARQL query thus looks for parts from the RDF graph which match the triple patterns and returns the data in the place of variables. For example, one could ask for all robots from the manufacturer Universal Robots with a limited payload as follows:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX aml: <http://www.ipr.kit.edu/aml#> .
SELECT ?robot
WHERE {
    ?robot aml:hasManufacturer  "Universal Robots" .
    ?robot aml:hasIE            ?arm .
    ?arm   rdf:type             aml:Robot .
    ?arm   aml:hasPayload       ?payload .
    FILTER (?payload < 10)
}
```

3.1.2. Description Logics

The expressivity of RDF(S) is rather limited for modeling complex domain knowledge, which leads to the development of the Web Ontology Language (OWL). While the design of OWL was subject to several influences [55], its mathematical foundation is primarily based on a family of knowledge representation languages called Description Logics (DL).

Historically, DLs originate in the study of the so-called *network-based structures* [56], e.g., semantic networks [57] and frames [58], that are attempts for representing knowledge similar to how humans perceive meaning. The reasoning services of these attempts, however, produce different results since no precise machine-interpretable semantics was given. In contrast to these predecessors, DLs are equipped with a formal semantics grounded on the classical first-order logic.

Knowledge representation with DLs starts with the modeling of terminologies in the domain of interest, i.e., *concepts* and *roles*. A concept represents a set of individuals that share some common properties, and a role represents the binary relation between two individuals. For example, in the family relation domain, one can define several DL concepts, including $\mathsf{Person}, \mathsf{Male}$, and Female, and state that all Males and Females are also Persons using *concept inclusion axioms* as:

$$\mathsf{Male} \sqsubseteq \mathsf{Person}, \mathsf{Female} \sqsubseteq \mathsf{Person} \tag{3.1}$$

To describe the parent-child relationship between two persons, one can define the DL roles $\mathsf{hasChild}$ and $\mathsf{hasParent}$. From the basic DL concepts and roles, more comprehensive knowledge can be built using *concept constructors*. For example, the following *equivalence axiom* gives a *concept definition* for the concept name Father:

$$\mathsf{Father} \equiv \mathsf{Male} \sqcap \exists \mathsf{hasChild}.\top \tag{3.2}$$

The symbol "$\equiv$" stands for the logical equivalence. The symbol "$\sqcap$" (reads "and") corresponds to the classical logic operator AND, the symbol "$\exists$" (reads "exists") is the *existential quantifier*, and "$\top$" (reads "top") means any individual in the domain. The expression $\exists\mathsf{hasChild}.\top$ describes all individuals which have at least one *filler* of the role $\mathsf{hasChild}$ that belongs to the concept $\top$. Therefore, axiom 3.2 characterizes the concept Father as Male with at least a child.

In addition to the terminological knowledge, one could make assertions about the individuals in the domain. In the family case, one could add individuals of Persons using *concept assertions*, and describe their parent-child relationships with *role assertions*. For example, the following axioms state that ALICE is a female, $\mathsf{YINGBING}$ is a male, and ALICE is a child of $\mathsf{YINGBING}$:

$$\begin{aligned} \mathsf{Female}(\mathsf{ALICE}), \mathsf{Male}(\mathsf{YINGBING}) \\ \mathsf{hasChild}(\mathsf{YINGBING}, \mathsf{ALICE}) \end{aligned} \tag{3.3}$$

This thesis adopts the conventional writing style in the DL literature, namely, the first letter of a concept name (for example Person) is written in uppercase, the individual names (for example ALICE) are always in uppercase, and the role names (for example $\mathsf{hasChild}$) are written in camel-case notations. Furthermore, a, b, c are used to denote arbitrary individual names, r, s are used to denote arbitrary role names, and A, B are used to denote arbitrary concept names. Finally, C, D are used to refer to (possibly) complex concept descriptions, for example, the right-hand side of axiom 3.2.

DLs can be distinguished regarding their expressivity, that is, what kind of concept constructors are supported and which assertions can be made. The most basic DL considered in this thesis is the *attributive concept descriptions with complements*, or $\mathcal{ALC}$, developed by Schmidt and Smolka in 1991 [59]. The concept constructors of $\mathcal{ALC}$ can be defined as follows [60]:

Definition 3.1 ($\mathcal{ALC}$ Concepts)**.** Let $\mathbf{C}$ be a set of concept names and $\mathbf{R}$ be a set of role names with $\mathbf{C} \cap \mathbf{R} = \emptyset$. The set of $\mathcal{ALC}$ concepts is inductively defined as follows:

- Each concept name $A \in \mathbf{C}$ is an $\mathcal{ALC}$ concept.
- The top concept $\top$ that refers to all individuals in the domain, and the bottom concept $\bot$ that refers to none individuals are $\mathcal{ALC}$ concepts.
- Let C and D be $\mathcal{ALC}$ concepts, and r a role name in $\mathbf{R}$, then the following are also $\mathcal{ALC}$ concepts:
  - $C \sqcap D$ (conjunction), $C \sqcup D$ (disjunction), $\neg C$ (negation)
  - $\exists r.C$ (existential restriction), $\forall r.C$ (universal restriction) $\square$

Two kinds of $\mathcal{ALC}$ concepts can be constructed: the *atomic* concepts are elements from $\mathbf{C} \cup \{\top, \bot\}$, and the *complex* concepts are built by compounding atomic concepts and roles with the operators $\sqcap, \sqcup, \neg, \exists, \forall$. Recall the family example above. The set $\mathbf{C}$ contains the concept names $\{\mathsf{Person}, \mathsf{Female}, \mathsf{Male}\}$, and the set $\mathbf{R}$ contains the role name $\{\mathsf{hasChild}\}$.

A DL *knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a *TBox* $\mathcal{T}$ that contains the terminological knowledge and an *ABox* $\mathcal{A}$ that includes the assertional knowledge.

Example 3.1 ($\mathcal{ALC}$ knowledge base)**.** The $\mathcal{ALC}$ knowledge base of the family example can be defined with:

the TBox $\mathcal{T}$:

$$\begin{aligned} \mathsf{Female} &\sqsubseteq \mathsf{Person} \\ \mathsf{Male} &\sqsubseteq \mathsf{Person} \\ \mathsf{Father} &\equiv \mathsf{Male} \sqcap \exists \mathsf{hasChild}.\top \end{aligned} \tag{3.4}$$

and the ABox $\mathcal{A}$:

$$\begin{aligned} &\mathsf{Male(YINGBING)} \\ &\mathsf{Female(WEI)} \\ &\mathsf{Female(ALICE)} \\ &\mathsf{hasChild(YINGBING, ALICE)} \quad \square \end{aligned} \tag{3.5}$$

Much effort has been spent to investigate the *trade-off* between the expressive power and the reasoning complexity of various DLs. Meanwhile, more efficient algorithms are developed to support the computation of inference. The DL research community favors some special letters for particular modeling features, and names DLs by combining such letters. This thesis follows this consensus and uses the following letters as described in the literature:

$\mathcal{S}$ ($\mathcal{ALC}$ with transitive roles): A transitive role r can model indirect relations between individuals. For example, the role $\mathsf{hasChild}$ is a direct relationship between two persons, while the role $\mathsf{hasDescendant}$ can be specified as transitive to state that, if $\mathsf{hasDescendant(a, b)}$ and $\mathsf{hasDescendant(b, c)}$, then $\mathsf{hasDescendant(a, c)}$.

$\mathcal{H}$ (role hierarchies): similar to the concept inclusion axioms, a hierarchy between roles can be specified using role inclusions. For example, the role $\mathsf{hasChild}$ can be defined as a sub-role of $\mathsf{hasDescendant}$, written $\mathsf{hasChild} \sqsubset \mathsf{hasDescendant}$, which indicates that, if $\mathsf{hasChild(a, b)}$ then $\mathsf{hasDescendant(a, b)}$.

$\mathcal{R}$ (complex role inclusions): the composition of two roles r and s, represented by $r \circ s$, can appear on the left-hand side of role inclusion axioms, which formulate complex role inclusions. For example, $\mathsf{hasBrother} \circ \mathsf{hasChild} \sqsubseteq \mathsf{hasCousin}$ states that, if $\mathsf{hasBrother(a, b)}$ and $\mathsf{hasChild(b, c)}$, then $\mathsf{hasCousin(a, c)}$. Complex role inclusion is an expressive modeling feature whose usage shall obey several restrictions to ensure the decidability of inference [54].

$\mathcal{I}$ (inverse roles): an inverse role is defined as the inverse of another role. For example, the following axiom denotes that if $\mathsf{hasParent(a, b)}$ then $\mathsf{hasChild(b, a)}$.

$$\mathsf{hasParent} = \mathsf{hasChild}^{-} \tag{3.6}$$

Axioms that make use of the features above are sometimes referred to as the *RBox* of a DL knowledge base. RBox can also contain equivalence and disjointness relations between roles. More expressive DLs support the following modeling features:

$\mathcal{O}$ (nominals): in addition to DL concepts, which represent a set of individuals, nominals are used to build a singleton set that contains only one individual, e.g., $\{\mathsf{YINGBING}\}$. Nominals can be combined with the union operator $\sqcup$ to enumerate a set of individuals. For example, $\{\mathsf{YINGBING}\} \sqcup \{\mathsf{ALICE}\} \sqcup \{\mathsf{WEI}\}$ enumerates the family members of $\{\mathsf{YINGBING}\}$.

$\mathcal{Q}$ (qualified number restrictions): they restrict the cardinality of individuals that can be reached via a role. For example, a complex concept "father with more than two female children" can be modeled as $\mathsf{Male} \sqcap > 2\mathsf{hasChild}.\mathsf{Female}$.

$\mathcal{D}$ (data types): some DLs explicitly support primitive data types, e.g., integer and string, to model the property of some individuals. Data types are realized with *concrete domains* and are used together with *concrete roles*. For example, the age of a person is a concrete role, and the value of age is an integer.

Until now, the syntax and meaning of DL axioms are described informally. Nevertheless, formal, machine-interpretable semantics is required to foster automated reasoning. For example, the concept assertion $\mathsf{hasParent}(\mathsf{ALICE}, \mathsf{YINGBING})$ is expected if the knowledge base in Example 3.1 is extended with the axiom 3.6.

DLs are usually equipped with a *model-theoretical* semantics that is defined by means of *interpretations*. For a DL with a set of concept names $\mathbf{C}$, a set of role names $\mathbf{R}$, and a set of individual names $\mathbf{I}$, where $\mathbf{C}, \mathbf{R}, \mathbf{I}$ are pairwise disjoint, an interpretation can be defined as follows [60]:

Definition 3.2 (Interpretation)**.** An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, and a mapping $\cdot^{\mathcal{I}}$ which maps:

- each concept name $A \in \mathbf{C}$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and
- the top concept $\top$ to $\Delta^{\mathcal{I}}$ and the bottom concept $\bot$ to $\emptyset$, and
- each role name $r \in \mathbf{R}$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and
- each individual name $a \in \mathbf{I}$ to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. $\square$

The set $\Delta^{\mathcal{I}}$ is called the *interpretation domain*, and the mapping $\cdot^{\mathcal{I}}$ is called the *interpretation function*. The following shows a possible interpretation of the knowledge base in Example 3.1:

Example 3.2 (Interpretation)**.**

$$\begin{aligned}
\Delta^{\mathcal{I}} &= \{\mathsf{YINGBING}, \mathsf{ALICE}, \mathsf{WEI}, \mathsf{BOB}\} \\
\mathsf{YINGBING}^{\mathcal{I}} &= \{\mathsf{YINGBING}\} \\
\mathsf{ALICE}^{\mathcal{I}} &= \{\mathsf{ALICE}\} \\
\mathsf{WEI}^{\mathcal{I}} &= \{\mathsf{WEI}\} \\
\mathsf{Person}^{\mathcal{I}} &= \{\mathsf{YINGBING}, \mathsf{ALICE}, \mathsf{WEI}, \mathsf{BOB}\} \\
\mathsf{Male}^{\mathcal{I}} &= \{\mathsf{YINGBING}, \mathsf{BOB}\} \\
\mathsf{Female}^{\mathcal{I}} &= \{\mathsf{WEI}, \mathsf{ALICE}\} \\
\mathsf{hasChild}^{\mathcal{I}} &= \{(\mathsf{WEI}, \mathsf{ALICE}), (\mathsf{YINGBING}, \mathsf{ALICE})\} \quad \square
\end{aligned} \tag{3.7}$$

In this example, the interpretation domain has one more element (BOB) than the individuals in Example 3.1. In fact, the interpretation domain can have an arbitrary cardinality (at least one) and may even be infinite. Furthermore, the labels of the elements in the domain have no significance. They are named identically to the individual names only for convenience. The interpretation of a complex concept follows from the interpretation of each component that constitutes it. Table 3.1 shows the semantics of complex concepts expressible in the language $\mathcal{SROIQ}$.

Table 3.1.: Semantics of $\mathcal{SROIQ}$ constructors.

	DL Syntax	**Semantics**
intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
complement	$\neg C$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
existential restriction	$\exists r.C$	$\{x \mid \exists y.(x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
universal restriction	$\forall r.C$	$\{x \mid \forall y.(x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
at-least restriction	$\geq nr.C$	$\{x \mid \lvert\{ y \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}} \}\rvert \geq n\}$
at-most restriction	$\leq nr.C$	$\{x \mid \lvert\{ y \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}} \}\rvert \leq n\}$
fills restriction	$\exists r.\{a\}$	$\{x \mid (x, a^{\mathcal{I}}) \in r^{\mathcal{I}}\}$
local reflexivity	$\exists r.Self$	$\{x \mid (x,x) \in r^{\mathcal{I}}\}$
transitive role	r transitive	$r^{\mathcal{I}}$ transitive
role inclusion	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
inverse role	r^{-}	$\{(b,a) \mid (a,b) \in r^{\mathcal{I}}\}$
complex role inclusion	$r_1 \circ \cdots \circ r_n \sqsubseteq s$	$r_1^{\mathcal{I}} \circ \cdots \circ r_n^{\mathcal{I}} \subseteq s$

The semantics of a DL knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is defined by *models*, which are interpretations that *satisfy* both $\mathcal{T}$ and $\mathcal{A}$, written as $\mathcal{I} \models \mathcal{K}$. Generally, the TBox $\mathcal{T}$ contains only concept inclusions axioms because concept equivalences can be seen as abbreviations of concept inclusions in both directions, i.e., $C \equiv D$ is the same as $\{C \sqsubseteq D, D \sqsubseteq C\}$. Therefore, an interpretation $\mathcal{I}$ satisfies $\mathcal{T}$ if for each axiom $C \sqsubseteq D$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds. For the ABox $\mathcal{A}$, $\mathcal{I}$ satisfies a concept assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and satisfies a role assertion $r(a,b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. Under this definition, the interpretation in Example 3.2 is a model of the knowledge base in Example 3.1. This interpretation, however, contains more information than needed to justify the knowledge base. For example, it states that WEI has the child ALICE and BOB is a Male. These additional statements are allowed due to the *open-world assumption* (OWA) of DLs. OWA indicates that some information might be missing from the current ABox. In other words, new facts are acceptable if they do not contradict the current knowledge.

The reasoning services of DLs are mainly concerned about the following problems [60][3]:

Definition 3.3 (Reasoning problems in DL)**.** Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a DL knowledge base, C and D be concepts, and **I** be the set of individual names in $\mathcal{K}$.

(i) **Satisfiability:** C is satisfiable with respect to $\mathcal{T}$ if there is a model $\mathcal{I}$ of $\mathcal{T}$ and $C^{\mathcal{I}} \neq \emptyset$.

[3]Note that some notions used here are taken from [39], which are more compact that the ones in [60].

(ii) **Consistency:** $\mathcal{K}$ is consistent if $\mathcal{K}$ has a model.

(iii) **Subsumption:** C is subsumed by D with respect to $\mathcal{T}$, written $C \sqsubseteq_{\mathcal{T}} D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$.

(iv) **Equivalence:** C and D are equivalent with respect to $\mathcal{T}$, written as $C \equiv_{\mathcal{T}} D$, if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$.

(v) **Instance Check:** an individual $a \in \mathbf{I}$ is an instance of C with respect to $\mathcal{K}$, written $\mathcal{K} \models C(a)$, if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$.

(vi) **Instance Retrieval:** an individual a belongs to the retrieval $R_{\mathcal{A}}(C)$ with respect to $\mathcal{K}$ if $a \in \mathbf{I}$ and $\mathcal{K} \models C(a)$. □

The reasoning problems (i) to (iv) are called terminological reasoning that focuses on concepts. On the contrary, the problems (v) and (vi) are referred to as assertional reasoning that cares about individuals. Assertional reasoning plays a vital role in this thesis since it can be used to pose questions to the knowledge base. For example, if the knowledge base in Example 3.1 is extended with the axiom 3.6, one can ask for the children of Yingbing by retrieving all instances that belong to the new concept $\mathsf{ChildOfYingbing}$, which is defined as follows:

$$\mathsf{ChildOfYingbing} \equiv \mathsf{Person} \sqcap \exists \mathsf{hasParent}.\{\mathsf{YINGBING}\} \tag{3.8}$$

An inference engine would answer the instance retrieval with the individual ALICE, although there is no explicit information expressed as $\mathsf{hasParent}(\mathsf{ALICE}, \mathsf{YINGBING})$.

3.1.3. OWL

DLs provide the formal means to model knowledge and are adopted for the development of the ontology language OWL. There have been two versions of OWL: the first one was standardized in 2004, and the second one, OWL 2, was released in 2012. The design of OWL was also influenced by RDF(S), which led to the RDF-based semantics[4]. However, it has been shown that the reasoning problems in Definition 3.3 are, in general, not decidable for OWL ontologies under the RDF-based semantics. In other words, there is no algorithm that guarantees to terminate while solving reasoning problems. The OWL 2 standard also specifies the *direct semantics* of OWL, which is compatible with the DL $\mathcal{SROIQ}(D)$. Therefore, OWL 2 ontologies interpreted under direct semantics are also called OWL 2 DL ontologies.

Besides the compatible semantics, there are slight differences in the terminology between OWL and DL: a DL concept is called a *class* in OWL, and a DL role is called a *property* in OWL. Because of the support of data types, one distinguishes between OWL *object properties* and *data properties*, and the latter corresponds to the concrete roles in DL.

OWL 2 specifies several serialization formats. The RDF/XML syntax is based on the mapping between OWL 2 and RDF graphs[5] and is mandatory for all OWL 2 tools. The

[4]OWL ontologies under RDF-based semantics are also called OWL (2) Full.

[5]`https://www.w3.org/TR/2012/REC-owl2-mapping-to-rdf-20121211/`

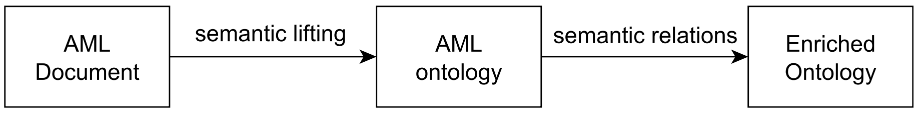


Figure 3.2.: Semantic lifting and enrichment of AML data.

functional-style syntax is used to define OWL 2 in its W3C specifications and provides foundations for the implementation of OWL 2 APIs and tools. Essentially, most parts of OWL 2 are a syntactic variant of $\mathcal{SROIQ}(D)$. Table 3.2 summarizes some OWL 2 constructors and axioms in the functional-style syntax and shows the mapping from OWL 2 to DL. OWL 2 also provides some non-DL features. For example, annotations are introduced for adding comments to OWL classes, properties, and individuals. For convenience, the remaining of the thesis uses the term “OWL" for OWL 2 DL and the term “OWL ontology” for “OWL 2 DL ontology”, if it is not otherwise stated explicitly.

3.1.4. Semantic lifting of AutomationML

In contrast to the formal model-theoretical semantics provided by OWL, Section 2.2 showed that AML only supports human-interpretable semantics. During the last decade, several studies have investigated the possibility of lifting AML data into an OWL ontology. The motivation of the semantic lifting is the potential of the automated inference that can be exploited in various applications, including model-driven software engineering [21], plant fault analysis [18], knowledge integration for robotics [61], and ontology-based data integration [62]. Fig. 3.2 shows the procedure of semantic lifting and enrichment of AML. The enriched ontology contains additional semantic information about the data, which is usually added by ontology experts for specific engineering purposes.

The first result of converting AML to OWL appeared in 2009 by Runde et al. in their German paper [16]. Two approaches were proposed and discussed. The abstract approach represents the CAEX vocabulary directly as OWL classes in the ontology and transforms CAEX classes, objects, and attributes as individuals of these OWL classes. The concrete approach generates an OWL class for each CAEX class with an annotation about its original type in the CAEX schema. For example, an AML role class Robot will be converted to an OWL class with the annotation RoleClass. Subsequent researches generally follow either the abstract or the concrete approach. For example, Kovalenko et al. proposed a lightweight ontology for covering core concepts of CAEX using the abstract approach [20], while Hua et al. followed the concrete approach for describing robotic components and systems [21].

The backward transformation from OWL to AML is less studied, although the first approach was already published in 2010 [17]. In this work, atomic OWL classes are mapped to appropriate CAEX classes using the CAEX type annotation of each OWL class. Then individuals of the top-level OWL classes are transformed into proper CAEX objects. Finally, the transformation handles each property associated with the individuals until all information in OWL is processed.

The existing approaches only target “simple” knowledge types, that is, atomic classes, objects, and properties. For handling complex ontological knowledge, e.g., OWL complex classes, one challenge arises that no regular AML model can preserve complex ontological

Table 3.2.: Mapping between the OWL 2 functional-style syntax and the DL syntax.

	OWL 2 Functional-Style Syntax	DL Syntax
OWL 2 class constructors	owl : Thing	$\top$
	owl : Nothing	$\bot$
	ObjectIntersectionOf($C_1 \cdots C_n$)	$C_1 \sqcap \cdots \sqcap C_n$
	ObjectUnionOf($C_1 \cdots C_n$)	$C_1 \sqcup \cdots \sqcup C_n$
	ObjectComplementOf(C)	$\neg C$
	ObjectOneOf$\{a, b, ...\}$	$\{a\} \sqcup \{b\} \sqcup \cdots$
	ObjectSomeValuesFrom(r C)	$\exists r.C$
	ObjectAllValuesFrom(r C)	$\forall r.C$
	ObjectMinCardinality(n r C)	$\geq nr.C$
	ObjectMaxCardinality(n r C)	$\leq nr.C$
	ObjectExactCardinality(n r C)	$= nr.C$
	ObjectHasValue(r a)	$\exists r.\{a\}$
	ObjectHasSelf(r)	$\exists r.Self$
OWL 2 data property class constructors	DataSomeValuesFrom(r DR)	$\exists r.(DR)$
	DataAllValuesFrom(r DR)	$\forall r.(DR)$
	DataMinCardinality(n r DR)	$\geq nr.(DR)$
	DataMaxCardinality(n r DR)	$\leq nr.(DR)$
	DataExactCardinality(n r DR)	$= nr.(DR)$
	DataHasValue(r lt)	$\exists r.\{lt\}$
OWL 2 property axioms	SubObjectPropertyOf(r s)	$r \sqsubseteq s$
	ObjectPropertyDomain(r C)	$\exists r.\top \sqsubseteq C$
	ObjectPropertyRange(r C)	$\top \sqsubseteq \forall r.\top$
	EquivalentObjectProperties($r_1, \cdots, r_n$)	$\cup_{i \neq j}\{r_i \sqsubseteq r_j\}$
	InverseObjectProperties(r s)	$r \equiv s^-$
	TransitiveObjectProperty(r)	r transitive
	FunctionalObjectProperty(r)	$\top \sqsubseteq (\leq 1r)$
OWL 2 class axioms and assertions	SubClassOf(C D)	$C \sqsubseteq D$
	EquivalentClasses($C_1 \cdots C_n$)	$\cup_{i \neq j}\{C_i \sqsubseteq C_j\}$
	DisJointClasses($C_1 \cdots C_n$)	$\cup_{i \neq j}\{C_i \sqsubseteq \neg C_j\}$
	SameIndividual($a_1 \cdots a_n$)	$\cup_{i \neq j}\{a_i = a_j\}$
	DifferentIndividuals($a_1 \cdots a_n$)	$\cup_{i \neq j}\{a_i \neq a_j\}$
	ClassAssertion(C a)	$C(a)$
	ObjectPropertyAssertion(r a b)	$r(a, b)$
	DataPropertyAssertion(r a lt)	$r(a, lt)$

semantics. In Section 5.1, an approach is proposed for the translation between OWL complex classes and AML models.

3.2. Foundations of Concept Learning

The research area of concept learning devotes to develop supervised machine learning algorithms for inducing concept definitions from labeled data. In DL (OWL), such concept definitions are complex concept descriptions (class expressions). For each concept C of interest, positive data examples represent members of C while negative ones do not. The goal of a learning algorithm is to find a concept definition that correctly classifies the data examples while being as simple as possible. Formally, the learning problem in DL is defined as follows [39]:

Definition 3.4 (concept learning in description logics)**.** Let Target be a concept name and $\mathcal{K}$ be a knowledge base (not containing Target). Let $E = E^+ \cup E^-$ be a set of examples, where E^+ are the positives examples and E^- are the negative examples and $E^+ \cap E^- = \emptyset$. The learning problem is to find a concept $C \equiv \mathsf{Target}$ with $\mathcal{K} \cup C \models E^+$ and $\mathcal{K} \cup C \not\models E^-$. □

In Definition 3.4, a learned concept C is called a *hypothesis* of Target. A learned concept C *covers* an example $e \in E$ if e is an instance of C with respect to the knowledge base $\mathcal{K}$, i.e. $K \models C(e)$. A hypothesis C is *complete* if it covers all positive examples $e \in E^+$, is *consistent* if it does not cover any negative example $e \in E^-$, and is *correct* if it is both complete and consistent. Besides the correctness, another criteria of the solution is the length of the concept hypothesis. For concepts in $\mathcal{ALC}$, the length is defined as follows:

Definition 3.5 (length of an $\mathcal{ALC}$ concept)**.** Let A be an atomic concept, r be a role, and D, E be concepts in $\mathcal{ALC}$, the length operator $|\cdot|$ is defined as:

$$\begin{aligned}
|A| &= |\top| = |\bot| = 1 \\
|\neg D| &= |D| + 1 \\
|D \sqcup E| &= |D \sqcap E| = 1 + |D| + |E| \\
|\exists r.D| &= |\forall r.D| = 2 + |D|
\end{aligned}$$

3.2.1. Refinement Operators

Refinement operators formulate the theoretical foundation of concept learning in DL. One can distinguish between downward and upward refinement operators that either specialize the most general concept $\top$ or generalize the most specific one $\bot$. Following a learning-by-searching paradigm, concept learning algorithms use the refinement operator to traverse the concept space and generate suitable hypotheses for the current learning problem. In the literature of learning in DL, the subsumption relation $\sqsubseteq$ is usually taken as a quasi-ordering of the search space. Based on $\sqsubseteq$, a downward (upward) refinement operator specializes (generalizes) a concept C to C' with $C' \sqsubseteq C$ ($C \sqsubseteq C'$), respectively. This thesis focuses on top-down learning algorithms, which can be characterized as follows:

Definition 3.6 (properties of downward refinement operators)**.** A refinement operator ρ for a DL language $\mathcal{L}$ is called:

- **complete** if for all concepts C, D with $C \sqsubset D$, there is a concept E with $E \in \rho(D)$ and $E \equiv C$.
- **weakly complete** if for all concepts $C \sqsubset \top$, there is a concept E with $E \in \rho(\top)$ and $E \equiv C$.
- **(locally) finite** if for all concepts C, $\rho(C)$ is finite.
- **proper** if for all concepts C, D with $D \in \rho(C)$, $C \not\equiv D$.

Among all the properties, completeness is the most critical one since it guarantees that given enough time, all possible subsuming concepts C can be reached from D. In contrast, incomplete refinement operators may fail to find the solution at all. Fig. 3.3 shows the definition of the first complete refinement operator ρ for $\mathcal{ALC}$, proposed by Lehmann and Hitzler in [63]. Additionally, ρ does not reduce the length of a concept, i.e. $\forall D \in \rho(C) : |D| \geq |C|$. In the same paper, the authors extended ρ to the proper refinement operator ρ^{cl}, which in turn, is extended to cover qualified number restrictions and concrete domains in OWL ontologies.

3.2.2. Learning Algorithms and Systems

The most related work on top-down concept learning in description logics is the DL-Learner framework [64], which, to the best of the author's knowledge, represents the state-of-the-art in non-parallelized concept learning algorithms in description logics. Besides other practical features e.g., knowledge fragment segmentation and approximate coverage test, DL-Learner implements two algorithms OCEL [63] and CELOE [65] for learning concepts in OWL. In particular, CELOE is a top-down learning algorithm using the operator ρ^{cl}, as described in the last section. CELOE employs a heuristic to evaluate the quality of concept hypotheses and iteratively searches for a better solution. To handle the infiniteness of ρ^{cl}, CELOE follows the iterative widening [66] approach to span a search tree of concept hypotheses and adopts a simple greedy strategy to select the most promising tree node for expansion. The expansion is controlled by a length upper bound of new refinements that is increased successively during a revisit of the same tree node. However, this simple greedy selection strategy does not utilize the structure of the search tree well and may rapidly lead to a local optimum. Another problem of CELOE is the number of parameters used to implement the heuristic. Section 4.1 elaborates more details of CELOE, especially on the procedure of search tree construction. Because instance checks under the Open-World Assumption (OWA) are costly, DL-Learner adopts a partially closed world assumption (CWA) and implements a reasoner for the approximation of the coverage of hypotheses. Moreover, Lehmann showed that only CWA allows learning universal restrictions, negations, and cardinality restrictions [39].

Further approaches that make explicit use of refinement operators exist. Badea et al. proposed a refinement operator for top-down concept learning in the language $\mathcal{ALER}$ and showed a basic learning procedure based on this operator [67]. YingYang is a learning system for $\mathcal{ALC}$ [68, 69, 70]. In contrast to a standard refinement-based approach as DL-Learner, YingYang uses an upward refinement operator to generalize

$$\rho(C) = \begin{cases} \{\bot\} \cup \rho_\top(C) & \text{if } C = \top \\ \rho_\top(C) & \text{otherwise} \end{cases}$$

$$\rho_B(C) = \begin{cases} \emptyset & \text{if } C = \bot \\ \{C_1 \sqcup \dots \sqcup C_n \mid C_i \in M_B \ (1 \leq i \leq n)\} & \text{if } C = \top \\ \{A' \mid A' \in sh_\downarrow(A)\} & \text{if } C = A \ (A \in N_C) \\ \quad \cup \{A \sqcap D \mid D \in \rho_B(\top)\} & \\ \{\neg A' \mid A' \in sh_\uparrow(A)\} & \text{if } C = \neg A \ (A \in N_C) \\ \quad \cup \{\neg A \sqcap D \mid D \in \rho_B(\top)\} & \\ \{\exists r.E \mid A = ar(r), E \in \rho_A(D)\} & \text{if } C = \exists r.D \\ \quad \cup \{\exists r.D \sqcap E \mid E \in \rho_B(\top)\} & \\ \quad \cup \{\exists s.D \mid s \in sh_\downarrow(r)\} & \\ \{\forall r.E \mid A = ar(r), E \in \rho_A(D)\} & \text{if } C = \forall r.D \\ \quad \cup \{\forall r.D \sqcap E \mid E \in \rho_B(\top)\} & \\ \quad \cup \{\forall r.\bot \mid & \\ \qquad D = A \in N_C \text{ and } sh_\downarrow(A) = \emptyset\} & \\ \quad \cup \{\forall s.D \mid s \in sh_\downarrow(r)\} & \\ \{C_1 \sqcap \dots \sqcap C_{i-1} \sqcap D \sqcap C_{i+1} \sqcap \dots \sqcap C_n \mid & \text{if } C = C_1 \sqcap \dots \sqcap C_n \\ \qquad D \in \rho_B(C_i), 1 \leq i \leq n\} & (n \geq 2) \\ \{C_1 \sqcup \dots \sqcup C_{i-1} \sqcup D \sqcup C_{i+1} \sqcup \dots \sqcup C_n \mid & \text{if } C = C_1 \sqcup \dots \sqcup C_n \\ \qquad D \in \rho_B(C_i), 1 \leq i \leq n\} & (n \geq 2) \\ \quad \cup \{(C_1 \sqcup \dots \sqcup C_n) \sqcap D \mid & \\ \qquad D \in \rho_B(\top)\} & \end{cases}$$

Figure 3.3.: The definition of the top-down refinement operator ρ proposed by Lehmann and Hitzler. Initially, ρ delegates to an operator ρ_B with $B = \top$. The subscript B is used to filter out concepts during the refinements of existential and universal restrictions (case 5 and 6). For more details of the operator, please refer to [63].

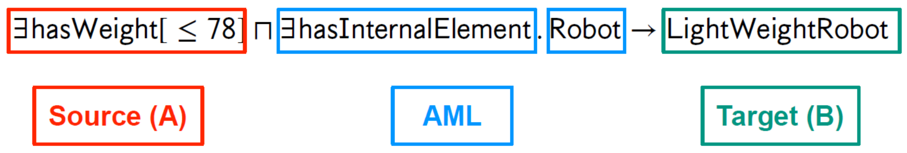

∃hasWeight[≤ 78] ⊓ ∃hasInternalElement.Robot → LightWeightRobot
Source (A)
AML
Target (B)

Figure 3.4.: Concept Learning for AutomationML.

approximated MSCs (most specific concepts). Afterwards, the so-called counterfactuals and a non-deterministic downward refinement operator are used to remove overly generalized hypotheses. Because MSCs need to be approximated for $\mathcal{ALC}$ and more expressive DL languages, YINGYANG tends to produce unnecessarily long concepts [39].

PArCEL and SPArCEL are recent works on learning in OWL and are built on top of DL-Learner [71, 72]. More specifically, both algorithms are parallelized by adopting a separate-and-conquer strategy. While both PArCEL and SPArCEL show the great advantage of parallelism, they use DL-Learner as a subroutine for the top-down refinement.

DL-FOIL is another DL learning system that adopts the separate-and-conquer strategy [73, 74]. As opposed to PArCEL and SPArCEL, DL-FOIL is a DL variant of the FOIL algorithm [75]. The inner loop of DL-FOIL uses a refinement operator for specializing partial solutions of the learning problem, while the outer loop combines the partial solutions with disjunctions. The specialization procedure acts as a hill climbing search with a predefined upper bound of refinement steps. Therefore it does not construct a search tree as with CELOE.

3.2.3. Relevance to AutomationML

Concept learning can be seen as an approach for ontology engineering [39], where the task is to find a definition for a new concept using the vocabulary of an existing ontology. Consider the AML ontology as illustrated in Fig. 3.2 and suppose that the AML ontology contains the data property `hasWeight`, the object property `hasInternalElement`, and the atomic class `Robot`. For a new concept, `LightWeightRobot`, a possible definition based on the AML ontology is illustrated in Fig. 3.4. Suppose that the AML ontology is generated from the data of a source engineering tool A and `LightWeightRobot` is a private role class within the target engineering B, then the concept definition in Fig. 3.4 bridges the semantic gap between A and B. This kind of semantic information is crucial in ontology-based data access or integration, and is usually modeled by an ontology expert. However, ontology experts are often not effectively available in practice and are argued to be inefficient for building industry-scale heavyweight ontologies because they lack domain-specific knowledge [76]. Therefore, one aim of the thesis is to develop a machine learning system that supports the domain experts for finding such concept definitions. Nevertheless, several research questions arise while applying concept learning for AML.

First, what kind of semantic lifting is appropriate for the learning task? This question is not only a design choice but also strongly related to the trade-off between learning performance and the expressiveness of concept definition. Section 4.3 presents an approach that is dedicated to achieving a balance between efficiency and quality.

Second, because AML imposes certain modeling constraints, as described in Section 2.1, the existing general-purpose learning algorithm may not scale well as the size of the AML data increases. To this end, Section 4.3.1 introduces an AML-specific refinement operator that can significantly increase the learning performance by exploiting the modeling constraints of AML. In addition to that, Section 4.2 proposes the Rapid Restart Hill Climbing (RRHC) algorithm that outperforms the CELOE algorithm in general learning tasks.

Finally, to better involve the domain experts in the learning procedure, Chapter 5 presents the AMLLEARNER framework that allows the user to communicate with the learner by representing the learned OWL class in AML. The user can edit such AML models while AMLLEARNER is able to react to the changes made by the user and restart the learning using the AML model as an initial guess.

4. Concept Learning

This chapter presents algorithms for concept learning in OWL and AML. Section 4.1 conducts an in-depth study of CELOE, which reveals several drawbacks of the well-known algorithm, including its traversing strategy, usability, and performance. The results of the study inspired the design of the novel learning algorithm *Rapid Restart Hill Climbing* (RRHC), which is presented in Section 4.2. Afterward, Section 4.3 shows how to perform concept learning in AML, and proposes a new refinement operator ρ^{aml} that utilizes the modeling constraints of AML for improving the performance. Finally, Section 4.4 shows the experimental results of RRHC and ρ^{aml} by comparing them with the CELOE algorithm and the original refinement operator ρ^{cl}.

4.1. A Comparative study of CELOE

CELOE is a top-down learning algorithm that employs the refinement operator ρ^{cl}, as described in Section 3.2.1. Because ρ^{cl} is infinite, it is impossible to evaluate all hypotheses at once. Therefore, CELOE follows the iterative widening [66] approach to span a search tree of concept hypotheses successively. In each iteration, the following steps are performed:

1. **Selection**: Find a tree node n with the maximum heuristic score in the search tree. CELOE implements this by using a global priority queue of all tree nodes, sorted by their score.
2. **Refinement**: Invoke ρ^{cl} to generate new refinements of the node n. Since ρ^{cl} is infinite, CELOE restricts the number of refinements per step using a length upper bound, which is increased by one for each revisit of n. This length upper bound is called the *horizontal expansion* of n. Note that because the horizontal expansion limits the length of refinements, there might be no refinement for a certain iteration.
3. **Expansion**: New refinements generated in the last step are tested against redundancy and completeness. A refinement C is ignored if the search tree already contains a concept C' which is weakly equal to C (redundant), or C does not cover all positive examples E^+ (too weak). Refinements that are neither redundant nor too weak are added as child nodes to the current node n, and CELOE starts over with the selection step.

The basic form of the heuristic employed for assessing the quality of a node n was first proposed in [39] as follows:

$$score(n) = accuracy(C) + \alpha \cdot acc\_gain(n) - \beta \cdot he \quad (\alpha \geq 0, \beta \geq 0) \tag{4.1}$$

In Equation 4.1, C is the concept hypothesis of node n, acc_gain is the accuracy gain compared with n's parent, and he is the horizontal expansion. The two parameters α, β

control the expansion behavior of the search tree: larger α tends to exploit better hypotheses while larger β favors less explored areas. The accuracy of a hypothesis can be defined in several ways. Equation 4.2 shows the formula of *predictive accuracy*, where fp and fn stand for false positives and false negatives, respectively. Equation 4.3 shows the well known F-measure (or F_1 score) that is based on precision and recall.

$$pred\_acc(C) = 1 - \frac{fp + fn}{|E|} \tag{4.2}$$

$$F_1(C) = 2 \cdot \frac{precision \cdot recall}{percision + recall} \tag{4.3}$$

One major drawback of CELOE is that the selection phase handles the search tree as a single priority queue, and the *global* best tree node is always selected for further refinement. The term *global* shall be emphasized since CELOE does not traverse the search tree in any form, but keeps tracking the best node in the queue. While this simple greedy approach is fast, it neglects the structure of the search tree and may rapidly lead to a local optimum in the lower part of the tree. Unfortunately, there are often many more nodes in the lower part of the tree, so that significant effort is required to regret previous radical decisions.

The concrete implementation of CELOE tries to handle this problem by using two additional parameters in the heuristic. One is the *start node bonus* that gives extra value to the start node $\top$ for a hopefully sufficient exploration in the upper part of the tree, and another one is the *refinement penalty* that penalizes tree nodes proportionally to the number of their child nodes. Nevertheless, consider the combination with the gain bonus and expansion penalty, finding a proper setting of these parameters is laborious in practice. More importantly, different learning problems do not share these parameters. Consider the Uncle example from the family benchmark provided by DL-Learner. With the default configuration, CELOE was able to find the following solution in 4s 301ms.

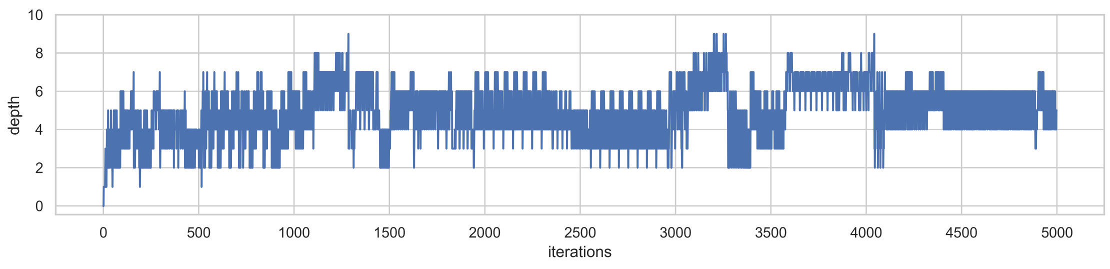
$$\begin{aligned}\mathsf{Uncle} \equiv\ &((\exists \mathsf{hasSibling}.(\exists \mathsf{hasChild}.\top)) \sqcup \\ &(\exists \mathsf{married}.(\exists \mathsf{hasSibling}.(\exists \mathsf{hasChild}.\top)))) \sqcap (\neg \mathsf{Female})\end{aligned} \tag{4.4}$$

This solution is correct and reads, "An uncle is <u>not a female</u> and <u>has a sibling who has a child</u> or <u>is married to someone that has a sibling who has a child</u>." To better illustrate the learning process, an ID is assigned for each node in the search tree as follows.

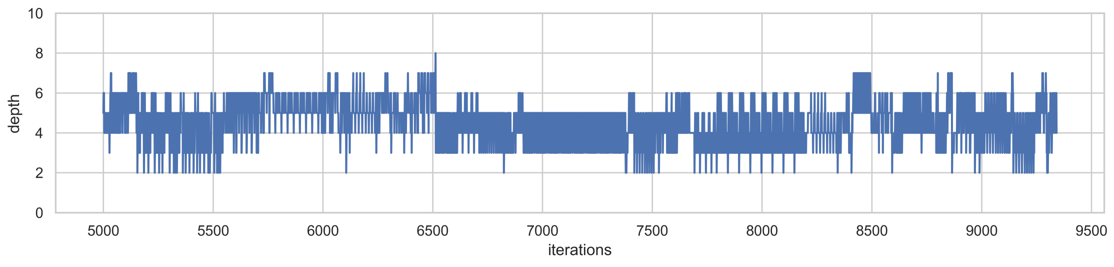
Definition 4.1 (ID of a tree node)**.** An ID of a tree node n describes its position in the search tree. It is a sequence of position numbers connected by the hyphen as P_0-P_1-P_2-$\cdots$. Each position number P_i represents the index of the branch located at the depth level i that contains n as a descendant node.

For example, the solution in Equation 4.4 has the ID `0-2-52-9-17-9`, which means that it is the 9th child of its parent `0-2-52-9-17`. The latter in turn, is the 17th child of its parent `0-2-52-9`.

Fig. 4.1 shows the tree depth of expanded nodes for learning the concept of Uncle with CELOE. First, it can be observed that most expansions occurred at a level lower than



(a) Iterations up to 5000.



(b) Iterations from 5000.

Figure 4.1.: The tree depth of expanded nodes for learning the concept of Uncle with the default setting of CELOE.

3 (depth larger than 3). In particular, from the iteration 1000 to 9150, CELOE visited level 2 very rarely and put much effort to search from level 3 to 6. After this extensive searching of lower levels, the algorithm realized that a better solution originated in the less explored areas from the upper part of the tree and therefore started to explore level 2 from the iteration 9150. Indeed, a good seed for the solution was the tree node `0-2-52` with the hypothesis $(\exists \mathsf{hasSibling}.\top \sqcup \exists \mathsf{married}.\top) \sqcap (\neg \mathsf{Female})$. As the ID indicates, `0-2-52` was the 52nd child of the level 1 hypothesis $\neg\mathsf{Female}$, while $\neg\mathsf{Female}$ itself was the second child of $\top$. CELOE came back to `0-2-52` in the iteration 9303 and was able to find the solution within 40 further iterations.

To demonstrate the effects of different parameter settings, Table 4.1 compares the performance of various CELOE configurations for the Uncle example. The configuration `ori` is the default setting provided by DL-Learner. `ori` has different expansion penalty for individual learning problems and is set to 0.02 for the Uncle problem. The configuration `spa` is suggested by Tran et al. in [72]. Table 4.1 also contains three variants of `ori`: `ori_v1` and `ori_v3` are intended to test the influence of the refinement penalty and the start bonus, while `ori_v2` slightly modifies the expansion and refinement penalty.

For each of these configurations, Table 4.2 summarizes measurements collected until the first correct solution was found. On the one hand, while `ori`, `ori_v1` and `ori_v3` had comparatively good performance, they still differ in the number of tested expressions and in the tree depth. In particular, the results of `ori_v1` and `ori_v3` indicated that the refinement penalty and the start bonus were unnecessary for learning the definition of Uncle. However, as shown later in Section 4.4, it is not always the case for other learning problems. More importantly, `ori_v1` and `ori_v3` found a slightly better solution than

configuration	exp. penalty	gain bonus	ref. penalty	start bonus
`ori`	0.02/0.1	0.3	0.0001	0.1
`spa`	0.05	0.2	0.0001	0.1
`ori_v1`	0.02/0.1	0.3	0	0
`ori_v2`	0.03	0.3	0.00005	0.1
`ori_v3`	0.02	0.2	0	0
`rrhc-default`	0.02	0.2	0	0

Table 4.1.: The configurations of learning algorithms used for comparison. Note that both `ori_v3` and `rrhc-default` do not use the parameters refinement penalty and start bonus by setting them to 0.

	time(s)	#iterations	#expressions	#tree nodes	tree depth	length
`ori`	4.301	9342	16584	6655	9	16
`spa`	177.863	429207	526022	322902	12	15
`ori_v1`	4.550	9346	10912	7088	10	15
`ori_v2`	42.808	55206	115141	46546	11	16
`ori_v3`	4.325	9342	10457	6655	10	15

Table 4.2.: The learning performance of five different CELOE configurations for the Uncle example. All statistics are collected until the first correct solution is found.

`ori` that replaced $\neg$Female in Equation 4.4 with Male. On the other hand, the terrible performance of both `ori_v2` and `spa` suggested that the (slightly) higher expansion penalty is fatal in the Uncle example.

4.2. Rapid Restart Hill Climbing

For tackling the problems of CELOE, this section presents the Rapid Restart Hill Climbing (RRHC) algorithm that selects a node for expansion by traversing the search tree in a hill climbing fashion. RRHC uses the same heuristic as CELOE, and the default parameter setting is depicted by the configuration `rrhc-default` in Table 4.1, which does not use the start node bonus and the refinement penalty. For the expansion, RRHC commits to the iterative widening approach for producing a finite set of refinements. Unlike conventional hill climbing techniques used in Inductive Logic Programming (ILP) [75], RRHC rapidly restarts with one-step backtracking after the expansion.

Algorithm 4.1 illustrates the skeleton of RRHC that has two nested loops. For a current node n, the inner loop selects a tree node for expansion in a hill-climbing manner (line 4-11) while the outer loop expands the selected node n using the refinement operator ρ^{cl}. The horizontal expansion of the node n is increased by one after the refinement (line 18), so that a revisit of n could generate new refinements. Because the heuristic in Equation 4.1 depends on the horizontal expansion, the score of n drops after each refinement such that n might not be the best one among its siblings. Therefore, the algorithm backtracks to n's parent (line 20) and rapidly restarts without checking the new refinements.

Fig. 4.2 compares the tree construction process of RRHC and CELOE. For the sake of simplicity, the Mother example from the family benchmark is used. The numbers on the

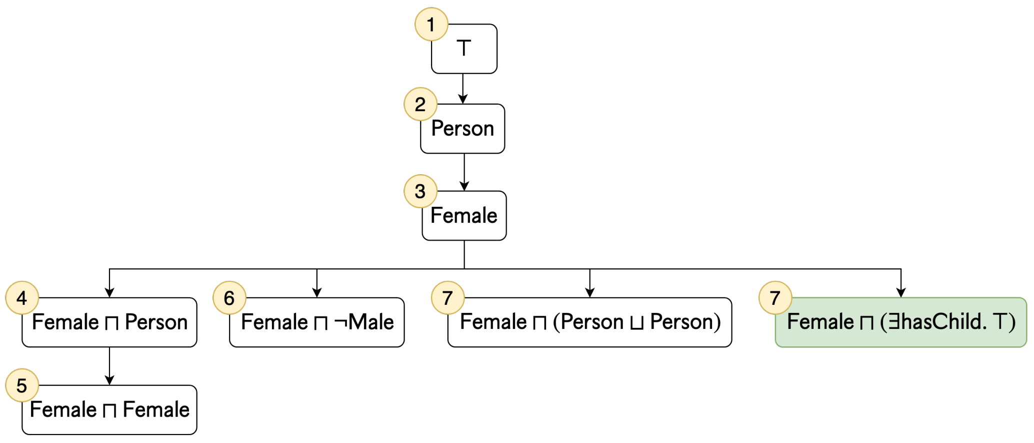
Algorithm 4.1 Rapid Restart Hill Climbing (RRHC)

```
Input: background knowledge $\mathcal{K}$, positive examples $E^+$, negative examples $E^-$
Output: best concept found
 1: initialize a search tree $ST$ with the root node $(\top, |\top|)$
 2: let the current node $n = (C, he)$ be the root node with $C = \top, he = |\top|$
 3: while Solution not found or timeout not triggered do
 4:     while $size(n.children) \neq 0$ do
 5:         select a node $child$ with the best score among $n.children$
 6:         if $score(child) > score(n)$ then
 7:             $n \leftarrow child$
 8:         else
 9:             break
10:         end if
11:     end while
12:     let $refinements = \{D \mid D \in \rho^{cl}(C), |D| \leq he\}$
13:     for $D \in refinements$ do
14:         if $D$ is complete and $D$ is not redundant then
15:             add $(D, |D|)$ as a child of $n$
16:         end if
17:     end for
18:     increase the horizontal expansion of $n$ by $1$
19:     if $n \neq root$ then
20:         $n \leftarrow n.parent$
21:     end if
22: end while
23: return concept with best accuracy
```

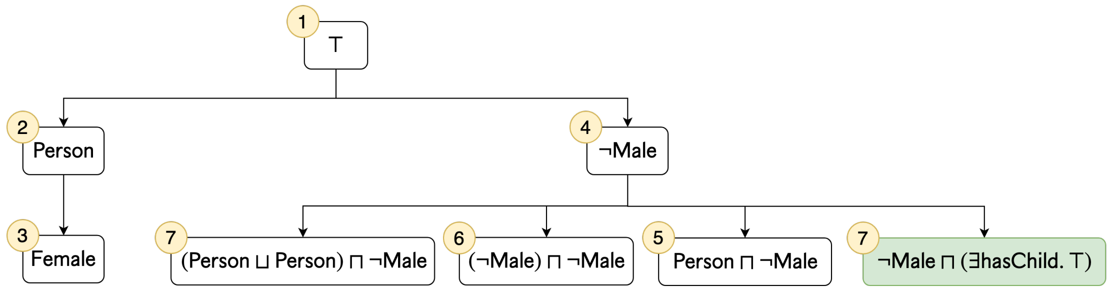
upper left corner of each tree node illustrate the sequence of refinement generation. The tree node marked as green depicts the solution found by the algorithms.

Two major differences can be observed between Fig. 4.2a and 4.2b. The first one is that CELOE committed to the refinement chain $\top \rightsquigarrow$ Person $\rightsquigarrow$ Mother $\rightsquigarrow \cdots$ while RRHC preferred $\neg$Male rather than Person (the symbol $\rightsquigarrow$ reads “is refined to” and refers to a refinement step). In fact, RRHC also generated Female in the early phase but decided to go back to the upper layer instead of exploiting the high score of Female, because the score of Person decreased after one refinement and did not show any advantage against the root node. After discovering the node $\neg$Male, RRHC insisted on refining it since its score is higher than Person, even after three times of expansion (step 5 to 7). One reason is that $\neg$Male has the same individuals (instances) as Female, although their semantic equivalence is not explicitly stated in the knowledge base. As a result, the tree constructed by RRHC is shallower than the one of CELOE. Indeed, RRHC often generates a much smaller tree than CELOE, which also indicates that RRHC is more efficient than CELOE (see the experiments in Section 4.4).

The second difference is that CELOE found a better solution than RRHC regarding the concept length. However, as shown in Table 4.2, the solution found by CELOE depends on its parameter setting. In fact, with the configurations `spa` and `ori_v2`, CELOE generated the same solution as RRHC.



(a) Search tree of the Mother example by CELOE.



(b) Search tree of the Mother example by RRHC.

Figure 4.2.: The search tree constructed for learning the concept of Mother with the default setting of CELOE and RRHC.

After explaining the learning procedure of RRHC, its theoretical properties needs to be analyzed. Proposition 4.1 shows that RRHC is correct in $\mathcal{ALC}$.

Proposition 4.1 (Correctness in $\mathcal{ALC}$)**.** If a learning problem has a solution in $\mathcal{ALC}$, then Algorithm 4.1 terminates and finds a correct solution.

Proof 4.1. The proof first repeats the correctness proof for OCEL/CELOE in [77], and then extends it for Algorithm 4.1.

Suppose that the learning problem has a solution D in $\mathcal{ALC}$, then the weak completeness of the refinement operator ρ^{cl} guarantees a refinement path in the form of $\top \rightsquigarrow D_1 \rightsquigarrow D_2 \rightsquigarrow \cdots \rightsquigarrow D_n = D$. The basic heuristic in Equation 4.1 has the property that $score(D) \geq -|D|$ for $\beta \in [0,1]$, since the first two terms in Equation 4.1 are positive, and the horizontal expansion of D is initialized to $|D|$. Moreover, $score(D_i) \geq -|D|$, since ρ^{cl} does not reduce the length of a refined hypothesis. On the other hand, because $\beta > 0$, a hypothesis D' with a sufficiently high horizontal expansion would have a score lower than $-|D|$ and would not exist in the chain above. As OCEL/CELOE greedily selects the global best node from the tree, D' would never be selected until all concepts in the chain above are sufficiently refined. Thus, either D or another solution will be found.

The proof above needs to be extended for Algorithm 4.1, since RRHC employs a different selection strategy. Essentially, it is sufficient to show that RRHC would not select a node D' that has no chance to refine to D, i.e. $score(D') < -|D|$. This can be

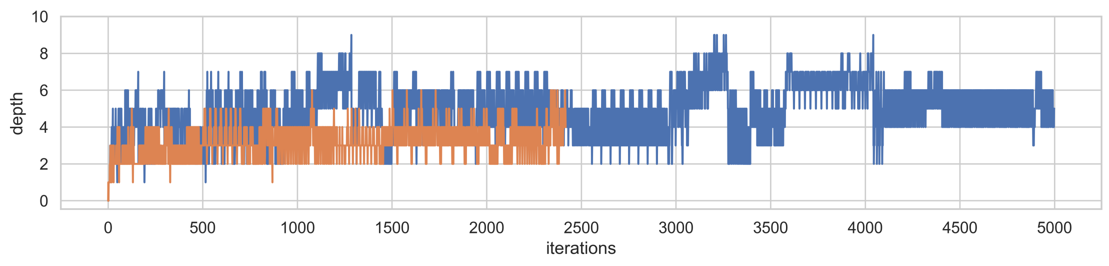


Figure 4.3.: Comparison of RRHC (orange) and CELOE (blue) regarding the expanded nodes in the Uncle example. Iterations from 5001 are omitted since RRHC found a solution in iteration 2418. Note that CELOE continues after iteration 5000 as shown in Fig. 4.1.

done by deriving a contradiction. Suppose that D' would be selected, then $-|D| > score(D') \geq score(D^*)$ for each D^* that is a sibling of D'. In other words, no sibling node of D' can be refined to the concept D. Furthermore, let D'_p be the parent of D', then $-|D| > score(D') > score(D'_p)$, since otherwise Algorithm 4.1 would choose D'_p and restart. Consider the layers above D', it is evident that the score of all predecessors of D' and the siblings of those are lower than $score(D')$ and therefore lower than $-|D|$, including the root node $\top$. As a consequence, even with sufficiently high horizontal expansion, ρ^{cl} can not generate D from $\top$, which leads to a contradiction of the weak completeness of ρ^{cl}. $\square$

Recall the Uncle example from the family benchmark. Fig. 4.3 shows the comparison of RRHC (orange) and CELOE (blue) regarding the tree depth of expanded nodes. Besides that RRHC tended to explore the upper part of the tree, its behavior was also more stable than CELOE regarding depth changes. However, it is worth noting that RRHC is fundamentally different to a breadth-first search, as an exhaustive expansion of any tree node is not viable due to the infiniteness of the refinement operator. Principally, the heuristic still guides the search. However, the restart mechanism requires a sequence of good candidates from the root node $\top$ to the selected one. Consequently, RRHC also spent quite much time in level 3 and 4, since the score of the node `0-2-52` was lower than its siblings. However, because the upper part of the tree has much fewer nodes than the lower part, RRHC was able to expand `0-2-52` in iteration 2382, compared with iteration 9303 in the case of CELOE.

4.3. Concept Learning in AutomationML

As described in Section 3.2.3, concept learning can be used to build the semantic mapping between a target-specific engineering concept and the vocabulary of a source engineering tool. Fig. 4.4 illustrates the main idea of learning in AML. The users Alice (expert of tool A), and Bob (expert of tool B) have their proprietary role class libraries that are not aligned beforehand. Alice generates AML system unit classes (hexagons) from the source system and Bob wants to align them with the target concepts. In practice, Bob often recognizes the desired engineering objects (or at least some of them) from Alice's system unit classes, for example, some products from a certain component supplier, and Bob knows their semantic definitions (ellipses) in the target system. Given

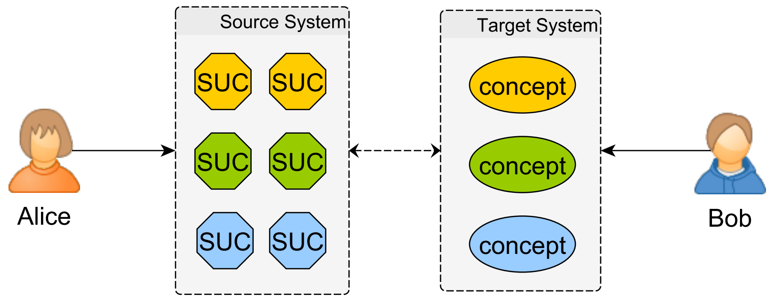


Figure 4.4.: Mapping system unit classes from source system to concepts in the target system. Same color between source and target system represents the semantic correspondence.

this information, concept learning can derive a formal representation of Bob's concepts using the terminology from Alice, thus close the semantic gap between them. Note that, because the concrete engineering data of Alice will be stored as AML internal elements which are supposed to be instances of the external interfaces, this approach also generalizes to the learning from internal elements.

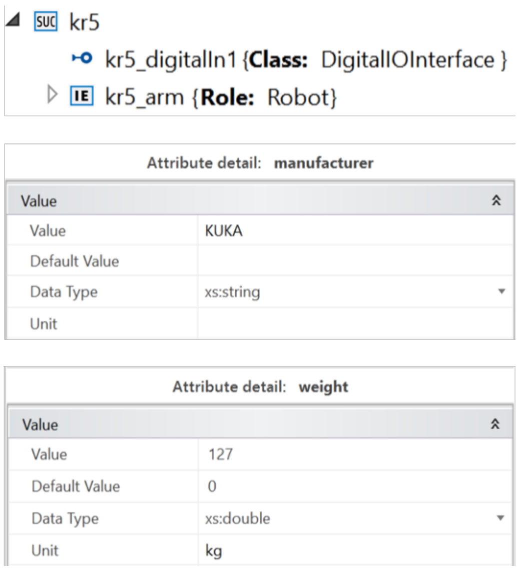
A prerequisite for the learning is an ontological representation of the data from the source system, that is, the semantic lifting from the source AML file to an AML ontology. From the existing methods as described in Section 3.1.4, the concrete approach from [16] is adopted since it is of great importance that domain concepts, e.g., Robot and ConveyorBelt, are treated as terminological classes to enable inference over the subsumption hierarchy. Table 4.3 shows an overview of the mappings from AML to OWL models. In the terminological layer (T-Box), AML role classes and interface classes, as well as their inheritance structures are transformed into OWL class hierarchies. In the axiomatic layer (A-Box), AML internal elements and external interfaces are transformed into OWL individuals. The major difference to [16] is that AML system unit classes are also converted to OWL individuals, which might seem to be inconsistent with the object-oriented paradigm of CAEX. Nevertheless, this is necessary to cope with the supervised setting of concept learning, since system unit classes are the *labeled data instances* for the learning from which a class definition shall be generated.

In the relational layer (R-Box), on the one hand, the relationships between system unit classes, internal elements, and external interfaces are transformed into OWL object properties, i.e., hasIE and hasEI. On the other hand, CAEX attributes are quite sophisticated since they possess a whole data structure, including the data type, unit, and default values, etc. Hua et al. [21] presented a way to represent this structure as an RDF graph completely, but it would be overwhelming for concept learning. Because the essential semantics of a CAEX attribute can be captured by its name, data type, and value, each distinct CAEX attribute is transformed to an OWL data property with its data type, and concrete attribute values are associated to the corresponding OWL individuals.

Consider the system unit class kr5 shown in Fig. 4.5. The system unit class has two CAEX attributes manufacturer and weight, an external interface of the interface class DigitalIOInterface, and an internal element of the role class Robot. The semantic lifting of kr5 generates the following assertions:

AM models	OWL models	Example
role class	class	Robot
interface class	class	DigitalIOInterface
system unit class	individual	KR5
internal element	individual	KR5_ARM
external interface	individual	KR5_DIGITALIN1
object relationship	object property	hasIE, hasEI
CAEX attribute	data property	hasWeight

Table 4.3.: Overview of the conceptual mapping from AML to OWL.



kr5
kr5_digitalIn1 {**Class:** DigitalIOInterface }
kr5_arm {**Role:** Robot}

Attribute detail: **manufacturer**

Value	
Value	KUKA
Default Value	
Data Type	xs:string
Unit	

Attribute detail: **weight**

Value	
Value	127
Default Value	0
Data Type	xs:double
Unit	kg

Figure 4.5.: The simplified system unit class of the robot kr5 in the AML editor.

$$
\begin{array}{l}
\mathsf{hasManufacturer(KR5, "KUKA")} \\
\mathsf{hasWeight(KR5, 127)} \\
\mathsf{hasEI(KR5, KR5\_DIGITALIN1)} \\
\mathsf{hasIE(KR5, KR5\_ARM)} \\
\mathsf{DigitalIOInterface(KR5\_DIGITALIN1)} \\
\mathsf{Robot(KR5\_ARM)}
\end{array}
$$

The workflow for concept learning in AML is illustrated in Fig. 4.6. First, the source system generates the system unit classes following the externalization approach proposed in [49]. For each concept from the target system, some system unit classes are selected as positive examples and some others as negative examples. After the semantic lifting of the AML file, the learning procedure is carried out with custom configurations, including the selected examples and the parameters of the learning algorithm. Finally, the results are presented and ordered by their simplicity (in length), as shown in Fig. 4.7. Note that because the learned class expressions are intended to be used in data integration and exchange tasks, only 100% correct solutions are considered in this thesis.

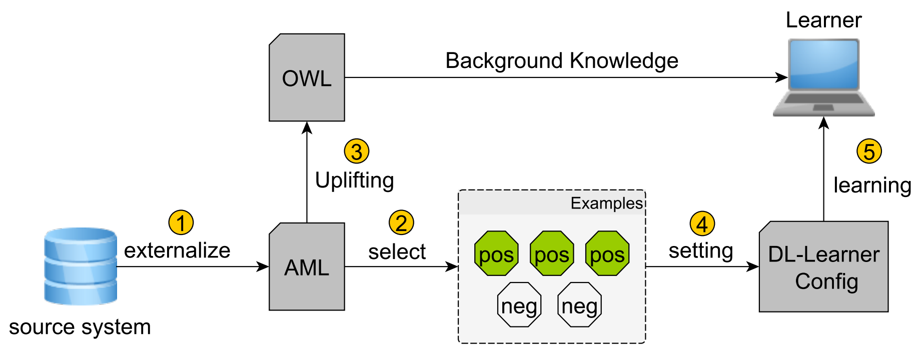


Figure 4.6.: Workflow for concept learning of AML system unit classes.

```
solutions:
1: Robot and (has_Frame_x some double[>= "-9.5"^^double]) (pred. acc.: 100.00%, F-measure: 100.00%)
2: Robot and (has_Frame_rz some double[<= "45.0"^^double]) (pred. acc.: 100.00%, F-measure: 100.00%)
3: Robot and (has_Frame_rz some double[<= "0.0"^^double]) (pred. acc.: 100.00%, F-measure: 100.00%)
4: Robot and (isIEOf some (hasIEDescendant some Structure)) (pred. acc.: 100.00%, F-measure: 100.00%)
5: Robot and (PhysicalPortList or (has_Frame_x some double[>= "-9.5"^^double])) (pred. acc.: 100.00%, F-measure: 100.00%)
6: Robot and (PhysicalPortList or (has_Frame_rz some double[<= "45.0"^^double])) (pred. acc.: 100.00%, F-measure: 100.00%)
7: Robot and (PhysicalPortList or (has_Frame_rz some double[<= "0.0"^^double])) (pred. acc.: 100.00%, F-measure: 100.00%)
8: Robot and (PhysicalNetwork or (has_Frame_x some double[>= "-9.5"^^double])) (pred. acc.: 100.00%, F-measure: 100.00%)
9: Robot and (PhysicalNetwork or (has_Frame_rz some double[<= "45.0"^^double])) (pred. acc.: 100.00%, F-measure: 100.00%)
10: Robot and (PhysicalDevice or (has_Frame_x some double[>= "-9.5"^^double])) (pred. acc.: 100.00%, F-measure: 100.00%)
```

Figure 4.7.: The output of a learning task.

4.3.1. A Refinement Operator for AutomationML

Until now, the refinement operator ρ^{cl} is used in all the learning problems. However, because ρ^{cl} is designed generally, it does not take into account the syntactic constraints defined in the CAEX schema and can be inefficient for learning in AML. Fig. 4.8 shows the interaction between different CAEX schema elements. Obviously, only external interfaces can reference interface classes, and each external interface can only reference one interface class. These constraints can be integrated into the refinement operator to improve the performance of learning, especially for a large T-Box. Therefore, the set of named concepts N_C is separated into two subsets, i.e., the set N_{ar} for all AML role classes and the set N_{ai} for all AML interface classes.

Recall the definition of ρ in Fig. 3.3. The idea is to adapt the refinements in place of role fillers, because it is desired to handle fillers of hasIE and hasEI differently. To this

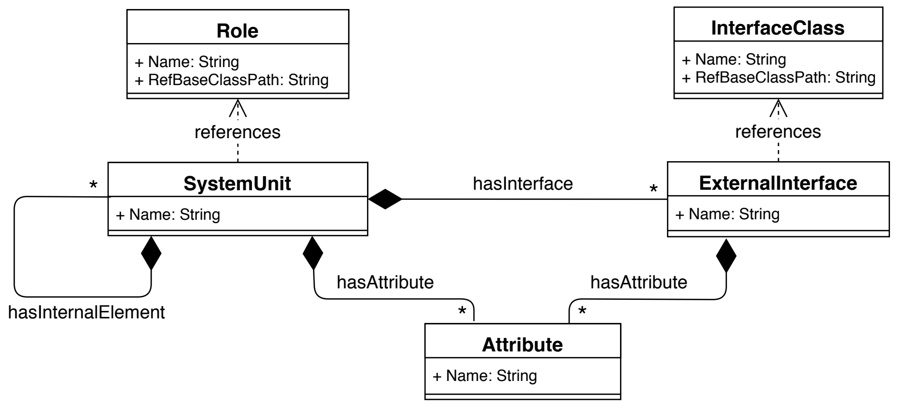


Figure 4.8.: The XML schema of AML. Some details are omitted for brevity.

end, several auxiliary sets are defined as follows:

$$M_{op} = \{\exists \mathsf{hasIE}.\top, \exists \mathsf{hasEI}.\top, \forall \mathsf{hasIE}.\top, \forall \mathsf{hasEI}.\top\}$$
$$M_{ie} = \{A \mid A \in N_{ar}, \nexists A' \in N_{ar} : A \sqsubset A'\}$$
$$M_{ei} = \{A \mid A \in N_{ai}, \nexists A' \in N_{ai} : A \sqsubset A'\}$$

M_{op} is the set of all existential and universal restrictions with the filler $\top$, M_{ie} is the set of top level AML role classes, and M_{ei} is the set of top level AML interface classes. Further, let $U_{ie} = \{C_1 \sqcup C_2 \sqcup ... \mid C_i \in M_{ie} \cup M_{op}\}$, $U_{ei} = \{C_1 \sqcup C_2 \sqcup ... \mid C_i \in M_{ei}\}$ and $sh_{\downarrow}(C)$ be the set of direct sub classes of a named concept $C \in N_C$, then ρ is adapted in the following cases:

(i) $\rho(C) = U_{ei}$, if $C = \top$ and C is a filler of hasEI

(ii) $\rho(C) = U_{ie}$, if $C = \top$ and C is not a filler of hasEI

(iii) $\rho(C) = sh_{\downarrow}(C)$, if $C \in N_C$ is a filler of hasEI

(iv) $\rho(C) = sh_{\downarrow}(C) \cup \{C \sqcap D | D \in \rho(\top)\}$, if $C \in N_C$ is not a filler of hasEI

Cases (i) and (ii) refines the top concept $\top$ and are complementary to each other. If $\top$ is a filler of an object property hasEI (case (i)), then only elements from the set U_{ei} is used for the refinement, because external interfaces can only refer to interface classes and cannot have nested object structures. For example, the following refinements cannot be produced in case (i):

$$\times \quad \exists \mathsf{hasEI}.\top \rightsquigarrow \exists \mathsf{hasEI}.\mathsf{AutomationMLBaseRole}$$
$$\times \quad \exists \mathsf{hasEI}.\top \rightsquigarrow \exists \mathsf{hasEI}.(\exists \mathsf{hasEI}.\top)$$
$$\times \quad \exists \mathsf{hasEI}.\top \rightsquigarrow \exists \mathsf{hasEI}.(\exists \mathsf{hasIE}.\top)$$

If $\top$ is not a filler of hasEI (case (ii)), elements from U_{ie} are candidates for the refinement because an AML system unit class and internal element can refer to role classes and have nested internal elements or external interfaces. In the following examples, the first two refinements are valid, but the third one cannot be produced in case (ii).

$$\checkmark \quad \exists \mathsf{hasIE}.\top \rightsquigarrow \exists \mathsf{hasIE}.\mathsf{AutomationMLBaseRole}$$
$$\checkmark \quad \exists \mathsf{hasIE}.\top \rightsquigarrow \exists \mathsf{hasIE}.(\exists \mathsf{hasIE}.\top)$$
$$\times \quad \exists \mathsf{hasIE}.\top \rightsquigarrow \exists \mathsf{hasEI}.\mathsf{AutomationMLBaseInterface}$$

Similarly, case (iii) and (iv) are complementary to each other for refining an atomic concept C. If C is a filler of hasEI, then only direct subclasses of C can be used for refinement because external interfaces can only refer to one interface class. In other words, conjunctions are not allowed as the filler of hasEI. For example, the following refinement cannot be produced in case (iii):

$$\times \quad \exists \mathsf{hasEI}.\mathsf{AutomationMLBaseInterface} \rightsquigarrow \exists \mathsf{hasEI}.(\mathsf{PPRConnector} \sqcap \mathsf{Communication})$$

If C is not a filler of hasEI (case (iv)), then C can be refined to either one of its direct subclasses or the conjunction with $\rho(\top)$, since system unit classes and internal elements

benchmark	language	#class	#ind	#op	#dp	#axiom	category
arch	$\mathcal{ALC}$	8	19	5	0	80	simple
carcinogenesis	$\mathcal{ALC}(\mathbf{D})$	142	22372	4	15	74566	hard
family	$\mathcal{AL}$	4	202	4	0	1343	simp./med.
forte	$\mathcal{ALIF}$	3	86	3	0	347	medium
lymphography	$\mathcal{AL}$	53	148	0	0	2197	simp./hard
moral	$\mathcal{ALC}$/$\mathcal{ALC}$	41/44	43/202	0/0	0/0	1047/4710	simple
mutagenesis	$\mathcal{AL}(\mathbf{D})$	86	14145	5	6	62066	hard
poker	$\mathcal{AL}$/$\mathcal{AL}$	2/2	311/347	6/6	0/0	1334/1418	simple
bible	$\mathcal{SHOIN}(\mathbf{D})$	49	724	29	9	4434	simple
trains	$\mathcal{ALC}$	10	50	5	0	288	simple
yinyang	$\mathcal{ALI}$	3	31	3	0	157	simple

Table 4.4.: Statistics of used benchmarks. Moral and poker have two different versions which are shown with two values in each column.

can refer to several role classes. For example, the following refinement chain is possible in case (iv):

$$\begin{aligned}\checkmark\quad & \exists\mathsf{hasIE.DiscManufacturingEquipment}\\ & \quad\rightsquigarrow \exists\mathsf{hasIE.(Robot} \sqcap \mathsf{AutomationMLBaseRole)}\\ & \quad\rightsquigarrow \exists\mathsf{hasIE.(Robot} \sqcap \mathsf{DiscManufacturingEquipment)}\\ & \quad\rightsquigarrow \exists\mathsf{hasIE.(Robot} \sqcap \mathsf{Machine)}\end{aligned}$$

In other cases, ρ is kept as it was in Fig. 3.3. The new refinement operator is called ρ^{aml} and is implemented based on ρ. Note that negated atomic concepts such as $\neg A$ are ignored in both M_{ie} and M_{ei} since negations are not preferred in engineering and are not used in practice. Compared with ρ and ρ^{cl}, ρ^{aml} would generate far less unnecessary refinements for learning with AML data.

4.4. Results and Discussions

This section demonstrates the effectiveness of RRHC and ρ^{aml} by conducting extensive comparisons with DL-Learner using both standard DL learning benchmarks and AML-specific learning tasks.

4.4.1. RRHC vs. CELOE

For the comparison between the learning algorithms RRHC and CELOE, the benchmarks provided by DL-Learner is used. Table 4.4 enumerates the ontologies of these benchmarks with the statistics[1] of their classes (#class), individuals (#ind), object properties (#op), data properties (#dp), and axioms (#axiom).

For each benchmark, there can be several learning problems, such as Mother and Uncle in the family benchmark. These learning problems are categorized by the learning time

[1]The statistics are obtained using the Protégé editor.

required to find the first correct solution: *simple* problems can be solved by both algorithms within 3 seconds, *medium* problems require more than 3 seconds by at least one algorithm, and *hard* problems can not be solved within 300 seconds (timeout) by at least one algorithm. The last column of Table 4.4 shows the difficulty of each benchmark. Note that some benchmarks have mixed difficulties, such as family. To assess the performance of the algorithms more fairly, different evaluation criteria are chosen for individual categories as follows:

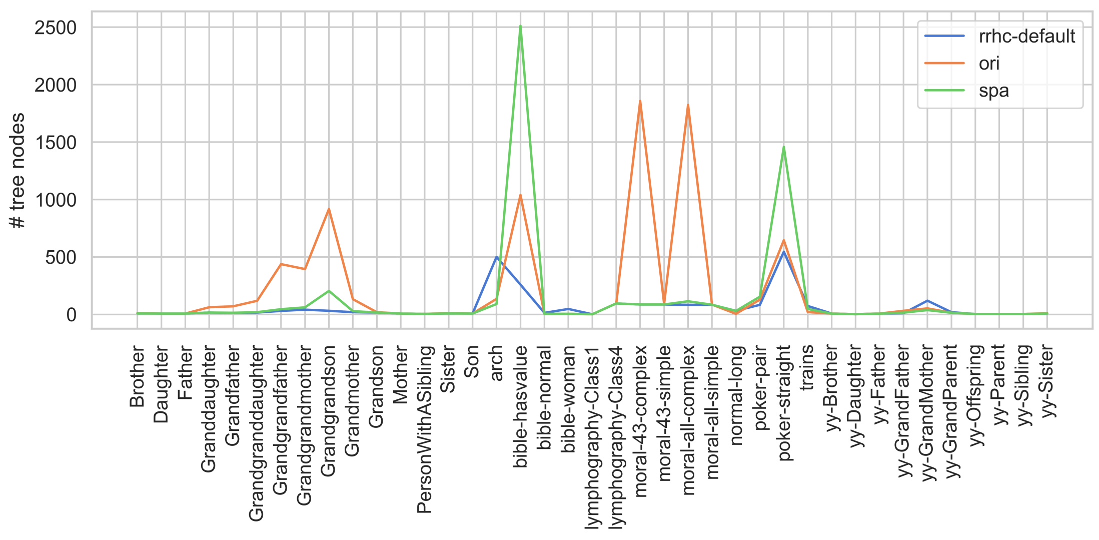
- *simple*: since both algorithms found the solution very fast, the size of the search tree is used to compare the learning efficiency in order to avoid the impact of oscillations of the computing power.
- *medium*: in this case, the time required for the first solution is used.
- *hard*: in all hard cases, the accuracy of the best concept found are compared between the algorithms.

One important intention of the experiments was to show that RRHC is easier to use than CELOE in terms of parameter tuning. Therefore, only the default configuration `rrhc-default` was chosen for RRHC (last row in Table 4.1) for all benchmarks. It is worth noting that `rrhc-default` is not always the best one for all learning problems, but it performed statistically well throughout all benchmarks. For CELOE, all configurations that are summarized in Table 4.1 are taken into account. Note that `ori_v3` has exactly the same setting as `rrhc-default`.

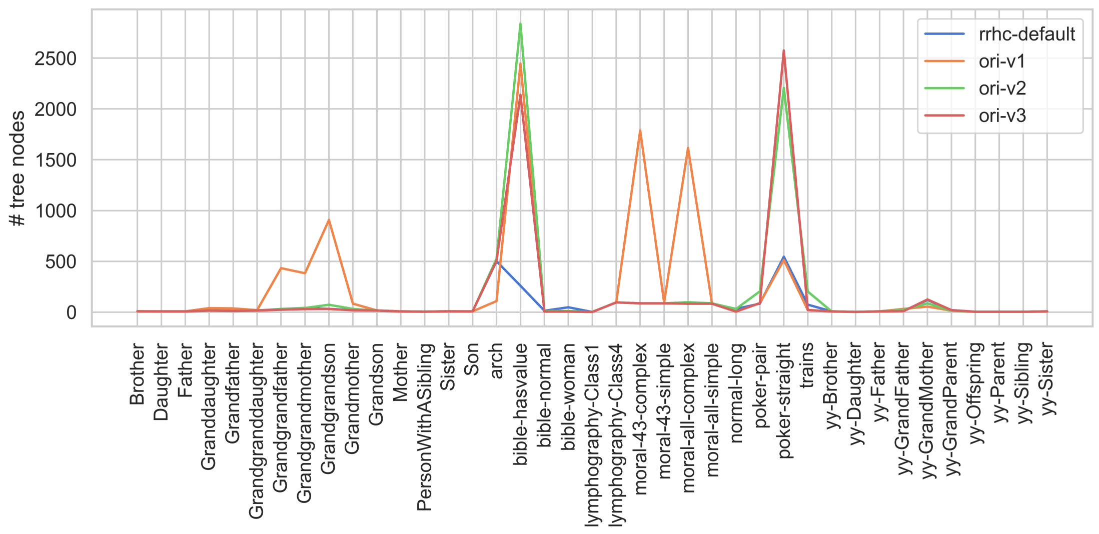
The experiments are run on a machine with a 2.6 GHz CPU and 16 GB memory. The timeout for all learning problems is 300 seconds. Moreover, *noise percentage* is disallowed in all experiments. Therefore only complete concept hypotheses are considered as potential solutions. For both algorithms, *predictive accuracy* is used to compute the accuracy of hypotheses. Fig. 4.9 and 4.10 show the comparisons of RRHC and CELOE using all positive and negative examples provided in the corresponding datasets. In all plots, the blue line or bar depicts the value of RRHC, while the others represent different CELOE configurations.

For the simple learning problems, Fig. 4.9 shows the number of tree nodes when the first solution is found. Besides some trivial ones in which both algorithms had very similar results, RRHC generally performed better than CELOE. One exception is the arch benchmark, which strongly prefers large expansion penalty, such as 0.1 from `ori` and `ori_v1` and 0.05 from `spa`. In contrast, the expansion penalty of both `ori_v3` and `rrhc-default` is set to 0.02, which lead to almost 500 tree nodes in Fig. 4.9b. To show the effects of different parameters, consider the learning problems in the family benchmark (the first 15 problems in the x-axis). On the one hand, `rrhc-default` worked better than both `ori` and `spa` in Fig. 4.9a. On the other hand, the difference between `rrhc-default` and `ori_v3` is minimal in Fig. 4.9b. This observation might indicate that `ori_v3` is also a better configuration for CELOE, which does not even utilize the start node bonus and the refinement penalty. However, for the learning problems bible-hasvalue and poker-straight, `ori_v3` was worse than other configurations, such as `ori` and `ori_v1`.

For the medium learning problems, Fig. 4.10a shows the time required (in seconds) for constructing the first solution, and RRHC was always the best one. One can also notice the huge difference between the various configurations of CELOE. For example, `spa` was extremely slow in the learning problem Uncle, but faster than `ori` and `ori_v2` in Cousin.

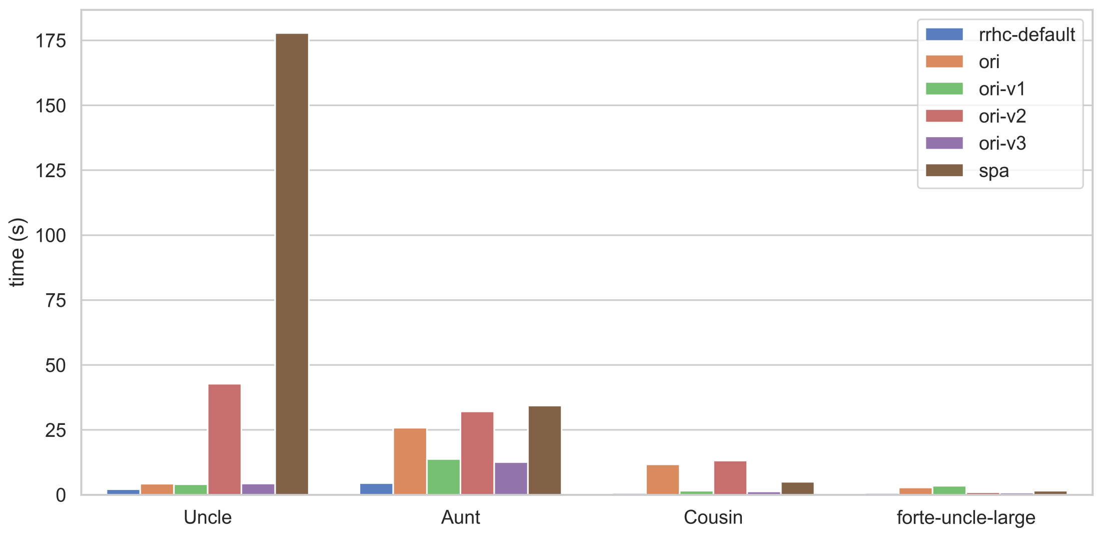


(a) Tree size of `rrhc-default, ori, spa` for simple learning problems.

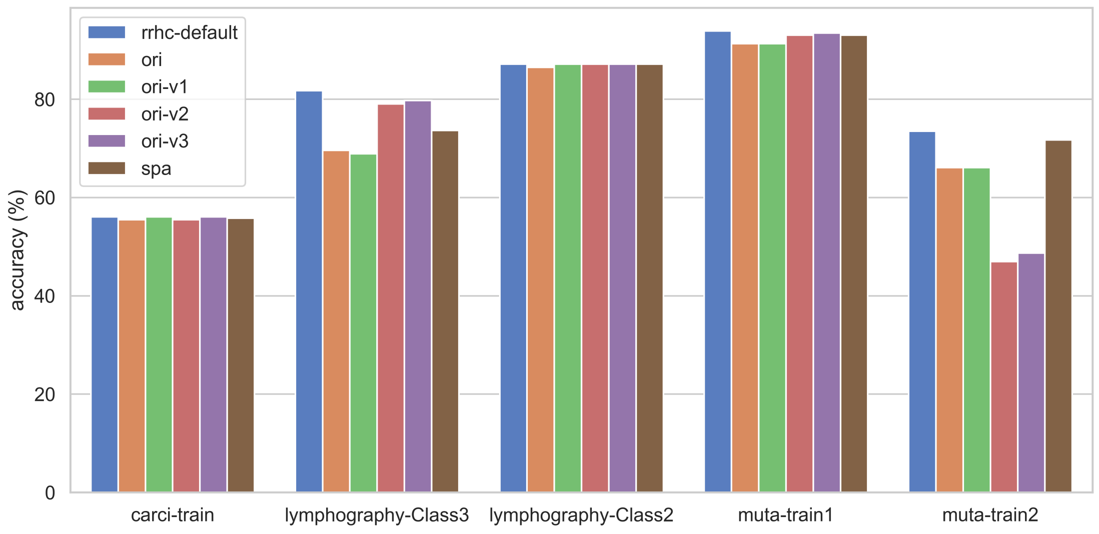


(b) Tree size of `rrhc-default, ori_v1, ori_v2, ori_v3` for simple learning problems.

Figure 4.9.: Comparison of RRHC and CELOE in simple learning problems.



(a) Time required of all configurations for medium learning problems.



(b) Best accuracy obtained from all configurations for hard learning problems.

Figure 4.10.: Comparison of RRHC and CELOE in medium and hard learning problems.

Learning Problem	CELOE-`ori`	CELOE-`ori_v3`	RRHC-`default`
Computing time of simple learning problems (sec)			
arch	**0.021** ± *0.008*	0.059 ± *0.03*	0.037 ± *0.02*
Grandfather	0.109 ± *0.032*	0.03 ± *0.004*	**0.027** ± *0.005*
Grandgrandson	0.66 ± *0.164*	0.061 ± *0.01*	**0.026** ± *0.005*
bible-hasValue	0.505 ± *0.205*	1.318 ± *0.98*	**0.335** ± *0.168*
moral-43-complex	0.742 ± *0.321*	**0.042** ± *0.004*	0.062 ± *0.009*
poker-pair	0.105 ± *0.031*	0.073 ± *0.019*	**0.059** ± *0.009*
poker-straight	**0.207** ± *0.036*	2.494 ± *1.727*	0.47 ± *0.228*
Computing time of medium learning problems (sec)			
Uncle	10.454 ± *9.237*	6.078 ± *1.743*	**4.697** ± *0.903*
Aunt	82.984 ± *83.206*	19.997 ± *7.942*	**7.877** ± *1.052*
Cousin	15.626 ± *1.779*	1.725 ± *0.538*	**1.4** ± *0.082*
forte-uncle-large	7.52 ± *2.357*	45.743 ± *83.427*	**3.141** ± *4.658*
Accuracy of hard learning problems (%)			
carcinogenesis	70.325 ± *2.32*	69.804 ± *2.384*	**70.581** ± *1.989*
lymphograhy-Class2	**86.61** ± *7.785*	84.998 ± *7.325*	85.063 ± *8.138*
lymphograhy-Class3	65.142 ± *4.484*	68.729 ± *10.039*	**72.12** ± *5.877*
mutagenesis-train1	91.92 ± *2.764*	**93.868** ± *3.319*	92.95 ± *3.085*
mutagenesis-train2	18.547 ± *11.077*	18.722 ± *14.166*	**19.359** ± *6.014*

Table 4.5.: Comparison of CELOE and RRHC in a 10-fold cross validation. For simple and medium learning problems, the time required for producing the first solution is used for comparison (the lesser the better). For hard learning problems, because no algorithm could find a solution within 300 seconds, the accuracy of the best solution found within 300 seconds is used for comparison (the greater the better). Bold values are the best among all three configurations. Italic values represent the standard deviation.

For the hard cases, Fig. 4.10b shows the accuracy of the best solution found within 300 seconds. Apparently, the performance of RRHC was either comparable to or better than the best CELOE configuration. One extremely hard benchmark is carcinogenesis, which is very noisy, so that both algorithms had poor results because the noise percentage is always set to zero. Among the CELOE configurations, it is evident that no single setting was significantly better than the others.

Because `ori` and `ori_v3` had comparatively good results when using CELOE, they are compared again with `rrhc-default` in a 10-fold cross-validation. Table. 4.5 shows the means and standard deviations of the measurements for some representative learning problems. In 11 out of 16 cases, `rrhc-default` performed the best, while in the remaining 5 cases, the result of RRHC was between the two CELOE configurations. In terms of stability, the standard deviation of RRHC is much smaller in 3 out of 4 medium learning problems and is comparable with CELOE in simple and hard learning problems.

4.4.2. ρ^{aml} vs. ρ^{cl}

This section compares the performance between the refinement operators ρ^{cl} and ρ^{aml} for learning engineering concepts from AML data. Similar to Section 4.4, only 100% accurate solutions are taken into account, and no noise percentage is allowed. The raw AML document comes from the research project ReApp, which was originally used for the modeling of industrial robot systems [21]. The lifted AML ontology comprises of 224 classes, 625 individuals, 75 data properties, and a total amount of 3794 axioms. The experiments are carried out for 12 synthetic learning problems with the following ground truths:

$$
\begin{aligned}
\mathsf{C1} &\equiv \exists \mathsf{hasIE}.(\forall \mathsf{hasIE}.\mathsf{ServoMotor})\\
\mathsf{C2} &\equiv \exists \mathsf{hasIE}.(\exists \mathsf{hasNumAxis}.(>= 5))\\
\mathsf{C3} &\equiv \exists \mathsf{hasIE}.(\exists \mathsf{hasIE}.\mathsf{EthernetPhysicalDevice})\\
\mathsf{C4} &\equiv \exists \mathsf{hasIE}.(\exists \mathsf{hasIE}.(\exists \mathsf{hasEI}.\mathsf{DigitalIOInterface}))\\
\mathsf{C5} &\equiv \exists \mathsf{hasIE}.(\forall \mathsf{hasIE}.\mathsf{ServoMotor} \sqcap \exists \mathsf{hasNumAxis}.(>= 5))\\
\mathsf{C6} &\equiv \exists \mathsf{hasIE}.(\exists \mathsf{hasIE}.\mathsf{ServoMotor}) \sqcap \exists \mathsf{hasIE}.(\exists \mathsf{hasIE}.\mathsf{EthernetPhysicalDevice})\\
\mathsf{C7} &\equiv \exists \mathsf{hasIE}.(\exists \mathsf{hasNumAxis}.(>= 5)) \sqcap \exists \mathsf{hasIE}.(\exists \mathsf{hasIE}.\mathsf{EthernetPhysicalDevice})\\
\mathsf{C8} &\equiv \exists \mathsf{hasIE}.(\exists \mathsf{hasNumAxis}.(>= 5)) \sqcap \exists \mathsf{hasIE}.\mathsf{RC}\\
\mathsf{C9} &\equiv \exists \mathsf{hasIE}.(\mathsf{Robot} \sqcap \exists \mathsf{hasEI}.\mathsf{Communication} \sqcap \exists \mathsf{has}_{\mathsf{n}}\mathsf{umAxes}.(\geq 5))\\
\mathsf{C10} &\equiv \forall \mathsf{hasIE}.(\mathsf{ArticulatedRobot} \sqcup (\mathsf{RC} \sqcap \exists \mathsf{hasIE}.\mathsf{EthernetPhysicalDevice}))\\
\mathsf{C11} &\equiv \exists \mathsf{hasIE}.((\exists \mathsf{hasIE}.\mathsf{ServoMotor}) \sqcap \exists \mathsf{hasNumAxis}.(>= 5))\\
&\qquad \sqcap \exists \mathsf{hasIE}.(\exists \mathsf{hasIE}.\mathsf{EthernetPhysicalDevice})\\
\mathsf{C12} &\equiv \exists \mathsf{hasIE}.(\mathsf{Robot} \sqcap \exists \mathsf{hasIE}.\mathsf{ServoMotor} \sqcap \exists \mathsf{hasNumAxis}.(>= 5))\\
&\qquad \sqcap \exists \mathsf{hasIE}.(\mathsf{RC} \sqcap \exists \mathsf{hasIE}.\mathsf{EthernetPhysicalDevice})
\end{aligned}
$$

For all learning problems, each refinement operator ρ^{cl} and ρ^{aml} is evaluated with both learning algorithms CELOE and RRHC. In other words, the following 4 test cases are taken into account: (i) (CELOE, ρ^{cl}) (ii) (CELOE, ρ^{aml}) (iii) (RRHC, ρ^{cl}) (iv) (RRHC, ρ^{aml}). The parameters of CELOE and RRHC may vary across different target concepts but remain the same in one learning problem. Because the aim of ρ^{aml} is to restrict the space of hypotheses by eliminating refinements that would violate the schema definition of AML, we are interested in the number of tested OWL class expressions until the first correct solution is found. Table 4.6 shows the results that are recorded for the individual test cases. ρ^{aml} outperformed ρ^{cl} in all learning problems (the value of ρ^{aml}/ρ^{cl} is always smaller than 100%), independent whether the learning algorithm is CELOE or RRHC. Moreover, in many hard problems, e.g., C6-C12, the ratio between ρ^{aml} and ρ^{cl} is smaller than 30%. The bold values also indicate that RRHC with ρ^{aml} performed better than CELOE in 8 out of 12 learning problems.

Learning Problem	CELOE			RRHC		
	ρ^{cl}	ρ^{aml}	ρ^{aml}/ρ^{cl}	ρ^{cl}	ρ^{aml}	ρ^{aml}/ρ^{cl}
C1	626	**384**	61%	694	555	79%
C2	3930	2504	63%	3004	**1657**	55%
C3	1780	**1538**	86%	8731	2297	26%
C4	1288181	862531	66%	521670	**139819**	26%
C5	35857	**31510**	87%	55610	46204	83%
C6	1703464	62133	3%	317266	**60015**	18%
C7	1151476	**63266**	5%	378537	102432	27%
C8	967173	26580	2%	595538	**9308**	1%
C9	1000555	57961	5%	400673	**48590**	12%
C10	961610	26504	2%	95009	**13591**	14%
C11	2351609	63725	2%	356543	**61410**	17%
C12	1768393	27564	1%	43108	**2363**	5%

Table 4.6.: Number of tested OWL class expressions until the first correct solution is found. Bold values are the best (lowest) within each row. ρ^{aml}/ρ^{cl} is the ratio between the values of ρ^{aml} and ρ^{cl}. Because $\rho^{aml}/\rho^{cl} < 100\%$ for all learning problems, ρ^{aml} is more suitable for learning from AML data. Furthermore, the combination (RRHC, ρ^{aml}) outperformed the combination (CELOE, ρ^{aml}) in 8 out of 12 learning problems.

5. Interactive Concept Learning in AutomationML

The concept learning work flow illustrated in Fig. 4.6 is designed based on the architecture of the DL-Learner framework [78], which can be unintuitive for domain experts because they are unfamiliar with OWL. Therefore, this chapter proposes the interactive machine learning (IML) framework AMLLEARNER that involves domain experts more tightly into the learning procedure. The core idea of AMLLEARNER is the bidirectional communication between the user and the learner based on the intermediate AML Concept Model (ACM). ACMs are native AML models that can represent OWL complex class expressions. That means the user can review the outputs of the learner and modify them by demand. AMLLEARNER is able to react upon changes over the previous results and restart with the modified ones as an initial guess. This chapter is organized as follows. Section 5.1 introduces the ACM, and Section 5.2 describes the bidirectional translation between ACMs and OWL class expressions. Then, Section 5.3 presents AMLLEARNER and shows the workflow of interactive learning. Finally, Section 5.4 discusses the interactive features of AMLLEARNER according to the literature of IML systems.

5.1. The AML Concept Model

Recall the translation between AML and OWL described in Section 3.1.4. It is evident that while AML role and interface classes can be mapped from/to OWL atomic classes, most OWL class constructors, as shown in Table 3.2, have no correspondence in AML. Because the outputs of the concept learner are complex OWL class expressions, which consist of several (nested) class constructors, they are not immediately representable as AML models.

Formula 5.1 shows some examples of OWL complex classes. Note that while an AML ontology only contains two object properties hasIE and hasEI, it is beneficial to take into account the following inverse properties for describing “part-of" relations:

$$\mathsf{isIEOf} \equiv \mathsf{hasIE}^-, \mathsf{isEIOf} \equiv \mathsf{hasEI}^-$$

Class A refers to Robots without any internal element of the type $\neg\mathsf{IOController}$. Class B denotes internal elements of a Robot from the manufacturer KUKA. Class C describes Robots with an IOController that has at least three IOInterfaces. Class D corresponds to IOInterfaces from objects that have at least three IOInterfaces. Class E stands for Robots with either an IOController or an IODevice. Apparently, as the complexity grows, the intended meaning of an OWL complex class becomes more difficult to understand.

$$\begin{aligned}
&\mathsf{A} \equiv \mathsf{Robot} \sqcap \neg\exists \mathsf{hasIE}.(\neg \mathsf{IOController})\\
&\mathsf{B} \equiv \exists \mathsf{isIEOf}.(\mathsf{Robot} \sqcap \mathsf{hasManufacturer}.\text{"KUKA"})\\
&\mathsf{C} \equiv \mathsf{Robot} \sqcap \exists \mathsf{hasIE}.(\mathsf{IOController} \sqcap \geq 3 \mathsf{hasEI}.\mathsf{IOInterface})\\
&\mathsf{D} \equiv \mathsf{IOInterface} \sqcap \exists \mathsf{isEIOf}.(\geq 3 \mathsf{hasEI}.\mathsf{IOInterface})\\
&\mathsf{E} \equiv \mathsf{Robot} \sqcap \exists \mathsf{hasIE}.(\mathsf{IOController} \sqcup \mathsf{IODevice})
\end{aligned} \tag{5.1}$$

For representing such OWL complex classes in AML, a modeling approach is required for mapping individual OWL class constructors to appropriate AML artifacts. This mapping, however, does not need to cover `DataExactCardinality`, `DataAllValuesFrom`, `DataMinCardinality`, `DataMaxCardinality`, and `ObjectHasSelf`. The first four are not relevant for AML since data properties are mapped from CAEX attributes, which are assigned to each distinct AML object only once. Similarly, `ObjectHasSelf` cannot appear in an AML ontology, because the object properties hasIE, hasEI, isIEOf and isEIOf can never be reflexive. For all other OWL class constructors, the mapping is defined as follows:

Atomic class: similar to [17], an atomic class is represented by a CAEX role or interface class. A class reference in CAEX is therefore equivalent to a class assertion in OWL. For example, an internal element a of the role class A is represented as $A(a)$.

Thing: Thing is the most general concept in OWL and contains all individuals. Depends on its CAEX annotation, Thing is represented by a corresponding CAEX schema element with no specific configurations.

Nothing: Nothing is the most specific concept in OWL and contains no individual. Nothing is handled as the complement of Thing (see the complement case below).

Intersection: an intersection $C \sqcap D$ contains individuals that are instances of all the operands C and D in the intersection. Therefore, an intersection is represented by the composition of several AML models that correspond to each of the operands, including CAEX class references, attributes, and subordinate object structures.

Union: a union $C \sqcup D$ contains individuals that are instances of at least one operand C or D of the union. Because XML does not support unions in general, each operand of a union generates one AML model individually.

Nominal: a nominal $\{a, b, ...\}$ enumerates all individuals that an OWL class shall contain. Similar to the union constructor, nominals cannot be directly represented in XML, and one AML model is generated for each element inside a nominal.

Existential restriction: an existential restriction $\exists R.C$ or $\exists R.(DR)$ states the existence of the relation R with the filler C or the data range DR. If R is an object property, the existential restriction is represented by a child object (internal element or external interface) while the filler C is represented by the model of the child object. If R is a data property, the existential restriction is represented by a CAEX attribute while the data range DR is represented by the configuration of the CAEX attribute, e.g. data type and value requirements.

Object cardinality restrictions: an object cardinality restriction, i.e. an exact restriction $= nR.C$, an at-least restriction $\geq nR.C$, or an at-most restriction $\leq nR.C$, defines the number of child objects of the class C w.r.t. the relation R. The CAEX attributes *minCardinality* and *maxCardinality* are added to the child objects to represent the minimum and maximum number respectively. The exact cardinality of n is represented by $minCardinality = maxCardinality = n$.

Fills restriction: a fills restriction $\exists R.\{a\}$ or $\exists R.\{lt\}$ corresponds to an existential restriction with a Singleton filler, i.e. a single object a or a literal value lt. If R is an object property, the CAEX attribute *identifiedByID* is used to restrict the ID of the child object a, as ID is unique in AML. If R is a data property, lt is set as the required value of the corresponding CAEX attribute.

Universal restriction: a universal restriction $\forall R.C$ forces all child objects w.r.t. the relation R to be instances of the class C. For example, $\forall \mathsf{hasIE.C}$ describes things that have internal elements of type C only. While universal restrictions can not be directly represented in XML, it can be simulated by disallowing child objects that are instances of the class $\neg C$ [79] using the exact cardinality $= 0\, R.(\neg C)$.

Complement: a complement $\neg C$ contains all individuals that are not instances of C. Since an OWL class can have arbitrarily nested complements, its negation normal form (NNF) is first computed where the complements are only bound to atomic classes [54]. For example, the NNF of the OWL class A in Formula 5.1 is:

$$\mathsf{NNF(A)} \equiv \mathsf{Robot} \sqcap \forall \mathsf{hasIE.IOController}$$

Obviously, $\mathsf{NNF(A)}$ does not contain any complements. In fact, complements can only appear in the following three cases in the NNF of an OWL class:

(a) A complement can be bound to an atomic class as $\neg A$ or a data range as $\neg DR$, and is not part of any restrictions. In this case, a CAEX attribute *negated=true* is added to the AML model. Note that intersections of a mixture of positive and negative atomic classes, e.g. $\neg A_1 \sqcap A_2$, cannot be modeled in AML.

(b) A complement can be the filler of an existential restriction, i.e. $\exists R.(\neg A)$ or $\exists R.(\neg DR)$. As with the existential restriction, a child CAEX object or CAEX attribute is first generated. Then the CAEX attribute *negated=true* is added to the child model.

(c) A complement can be the filler of a universal restriction as $\forall R.(\neg A)$ (recall that universal restrictions on data properties are ignored). In this case, child objects of the class A are disallowed w.r.t. the relation R, which can be expressed using the exact cardinality $= 0\, R.A$.

Table 5.1 summarizes the introduced CAEX attributes that are used to capture the semantics of OWL constructors mentioned above. The attribute *primary* is a helper flag to indicate which element in an AML model is described by the OWL class. These CAEX attributes are called *concept attributes*, and an AML model with concept attributes is called an *AML Concept Model* (ACM). Note that ACMs are merely an illustration of the corresponding OWL complex class. In other words, they are not part of an AML ontology and do not influence the semantics and the inference of an AML ontology. For example, the ACM of an existential restriction does not refer to any concrete individual and does not contradict with the open-world assumption of DL.

Table 5.2 enumerates the values of concept attributes based on possible forms of NNF. Note that intersections, unions, and nominals are omitted in the mapping since each

Table 5.1.: The AML concept attributes for capturing ontological semantics.

Name	Type	Default	Semantics in OWL
negated	bool	false	complement
minCardinality	integer	1	minCardinality
maxCardinality	integer	unlimited	maxCardinality
identifiedByID	bool	false	nominal
primary	bool	false	target individuals

Table 5.2.: Mapping between OWL constructors and AML concept attributes.

OWL Class Expression	Negated	minCard.	maxCard.
simple complement $\neg C$ or $\neg DR$	true	1	unlimited
existential restriction $\exists R.C$ or $\exists R.DR$	false	1	unlimited
existential restriction $\exists R.(\neg C)$ or $\exists R.(\neg DR)$	true	1	unlimited
universal restriction $\forall R.C$	true	0	0
universal restriction $\forall R.\neg C$	false	0	0
at-least restriction $\geq nR.C$	false	n	unlimited
at-most restriction $\leq nR.C$	false	0	n

operand (element) of them is handled individually. Intuitively, ACMs can be nested to represent nested OWL class expressions. An ACM is *proper* if it has exactly one primary element.

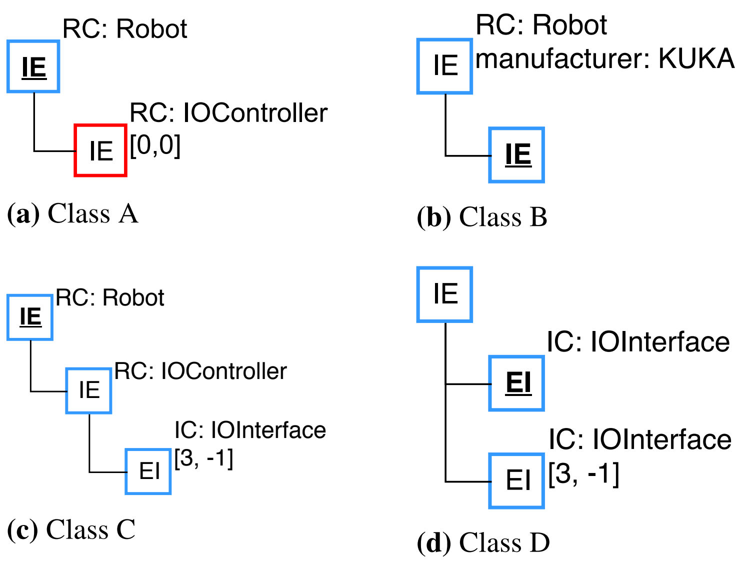
Fig. 5.1 illustrates the ACMs of the NNF of the OWL classes A, B, C and D in Formula 5.1 as tree structures (we handle the OWL class E later in Section 5.2.1). Internal elements (IE) and external interfaces (EI) are represented by tree nodes, and their class references and attributes are depicted as labels on the top right corner. A negated object is marked as red. The primary object is marked as bold with an underline. Numbers in square brackets are the min and max cardinality of the object, while a value -1 means that it is unlimited. Note that for the classes B and D, the primary object is not the root node since XML cannot describe "part-of" relations (i.e. $\mathsf{isIEOf}, \mathsf{isEIOf}$). Therefore, each inverse property is simulated by a corresponding predecessor node in the XML tree.

5.2. Translation between OWL and ACM

The core idea of the translation is to exploit the **tree structure** of OWL class expressions. More concretely, the so-called *AML concept trees* are used to depict OWL complex classes in a tree structure similar to ACMs. The *forward* translation from OWL to ACM, i.e., $\mathrm{TransF} : \mathrm{OWL} \mapsto \mathrm{ACM}$, is defined based on AML concept trees. The opposite direction, i.e., the *backward* translation $\mathrm{TransB} : \mathrm{ACM} \mapsto \mathrm{OWL}$ can be directly carried out using the mappings in Table 5.2.

5.2.1. From OWL to AND-tree

A tree is defined conventionally as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a finite set of nodes and $\mathcal{E}$ is a finite set of edges, to which the following rules apply:



(a) Class A **(b)** Class B **(c)** Class C **(d)** Class D

Figure 5.1.: The ACMs for the OWL classes in Formula 5.1.

- A tree $\mathcal{G}$ has a unique *root* node that has no predecessor.
- Each node $n \in \mathcal{V}$ has a unique predecessor.

In a tree $\mathcal{G}$, *leaf nodes* are the tree nodes with no successor, i.e. at the bottom of the tree. Furthermore, a *branching node* is an inner tree node that has a unique predecessor and arbitrarily many successors. Based on these notions, an *AND-tree* is a tree with the following properties:

- The root of an AND-tree represents the expression of an OWL complex class.
- Each branching node of an AND-tree represents either an intersection or a restriction in OWL.
- Each leaf node of an AND-tree represents either OWL Thing, OWL Nothing or an atomic class.

For each OWL complex class without unions and inverse properties, an AND-tree can be constructed by making a successor node for each operand of an intersection and the filler of a restriction, as shown in Algorithm 5.1. Fig. 5.2 illustrates the construction of the AND-tree for the OWL class D in Formula 5.1. Each box represents a tree node, and the number on the upper left corner of each box shows the sequence of node construction. The root node of the AND-tree corresponds to the expression of class D. Since the root is an intersection, the algorithm will handle each operand of it individually through line 4 to 6. The atomic operand $\mathsf{IOInterface}$ is returned directly and added as a child to the root in line 7. For the complex operand $\exists\mathsf{isIEOf}.(\geq 3\mathsf{hasEI}.\mathsf{IOInterface})$, the algorithm recursively generates sub-nodes until the final atomic filler $\mathsf{IOInterface}$ is reached in line 10. Note that all nodes are generated immediately in line 1 when $\mathrm{ConstructAndTree}$ is called.

It becomes more involved if the input OWL class contains any disjunctions (unions or nominals) because XML does not support *or* statements generally. The solution is to construct m AND-trees for a disjunction with m elements. However, since disjunctions can appear in any nested part inside an OWL class expression, a traverse of the logical

Algorithm 5.1 ConstructAndTree

```
Input: The class expression $ce$ of an OWL class $C$
Output: A tree node $root$
 1: make a tree node $root$ for $ce$
 2: if $ce$ is an atomic class then
 3:     return $root$
 4: else if ($ce$ is an intersection) then
 5:     for $operand \in ce$ do
 6:         let $child = \mathrm{ConstructAndTree}(operand)$
 7:         add $child$ as a successor to $root$
 8:     end for
 9: else if ($ce$ is a restriction) then
10:     let $child = \mathrm{ConstructAndTree}(ce.filler)$
11:     add $child$ as a successor to $root$
12: end if
13: return $root$
```

Algorithm 5.2 ConstructAndTreeD

```
Input: The class expression $ce$ of an OWL class $C$
Output: A set of tree nodes $roots$
 1: initialize $roots = \{\}$
 2: if $ce$ is an union or a nominal then
 3:     for each $element$ in $ce$ do
 4:         add $\mathrm{ConstructAndTreeD}(element)$ to $roots$
 5:     end for
 6: else
 7:     make a tree node $n$ for $ce$, add $n$ to $roots$
 8:     if $ce$ is an atomic class then
 9:         return $roots$
10:     else if ($ce$ is an intersection) then
11:         for $operand \in ce$ do
12:             let $nestedTrees = \mathrm{ConstructAndTreeD}(operand)$
13:             for $root$ in $roots$ do
14:                 copy $root$ $nestedTrees.size - 1$ times
15:                 add the root of each $tree \in nestedTrees$ as a successor to exactly one copy
                    of $root$
16:             end for
17:         end for
18:     else if ($ce$ is a restriction) then
19:         let $nestedTrees = \mathrm{ConstructAndTreeD}(ce.filler)$
20:         for $root$ in $roots$ do
21:             copy $root$ $nestedTrees.size - 1$ times
22:             add the root of each $tree \in nestedTrees$ as a successor to exactly one copy
                of $root$
23:         end for
24:     end if
25: end if
26: return $roots$
```

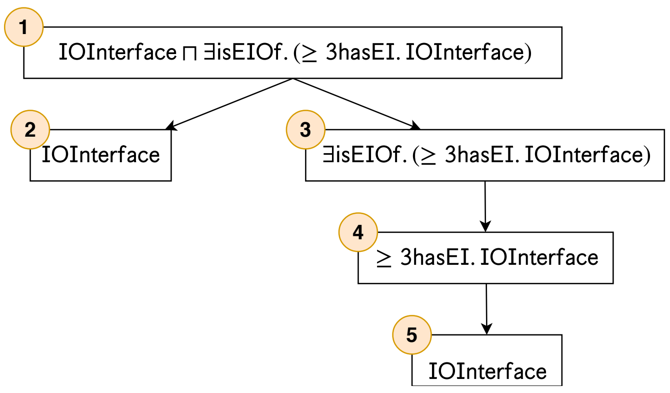

1
IOInterface ⊓ ∃isEIOf. (≥ 3hasEI. IOInterface)
2
IOInterface
3
∃isEIOf. (≥ 3hasEI. IOInterface)
4
≥ 3hasEI. IOInterface
5
IOInterface

Figure 5.2.: The AND-tree constructed from the OWL class D in Formula 5.1. The numbers in the tree nodes show the sequence of node construction.

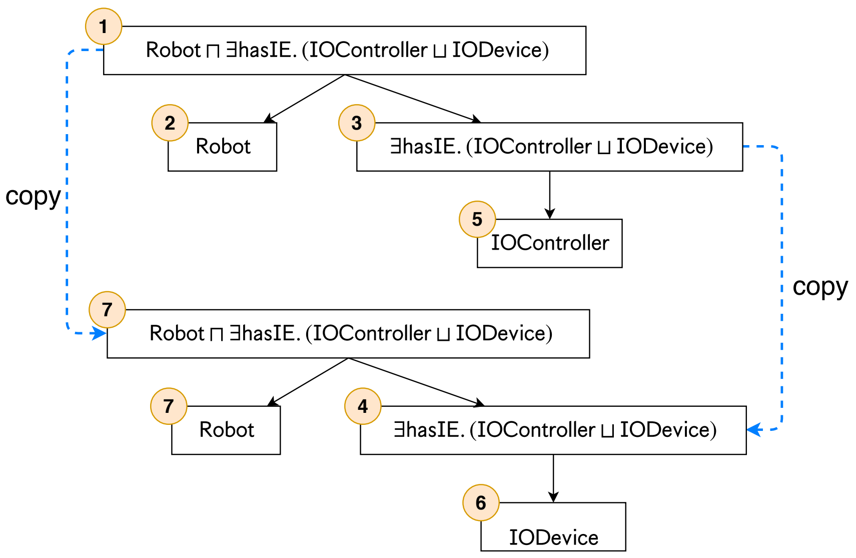

1
Robot ⊓ ∃hasIE. (IOController ⊔ IODevice)
2
Robot
3
∃hasIE. (IOController ⊔ IODevice)
5
IOController
copy
copy
7
Robot ⊓ ∃hasIE. (IOController ⊔ IODevice)
7
Robot
4
∃hasIE. (IOController ⊔ IODevice)
6
IODevice

Figure 5.3.: The tree construction process of the OWL class E in Formula 5.1. The numbers in the tree nodes show the sequence of node construction.

structure of the class expression is required to produce a set of AND-trees that is logically equivalent to the OWL class.

Algorithm 5.2 shows the AND-tree construction process for classes involving disjunctions. If the input class expression ce is a disjunction, then a set of tree nodes are generated for the elements of the disjunction (line 4). In case the input is an intersection, the recursive call of $\mathrm{ConstructAndTreeD}$ in line 12 will handle possible nested disjunction in each element and produce a set of nested trees. These nested trees need to be multiplexed with the existing trees in $roots$ through line 13 to 15. The algorithm treats restrictions similarly to intersections despite that the filler of a restriction is used to produce nested trees in line 19. It is worth noting that only $m - 1$ copies of $root$ are made in line 14 and 21 since the original $root$ also counts during the construction.

Fig. 5.3 illustrates the tree construction process of the OWL class E in Formula 5.1. In the first step, a root node is generated that contains the complete class expression (line 7). Then, for each operand of the intersection, a child node is generated in step 2 and 3 (line 12). Since the Robot node is atomic, no further construction is required in the recursive call (line 9). On the other hand, the restriction node ∃hasIE.(IOController ⊔ IODevice) is copied in step 4 (line 21), since its filler is a union and produces two atomic nodes IOController and IODevice (line 19). In step 5 and 6, the atomic nodes are added to the original and copied restriction nodes (line 22). Finally, the root node is copied once to accept the two distinct restriction nodes in step 7 (line 14-15).

5.2.2. Working with Inverse Properties

For OWL classes that describe objects in the instance hierarchy, inverse properties might appear for gathering information about their ancestors or siblings (see the OWL classes B and D in Formula 5.1). Due to structural restrictions in AML, following conditions apply when an inverse property $R^{-} \in \{\mathsf{isIEOf}, \mathsf{isEIOf}\}$ appear:

C1: R^{-} does not appear in the filler of any restriction that has R as property, e.g. $\exists R.(\exists R^{-}.C)$

C2: R^{-} does not appear in the filler of cardinality restrictions, e.g. $\geq n\ R^{-}.C$.

C3: R^{-} does not appear in the filler of any restriction that has a different property $R' \neq R$, e.g. $\exists R'.(\exists R^{-}.C)$.

C4: isEIOf does not appear in the filler of any restrictions that has an inverse property, e.g. $\exists R^{-}.(\exists \mathsf{isEIOf.C})$

The conditions C1 and C2 avoid modeling redundancies in OWL, since AML data has a tree structure, and each node in the tree has a unique predecessor. A class expression $\exists R.\exists R^{-}.C$ is therefore logically equivalent to C, and a cardinality restriction is redundant to an existential restriction. The condition C3 avoids modeling errors in OWL since the set of internal elements is disjoint with the set of external interfaces. The condition C4 holds since external interfaces have no child object in AML. An OWL class that meets the conditions C1-C4 is called a *proper AML class*.

The inverse properties of a proper AML class always appear continuously at the outermost layer of the class expression. In other words, the AND-tree of a proper AML class has all inverse properties in the upper part of the tree. Therefore, Algorithm 5.3 iteratively removes the inverse properties from the root of an AND-tree. We call an AND-tree that contains no disjunctions nor inverse properties an *AML concept tree*.

Fig. 5.4 shows how the inverse property in the root of class D's AND-tree (i.e., isEIOf in Fig. 5.2) is removed. Since the original root node is an intersection, the algorithm first constructs a template node for the new root (line 11). Then a new child node is constructed for the previous child IOInterface by formulating an existential restriction in step 2 (line 14 to 15). To keep the consistency of the tree, the expression of the new child node is added to the new root node in the third step (line 16). For the previous child $\exists\mathsf{isEIOf}.(\geq 3\mathsf{hasEI.IOInterface})$, the filler of its inverse property isEIOf, i.e., the expression $\geq 3\mathsf{hasEI.IOInterface}$ is added to the new root node as a conjunctive term in step 4 (line 18), and the corresponding grandchild with its sub-tree is added as a child to the new root in step 5 (line 19).

It is obvious that the inverse property isEIOf is now removed. Note that the OWL class expression of the new root node is different from the original one. Informally, the original root describes the *primary* object in an arbitrary position of a CAEX instance hierarchy (marked as yellow), while the new root describes the predecessor of the primary object.

Algorithm 5.3 removeInverseProperty

```
Input: The root of an AND-tree $root$
Output: The root of a new AND-tree $newRoot$
 1: let $ce$ = class expression in $root$
 2: if $ce$ contains no (nested) inverse property then
 3:     return $root$
 4: else
 5:     if $ce$ is a restriction then
 6:         construct a new node $newRoot$ for $ce.filler$
 7:         change the filler of $root$ to owl:Thing
 8:         add $root$ as a successor of $newRoot$
 9:         move $root.children$ as successors of $newRoot$
10:     else if $ce$ is an intersection then
11:         construct a template node $newRoot$
12:         let $inv$ be successors of $root$ with inverse property
13:         let $normal$ be other successors of $root$
14:         construct a new node $normalChild$ as an existential restriction with $normal$
            being its filler
15:         add $normalChild$ as a successor to $newRoot$
16:         add the expression of $normalChild$ to $newRoot$
17:         for $node \in inv$ do
18:             add the filler of $node$ to $newRoot$ conjunctively
19:             move $node.child$ as a successor of $newRoot$
20:         end for
21:     end if
22:     removeInverseProperty($newRoot$)
23: end if
```

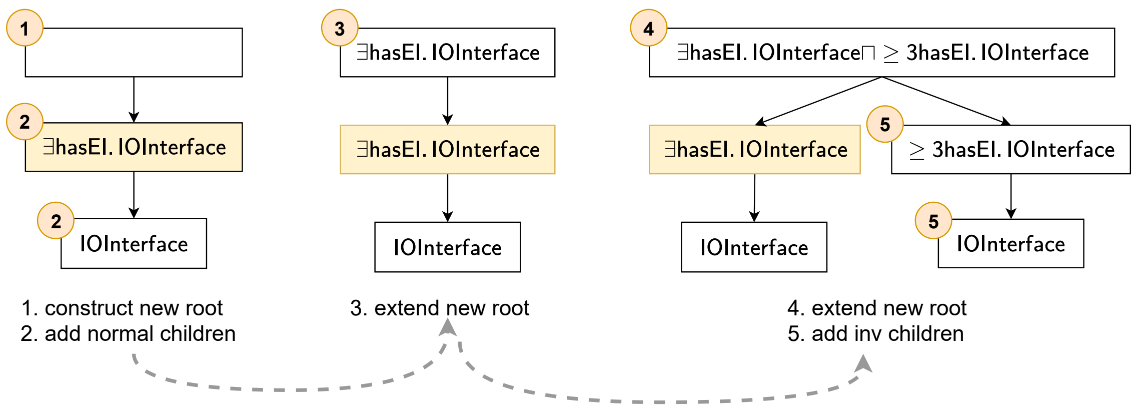


Figure 5.4.: The construction of the AML concept tree of class D. The numbers in the tree nodes show the sequence of node construction.

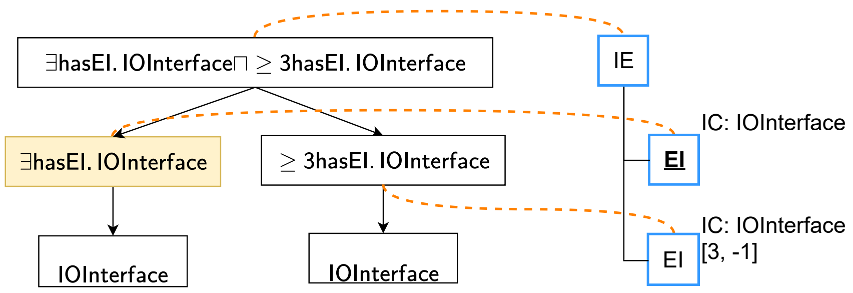


Figure 5.5.: The forward translation for class D.

5.2.3. The Forward Translation: from OWL to ACM

Until now, the algorithms to transform a proper AML class into an AML concept tree have been presented. The *forward* translation $\mathrm{TransF} : \mathrm{OWL} \mapsto \mathrm{AML}$ can be implemented by traversing AML concept trees in a depth-first manner. For every tree node, a corresponding ACM is generated whose concept attributes are configured based on the mappings in Table 5.2. The CAEX type of the target ACM is determined either by the object property being used in case of a restriction or by the CAEX type annotation of the OWL class in case of an intersection in the root node. The orange dashed lines in Fig. 5.5 show the translation from the AML concept tree of class D to its ACM illustrated in Fig. 5.1d. Note that OWL atomic classes are mapped to CAEX class references.

5.2.4. The Backward Translation: from ACM to OWL

If an ACM is proper, i.e. it has exactly one primary element (see section 5.1), then the *backward* translation $\mathrm{TransB} : \mathrm{AML} \mapsto \mathrm{OWL}$ can be directly carried out using the mappings in Table 5.2. First, a traverse of the ACM is necessary to localize the primary object. Afterwards, successors of the primary object are translated to restrictions with normal properties while the predecessors are translated to restrictions with inverse properties. If Algorithm 5.2 would have generated several ACMs during the forward translation, they are translated independently to several OWL classes and combined disjunctively as a union. In this case, an original OWL class with nested unions will be reproduced as a union of expressions, e.g. $\exists \mathsf{r}.(\mathsf{C} \sqcup \mathsf{D}) \rightarrow \exists \mathsf{r}.\mathsf{C} \sqcup \exists \mathsf{r}.\mathsf{D}$. Although the syntax of the reproduced OWL class differs from the original one, their semantics are the same.

It is worth noting that the mappings in Table 5.2 are used for both TransF and TransB. Therefore, the forward and backward translations are inverse functions of each other in terms of semantic equivalence. That means, for an OWL class C and an ACM M, the following relations hold:

$$\begin{aligned} \mathrm{TransB}(\mathrm{TransF}(C)) &\equiv C \\ \mathrm{TransF}(\mathrm{TransB}(M)) &\equiv M \end{aligned} \tag{5.2}$$

5.2.5. ACM in ontology engineering

To show the utility of ACM, consider ontology engineering for cyber-physical systems [26] where a so-called lightweight ontology is already modeled in AML.

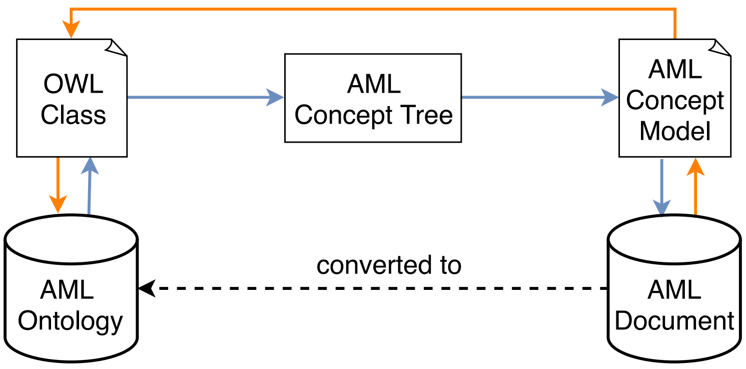


Figure 5.6.: The work flow for ontology engineering using bidirectional translation.

Fig. 5.6 shows two typical scenarios where OWL complex classes need to be either constructed or adapted.

The first scenario (orange arrows in Fig. 5.6) refers to the standard ontology building, in which a new OWL complex class needs to be constructed. Instead of manually creating the OWL class in the ontology, the domain expert (the user) constructs an ACM and translates it to OWL as follows:

1. The user generates the primary ACM for the target concept, i.e. a CAEX role class, system unit class, interface class, internal element or external interface with class reference and concept attributes.
2. The user adds CAEX attributes and sub-elements with sufficient constraints to the model. This process repeats recursively for nested attributes and sub-elements.
3. If the primary ACM shall be further restricted by the properties of its predecessor or siblings, a parent ACM is generated. This process repeats recursively for further predecessors and siblings.
4. The user generates the OWL class using the backward translation and adds it to the AML ontology.

The second scenario (blue arrows in Fig. 5.6) refers to ontology evolution where an existing OWL complex class needs to be adapted according to new requirements. In this case, the user might want to inspect and modify it by demand. First, the target OWL class is translated into ACMs via its AML concept trees. Then, the user can inspect the generated ACMs by browsing their XML structure. If any modification is necessary, the user can edit the ACMs as described above and export the new one to an OWL class.

In both scenarios, ACMs can be inspected and modified using a conventional AML editor, while the forward and backward translations are transparent to the user. By comparing the OWL complex classes in Formula 5.1 and their corresponding ACMs in Fig. 5.1, it is evident that ACMs are more intuitive for domain experts. Moreover, because the forward and backward translations are inverse functions of each other (see Formula 5.2 in Section 5.2.4), a round-trip engineering of OWL complex classes is also enabled by following the workflow of both scenarios successively. For various knowledge-driven industrial applications using AML, e.g., plant model validation [80], model-driven software engineering [21], and engineering of cyber-physical systems [81], ACMs can facilitate the modeling of sophisticated domain knowledge in OWL.

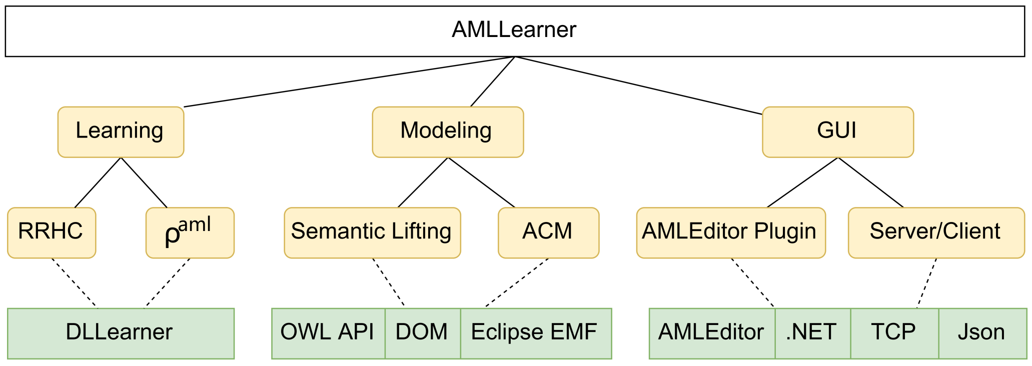


Figure 5.7.: The software architecture of the AMLLEARNER framework. Yellow boxes are modules/components implemented in AMLLEARNER while the green boxes are the used software libraries.

5.3. The AMLLEARNER Framework

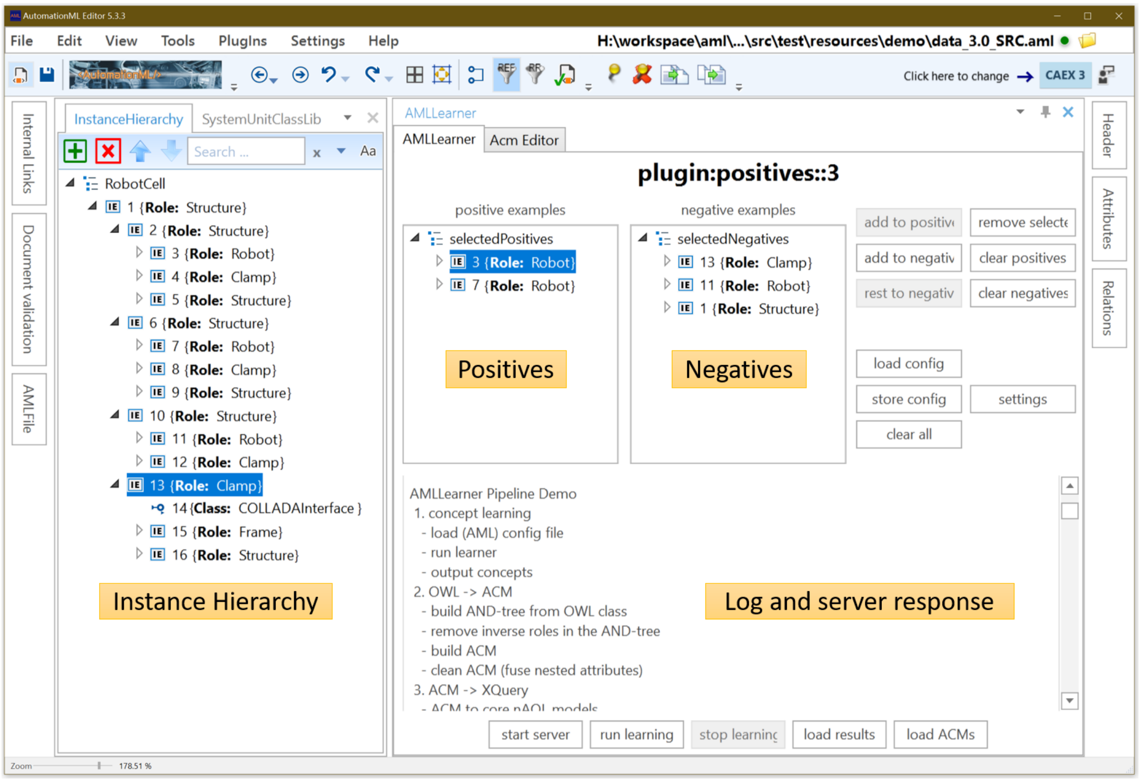
The advantage of the ontology building scenarios, as described above, is that no ontology expert is required. However, the domain expert still needs to create the ACMs for the desired engineering concepts manually. To further facilitate ontology building, this section presents the AMLLEARNER framework that allows the semi-automated creation of OWL complex classes by following the interactive machine learning (IML) approach [82]. Because formal semantics builds the foundation of concept learning algorithms, the behavior and results of the learning algorithms are inherently self-explainable. In fact, the main gap between the learner and the user is the non-intuitive syntax and semantics of OWL class expressions. Therefore, AMLLEARNER uses the concept learning algorithm presented in Chapter 4 as a subroutine and interacts with the user with the ACMs of the learned OWL classes.

Fig. 5.7 shows the software architecture of AMLLEARNER, which consists of the following three main modules:

- The *Learning* module is an extension of the DLLearner framework and contains the implementation of the RRHC algorithm and the refinement operator ρ^{aml}.
- The *Modeling* module comprises all components related to reading, writing, and transforming different data formats. Based on the OWL API[1], the XML DOM, and the Eclipse Modeling framework (EMF[2]), the modeling module supports the semantic lifting of AML and the forward and backward translations of ACMs.
- The *GUI* module realizes a server-client architecture in which the functionalities of AMLLEARNER are wrapped into a Java backend while a plugin of the AML Editor is provided as a C# client. The communication between the server and the client is based on JSON messages passed over a TCP channel.

The main window of the AMLLEARNER plugin is shown in Fig. 5.8. The left column is the viewer of AML instance hierarchies provided by the AML Editor. The right column is the GUI of the AMLLEARNER plugin, which contains two panels for the positive and negative examples, a text area for showing log and server responses, and two sets

[1]https://github.com/owlcs/owlapi
[2]https://www.eclipse.org/modeling/emf/



AMLLearner Pipeline Demo
1. concept learning
- load (AML) config file
- run learner
- output concepts
2. OWL -> ACM
- build AND-tree from OWL class
- remove inverse roles in the AND-tree
- build ACM
- clean ACM (fuse nested attributes)
3. ACM -> XQuery
- ACM to core nAQL models

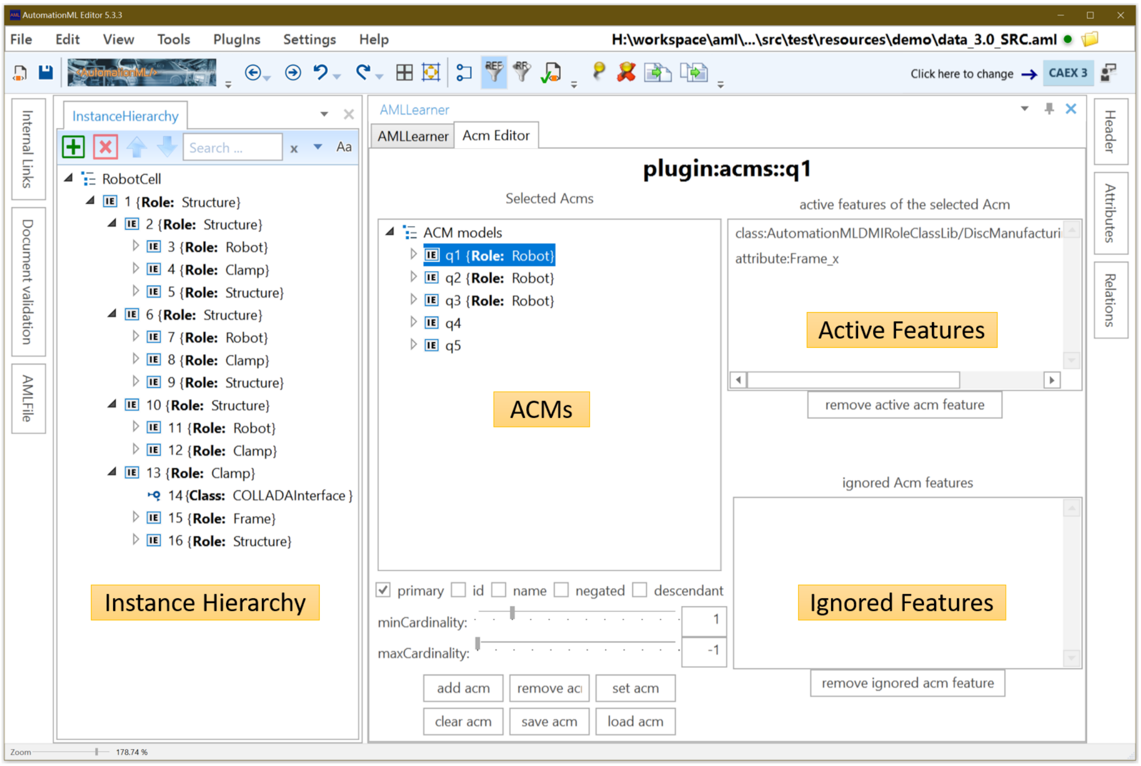
Figure 5.8.: The main window of the AMLLEARNER plugin in the AML editor.

of control buttons. The first set of control buttons (top right part) is used for editing the positive and negative examples and the configuration of learning algorithms. In particular, the *store* and *load config* buttons are used for storing the current configuration to a JSON file or loading an existing configuration to the plugin. The second set of control buttons (bottom part) is used for the interaction with the AMLLEARNER server. The user can stop the learning process at any time using the *stop learning* button and load the current outputs as ACMs using the *load ACMs* button.

Fig. 5.9 shows the ACM editing window of the AMLLEARNER plugin. Again, the left column is the view of instance hierarchies in the AML editor, and the right column contains the UI elements for showing and editing ACMs. For explaining the ACMs, the checkboxes and slides underneath the ACM window show the values of the concept attributes of the currently selected object. Moreover, the user can review the features used by the ACM in the *active features* panel. If any feature is not satisfied by the user, it can be removed by clicking the button *remove active acm feature*. The removed features of the currently selected ACM will be shown in the *ignored features* panel. The control buttons below the slides are designed for loading, removing, and storing ACMs. In particular, if the currently selected ACM shall be used as the initial guess for the next learning cycle, the user can click the button *set acm* and restart the learning in the main window of AMLLEARNER.

5.4. Results and Discussions

Recall the workflow for concept learning in AML, as described in Section 4.3. This standard workflow allows basic interactions between the user and the learner as follows:



```
RobotCell
  1 {Role: Structure}
    2 {Role: Structure}
      3 {Role: Robot}
      4 {Role: Clamp}
      5 {Role: Structure}
    6 {Role: Structure}
      7 {Role: Robot}
      8 {Role: Clamp}
      9 {Role: Structure}
    10 {Role: Structure}
      11 {Role: Robot}
      12 {Role: Clamp}
    13 {Role: Clamp}
      14 {Class: COLLADAInterface }
      15 {Role: Frame}
      16 {Role: Structure}
```

plugin:acms::q1

```
ACM models
  q1 {Role: Robot}
  q2 {Role: Robot}
  q3 {Role: Robot}
  q4
  q5
```

```
class:AutomationMLDMIRoleClassLib/DiscManufacturi
attribute:Frame_x
```

[✓] primary [ ] id [ ] name [ ] negated [ ] descendant

minCardinality: 1

maxCardinality: -1

Figure 5.9.: The ACM editing window of the AMLLEARNER plugin in the AML editor.

(F1): at the beginning, the user selects data examples.

(F2): at the beginning, the user can configure the parameters of the learning algorithm to control its behavior. Two major types of parameters are supported: a) parameters of the heuristic function that controls exploration vs. exploitation during search space traversal; b) language features of OWL that shall or shall not be used for constructing the solution, which can be certain class constructors or particular OWL classes and properties in the ontology. Note that the AMLLEARNER framework has a default setting for these parameters, and this type of interaction is an advanced feature for experts of OWL.

(F3): the learner communicates the learning result to the user when a solution is found, or it timed out. The result contains the accuracy and heuristic score of the best n solutions. The user can also ask for the coverage of data examples of each solution.

Fig. 5.10 shows the workflow of AMLLEARNER. The blue dashed lines illustrate the information flow from AMLLEARNER to the user using the forward translation (i.e., from OWL to ACM), while the orange dashed lines depict feedback from the user to AMLLEARNER using the backward translation (i.e., from ACM to OWL). Based on this paradigm, AMLLEARNER additionally provides the following features for interaction:

(F4): At any time point, the user can stop the learning and ask the system to show the current status.

(F5): If the response of the system would be rather complicated for human understanding, e.g., the class expressions in Formula 5.1, the user can request a translation of the concept into an ACM and inspect it in a conventional AML editor. If the solution is not satisfactory, the user can modify the ACM by demand and reinitialize AMLLEARNER with the modified ACM. Note that the modified ACM is firstly checked against semantic consistency concerning the user-supplied data examples. That means,

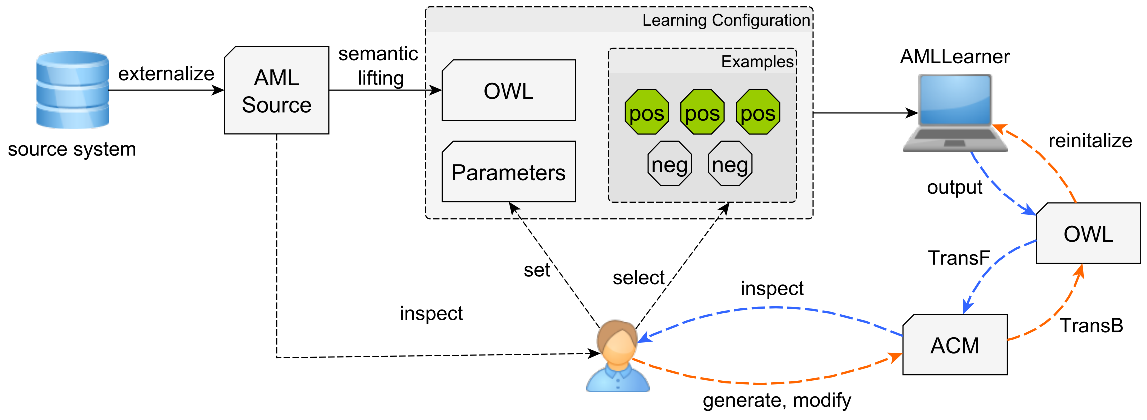


Figure 5.10.: The workflow for interactive concept learning in AML.

any user modified ACM that does not cover all positive examples are invalid for the learning and will trigger a warning. In this case, the user can either reconfigure the learner with different data examples or parameters, or withdraw the modifications of the ACM.

(F6): Often, the user has some ideas about the target concept. For example, not all robots are of interest, but only the ones produced by the manufacturer KUKA. Then the user would initialize AMLLEARNER with an ACM that contains such information. Again, the user will be notified if the supplied model is inconsistent with previously selected data examples.

(F7): When the learning terminates and the user is satisfied with the solution, the constructed OWL class expression can be saved both as part of the existing knowledge base and as an ACM for XML-level data integration.

Recently, Dudley and Kristensson discussed the interface design of IML systems and revealed the following four key challenges[83]:

(C1): Users can be imprecise and inconsistent. Users may introduce errors and inappropriate biases in the learning process.

(C2): There is a gap between user input and intent. For example, a data example that is not chosen as positive is not necessarily negative for the target concept.

(C3): Machine learning systems work with internal models, which may not be intuitively perceivable by the user. Furthermore, the system reaction of user inputs might not be understandable by the user.

(C4): Learning is often open-ended. In most cases, there exists more than one solution, and the user might want to discover more useful or interesting results.

The fundamental problem behind these challenges is the unpredictable behavior of both the user and the learner. To systematically overcome these challenges, the authors of [83] proposed a generalized workflow for designing comprehensive IML processes:

Feature selection (W1): the user shall be able to select features that are used in the machine learning algorithm. Amershi et al. also reported that people naturally want to provide more instructions to the learner for improving performance, including suggesting alternative features or denying currently employed ones [82]. For concept learning in AML, available features stem from the underlying ontology, i.e., the atomic classes, data properties, and object properties, which in turn, are transformed from classes and relations in AML. For example, the learned OWL class expression $\mathsf{Robot} \sqcap \mathsf{hasManufacturer}.$"$\mathsf{KUKA}$" contains the features Robot and hasManufacturer. Allowing the user to select features have two significant outcomes. First, *a prior* domain knowledge can be incorporated into the learning, which shall improve the performance. Second, because the learner may overfit to the current data examples, undesired features can be removed by the user.

Model selection (W2): the user can configure model parameters or select alternative models used in learning. Models, in general, refer to the component of a machine learning system that maps inputs to outputs of the learner, e.g., the structure of a neural network or the configuration of a support vector machine. For most concept learning systems, the model is the search tree constructed by traversing the concept space. In this context, model selection refers to the traversing strategy and the associated configuration parameters, as shown in Section 4.2.

Model steering (W3): the user can inspect, modify, and create data examples to guide the learning process. In general, model steering is the main activity performed by the user and costs the most time. The situation is slightly different in concept learning because often a small set of data examples is enough. However, an efficient selection of data examples will improve the usability of a concept learning framework.

Quality assessment (W4): the user can evaluate the solutions (temporal or final) found by the learner. Usually, a machine learning system does not seek for perfect solutions but an acceptable one according to some evaluation metric. In contrast, as described in Section 4.3, concept learning in AML devotes to find the 100% correct solution for the current data set. Inevitably, this setting can cause overfitting that shall be avoided in general. Therefore, it is important to assess the quality of the learned results. More concretely, the user needs to understand why the solution is correct by reviewing the used features and decide whether these features are appropriate. For example, if the manufacturer is not important for the target concept, the user can prohibit the learner from using hasManufacturer for the learning task.

Termination assessment (W5):. The user can decide when to terminate the learning based on the learner's current performance and boundary conditions of the learning task. This activity is crucial because concept learning systems often suffer from relatively long learning time, which can exceed 5 minutes, as described in Section 4.4. Moreover, the performance may vary dramatically upon a minor change of the learning parameters or the data examples. Although both DL-Learner and AMLLEARNER provide a global parameter for setting the time upper-bound, termination assessment will further improve the usability of the learning system. In particular, if the user is able to inspect the intermediate solutions and resume the learning by exploiting the efforts that have been made, e.g., restart the learning using an intermediate result as the initial guess.

Transfer (W6): a learned model shall be appropriately deployed for use. In concept learning, this activity is straightforward because the learned OWL classes can be directly added to the ontology. For AML-based engineering, one needs to consider additional efforts that are required for model translation between OWL and AML.

The challenges and workflow proposed by Dudely and Kristensson can be used as a metric for assessing the usability AMLLEARNER, as shown in Table 5.3. It is evident that AMLLEARNER takes care of most activities in the workflow above. In particular,

Table 5.3.: Comparisons of the interactive features provided by AMLLEARNER and the metric proposed by Dudely and Kristensson [83].

Challenges	Features
C1	F5, F6
C2	none
C3	F4, F5
C4	F3

Workflow	Features
W1	F2, F5
W2	F2, F6
W3	F1, F5
W4	F3, F4, F5
W5	F4
W6	F7

ACMs and the demonstration of their associated features contribute to the transparency of AMLLearner, which is helpful for increasing the user's confidence and even improving the labeling quality [82]. Based on ACMs, the features F5 and F6 become the key for the communication between the learner and the user. Nevertheless, the challenge C2 is not tackled by AMLLEARNER since the employed learning algorithm explicitly requires negative examples. Moreover, AMLLEARNER currently only supports the stop and restart mechanism for reusing the intermediate results. In the future, it is also considered to reuse the search tree constructed in the previous learning cycle.

Part II.

Database-based Approach

6. Preliminaries and Related Work

In the first part of the thesis, methods are presented for building complex ontological knowledge from AML data. Although the learned OWL classes bridge the semantic gap between concepts in heterogeneous engineering systems, the ontology itself is often insufficient for data access and exchange tasks. One could argue that an OWL class describes an engineering concept and can be used for extracting data from this particular concept. Indeed, OWL class expressions can be translated to the so-called Description Logic Rules (DL Rules) [84], which in turn, are closely related to conjunctive queries [54] and can be used for data retrieval. However, DL Rules are designed for tight integration with DL ontologies, which enjoys the decidability of reasoning but suffers from its limited expressiveness. More specifically, DL Rules can only express tree-like interdependencies between variables. In other words, for any two distinct variables x and y, there is at most a single path between x and y in the body of the rule [54]. For example, it is impossible to state *"get all robots with two connected interfaces"*, which can be formulated as the following rule:

$$\forall x \forall y \forall z.\, (\mathsf{Robot}(x) \land \mathsf{hasEI}(x,y) \land \mathsf{hasEI}(x,z) \land \mathsf{connected}(y,z) \rightarrow Q(x)) \tag{6.1}$$

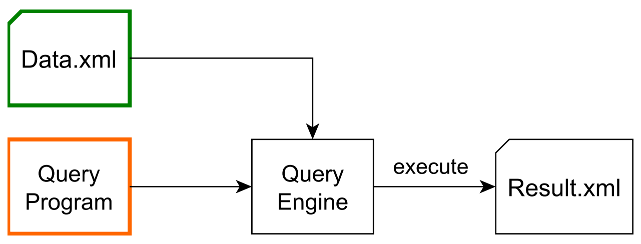
In rule 6.1, the left-hand side is called the rule body, and the right-hand side is called the rule head. More specifically, the predicate Q in the rule head indicates the answer to the query. This rule is not expressible in DL, because there are two dependency paths from x to z: one given by the atom $\mathsf{hasEI}(x,z)$ and the other one through the atoms $\mathsf{hasEI}(x,z)$ and $\mathsf{hasEI}(y,z)$. One way to work around with this problem is the integration of DL and Datalog, from which the so-called DL-safe rules and the SWRL[1] rule language emerge. However, to guarantee decidability, DL-safe rules need to satisfy the DL-safety by binding all variables in the rule to known individuals in the ontology. While this restriction might not be critical for data access, it limits the applicability of DL-safe rules in data exchange tasks, where the rule head often needs to invent new values using existential quantifications [28]. For example, the following rule states that *"for each pair of connected interfaces inside a robot, construct a connection object that comprises both interfaces"*:

$$\begin{aligned}\forall x \forall y \forall z.(\mathsf{Robot}(x) \land \mathsf{hasEI}(x,y) \land \mathsf{hasEI}(x,z) \land \mathsf{connected}(y,z) \rightarrow \\ \exists o.(\mathsf{Connection}(o) \land \mathsf{hasEI}(\mathsf{o},\mathsf{y}) \land \mathsf{hasEI}(\mathsf{o},\mathsf{z})))\end{aligned} \tag{6.2}$$

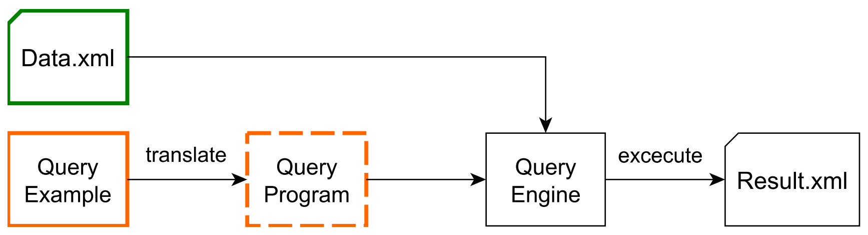
In rule 6.2, the variable o is existentially quantified and only exists in the rule head. That means, this rule requires the invention of a Connection object in the target AutomationML file, which can not be expressed using DL-safe rules.

In the literature, OBDA and OBDI approaches developed for engineering often require the manual implementation of the data access and exchange logic, e.g., using the SPARQL

[1] `https://www.w3.org/Submission/SWRL/`



(a) The universal XML query framework.



(b) The QBE-based XML query framework

Figure 6.1.: Comparison of conventional and QBE-based XML query frameworks.

query language [24]. However, SPARQL is not a standard programming language in industry and requires extra efforts for transforming data between its original formalism (e.g., AML) and the RDF graph [85].

The second part of the thesis aims at providing practical solutions for data access and exchange in AML-based engineering. Because AML is an XML-based data format, it is worth investigating whether any existing XML-driven technologies can be adopted for handling AML data. Indeed, the W3C has standardized several query languages for XML, including the XML Path Language (XPath) and the XML Query Language (XQuery). While XPath is primarily designed for addressing fragments of an XML document by navigating its tree structure, XQuery is a fully-fledged programming language for accessing and transforming XML data. Several XQuery engines exist on almost all mainstream operating systems and can either be used as stand-alone software or integrated as a library into existing engineering tools.

Fig. 6.1a shows the universal XML query framework employing a query engine. Besides the source XML data file (Data.xml), the user needs to implement the query program manually using e.g., XQuery. Nevertheless, XQuery is designed for experienced database developers and thus has a quite different programming style as the conventional ones, e.g., C# and Java. To support unskilled users by query construction, the so-called *Query By Example* (QBE) paradigm has been proposed [31, 32]. In contrast to the ontology-based approach, QBE does not require data transformation from XML to an ontology, but directly works with the inherent tree-structure of XML data. The core idea of QBE is to replace the manual implementation of the query programs with user-supplied query examples. Fig. 6.1b shows the QBE-based XML query framework. The only difference to Fig. 6.1a is that the query program is now automatically translated from the query examples.

This chapter first gives a brief introduction to XPath and XQuery in Section 6.1 for covering the notions and principles that are necessary for this thesis. Afterward, Section 6.2 discusses tree patterns, which are a fundamental tool for accessing tree-structured data and closely related to XPath. Tree patterns also inspired the theoretical research of XML data exchange, as described in Section 6.3. Finally, Section 6.4 discusses the related work on QBE-based approaches in the context of XML and AML.

6.1. XML Query Languages

In this thesis, an XML document is considered as an unordered[2] node-labeled tree $T = (V, F)$, where V is the set of nodes containing all XML elements and F the set of edges, i.e., the parent-child relationship between the nodes. The node with no parent is the root of the tree, and the nodes with no child are the leaf nodes. Each node $v \in V$ is a triple (id, tag, A) where id is the identifier, and tag is the related XML tag. Note that for AML, the set of possible values for tag stems from the vocabulary of the CAEX schema, which is partially shown in Fig. 2.2. Finally, A is a set of XML attributes associated with v. Again, for AML, A is a subset of the standardized XML attributes in IEC 62424, as described in Section 2.1.

Given an XML document tree $T = (V, F)$ as defined above, the basic utility of an XML programming language, such as XPath and XQuery, is to extract data in T, which can be XML elements, i.e., $v \in V$, and the associated XML attributes $v.A$. In addition, XQuery also supports the construction of a new XML document based on the extracted data.

6.1.1. XPath

The XML Path Language (XPath) is primarily designed for navigating an XML document. The term "path" refers to the ability to describe a route between nodes in the XML document. There have been three major revisions of XPath, the last being version 3.1 in 2017 [40]. In the following, the main features of XPath are introduced, which are already contained in the specification of XPath 1.0 [88].

The syntax of XPath contains three fragments: a) the Core XPath that contains the logical core of the language; b) the arithmetic features that include arithmetic and relational operations; c) the string manipulation features that provide string operations. This section does not elaborate on the details of XPath but discusses the fragment $XP^{/,//,[]}$ which can be defined by the following subset of the XPath's grammar [89]:

```
locpath ::= '/' locpath | locpath '/' locpath | locstep.
locstep ::=  axis ' :: ' ntst '[' bexpr ']'...'[' bexpr ']'.
bexpr   ::=  bexpr 'and' bexpr | bexpr 'or' bexpr |
             'not(' bexpr ')' | locpath.
axis    ::= 'self' | 'child' | 'descendant-or-self' | 'attribute'
```

XPath uses a location path (locpath) to navigate from a context node to another in the XML tree. A location path consists of a set of location steps (locstep) that are sequentially connected by the operator '/'. Each location step comprises the following three parts.

[2]In general, the sequence of elements appearing in an XML document is critical. However, because CAEX does not impose the ordering of its elements, this thesis disregards the ordering of XML elements inside an AML document. This specific setting is also often adopted in the literature on XML data access [86, 87].

An **axis** specifies the relationship between the nodes selected by the location step and the context node. A *self* axis, for example, selects the context node, and a *child* axis selects children of the context node. A *descendant-or-self* axis[3], which is defined as the transitive closure of the child axis, selects the context node and its descendants. An *attribute* axis selects the XML attributes of the context node.

A **node test** (ntst) denotes either a specific XML tag, e.g., `InternalElement`, or all tags using the wildcard symbol $*$.

A **predicate** is a boolean expression (bexpr) enclosed by a pair of square brackets that adds conditions to the selected nodes. Boolean expressions can be connected using the operators $\{\texttt{and}, \texttt{or}, \texttt{not}\}$. For example, the XPath expression in Formula 6.3 evaluates to true only if the selected `InternalElement` nodes have a child `InternalElement` node and a child `ExternalInterface` node.

$$\begin{aligned}\texttt{InternalElement}[\texttt{child} :: \texttt{InternalElement}\ \underline{\texttt{and}}\\ \texttt{child} :: \texttt{ExternalInterface}]\end{aligned} \tag{6.3}$$

XPath also provides the following relation operators (relop) for describing value constraints inside the predicates:

$$\text{relop} ::= \text{‘}=\text{’} \mid \text{‘}!=\text{’} \mid \text{‘}<\text{’} \mid \text{‘}<=\text{’} \mid \text{‘}>\text{’} \mid \text{‘}>=\text{’}.$$

For example, the XPath expression in Formula 6.4 is a two-step location path starting from the node `InstanceHierarchy`. The first step navigates to all descendant nodes from `InstanceHierarchy` which have the tag `InternalElement`. The second step navigates from those `InternalElements` to their `ExternalInterface` children whose XML attribute `RefBaseClassPath` has the value “`Communication`”.

$$\begin{aligned}&\texttt{InstanceHierarchy}/\texttt{descendant-or-self} :: \texttt{InternalElement}/\\ &\quad \texttt{child} :: \texttt{ExternalInterface}[\\ &\qquad \texttt{attribute} :: \texttt{RefBaseClassPath} = \text{“}\texttt{Communication}\text{”}]\end{aligned} \tag{6.4}$$

For brevity, XPath provides an abbreviated syntax, where `child`:: can be omitted from a location step, /`descendant-or-self`:: can be shortened as //, and `attribute`:: can be written as @. To further increase the readability of XPath expressions in this thesis, the following abbreviations are used to replace the full name of CAEX schema elements:

- IH for <u>I</u>nstance<u>H</u>ierarchy;
- IE for <u>I</u>nternal<u>E</u>lement;
- EI for <u>E</u>xternal<u>I</u>nterface;
- Attr for CAEX <u>Attr</u>ibute;
- RR for <u>R</u>ole<u>R</u>equirements;
- SRC for <u>S</u>upported<u>R</u>ole<u>C</u>lass;
- IL for <u>I</u>nternal<u>L</u>ink;

[3]In the literature of XML data access, the descendant-or-self axis is often referred to as the descendant axis for convenience.

Finally, `ref` is used for all kinds of references in AML, e.g., RefBaseRoleClassPath and RefRoleClassPath. With the abbreviated syntax, Formula 6.3 can be rewritten as:

$$\texttt{IE}[\texttt{IE}\ \underline{\text{and}}\ \texttt{EI}]$$

and Formula 6.4 can be simplified to:

$$\texttt{IH}//\texttt{IE}/\texttt{EI}[@\texttt{ref} = \text{“}\texttt{Communication}\text{”}]$$

6.1.2. XQuery

The XML Query Language (XQuery) is a strict superset of XPath. Therefore, every legal XPath program is also a valid XQuery program, but not vice versa. XQuery provides the so-called FLWOR expressions for building complex queries, i.e. **f**or, **l**et, **w**here, **o**rder by, and **r**eturn. Also, XQuery supports user-defined variables and functions. The latter one can be organized within *library modules* and can be called outside the modules. Since XQuery 3.0 [41], higher-order functions are also supported that can be used to build custom transitive closures. Note that the descendant axis of XPath is only the transitive closure of the child axis.

Listing 6.1 shows an example of an XQuery higher-order function that retrieves all descendant role classes of a given AML role class. Line 1 declares the function with its desired inputs, including the name and the library of an AML role class. Line 2 is an XQuery let clause that binds the child role classes of the input one to an XQuery variable $children$ by calling another XQuery function $\mathrm{getChildRC}$. The return clause in line 3 comprises an inner XQuery program and evaluates it for the $children$ that have been found. Line 4 is an XQuery for clause that iterates over each individual $child$ in $children$, while line 5 and 6 parse the name and the library tokens from the current $child$. Finally, line 8 returns the current $child$, and line 9 computes the descendants of $child$ by a recursive call.

Consider the role class inheritance hierarchy in Fig. 2.4a. For the following input:

$$\begin{aligned} roleName &= \text{“}\texttt{Resource}\text{”}, \\ libName &= \text{“}\texttt{AutomationMLBaseRoleClassLib}\text{”} \end{aligned}$$

The function $\mathrm{getDescendantRC}$ will return `DiscmanufacturingEquipment`, `Robot`, `Tool`, and `Machine`.

Listing 6.1 Retrieving all descendant role classes of a given AML role class

```
 1 declare function caex:getDescendantRC($roleName as xs:string,
       $libName as xs:string) as item()*{
 2     let $children := caex:getChildRC($roleName, $libName)
 3     return
 4         for $child in $children
 5         let $clib := caex215:getLibFromPath($child)
 6         let $cname := caex215:getNameFromPath($child)
 7         return
 8             $child,
 9             caex215:getDescendantRC($cname, $clib)
10 };
```

6.2. Tree Patterns

Tree patterns are a well-known concept for describing queries over a tree-structured data and are thus extensively studied in the field of XML databases. Historically, Chen et al. first coined out the term *twig query* as a node-labeled tree, which only has parent-child relationships between the connected tree nodes [90]. Afterward, twig query is extended to support the ancestor-descendant relationship in [91]. The general formalism of tree patterns[4] is proposed by Jagadish et al. in [92], which enriches twig queries with node predicates, i.e., constraints on the value of the nodes (XML elements) and their attributes. The following gives the definition of a tree pattern with some modifications to [92] that are necessary for this thesis:

Definition 6.1 (Tree Pattern)**.** A tree pattern is a node-labeled rooted tree $Q = (N, E)$, where:

- N is the set of nodes, E is the set of edges, and the root of the tree is the node with no parent;
- Each node $x \in N$ is represented as a triple (id, tag, C), where id is the identity of the node, tag is the related XML tag (the label) or the wildcard symbol $*$, and C is a set of constraints on x. Each constraint in C has the following form:

  $$(x.\texttt{value} \mid x.\texttt{attr}) \;\; \theta \;\; c$$

  where `value` is the text content of x, `attr` is the name of an XML attribute, c is a constant, and $\theta \in \{=, \neq, >, \geq, <, \leq\}$.
- An edge in E is a triple $(x, y, type)$, where x and y are identifiers of the connected nodes and $type \in \{pc, ad\}$ determines whether the edge describes a parent-child (pc) or an ancestor-descendant (ad) relationship from x to y. □

The semantics of tree patterns is given via *embeddings* [86], which is defined as follows:

Definition 6.2 (Embedding)**.** Let $Q = (N, E)$ be a tree pattern and $T = (V, F)$ be an XML document tree. An embedding from Q to T is a function $embedding : N \to V$ that satisfies:

- for each $x \in N$ with $x.tag \neq * : x.tag = embedding(x).tag$.
- for each $(x, y, pc) \in E : (embedding(x), embedding(y)) \in F$.
- for each $(x, y, ad) \in E : (embedding(x), embedding(y)) \in F^+$. The symbol F^+ stands for the transitive closure of all edges in F.
- for each $x \in N$: all value constraints in $x.C$ are satisfied by $embedding(x)$. □

The meaning of a tree pattern $Q = (N, E)$ w.r.t. an XML document $T = (V, F)$ is thus given as the set of all embeddings. That is:

$$Q(T) := \{e \mid e \text{ is an embedding from } Q \text{ to } T\}$$

Based on the Definition 6.1, a *tree pattern query* (TPQ) is a node-labeled rooted tree $Q = (N, E)$ where each node $n \in N$ is a 4-tuple (id, tag, r, C, D), in which r is a boolean flag that indicates whether n is returned or not, and D is a set of additional conditions[5]

[4]It was called *Pattern Tree* in [92].

[5]This is an extension of the standard definition of TPQs in the literature for incorporating rich query constraints.

attached to n. A returned node corresponds to the answer to the query. The meaning of id, tag, and C remains the same as in Definition 6.1. Consequently, the meaning of a TPQ $Q = (N, E)$ w.r.t. an XML document $T = (V, F)$ and a returned node $x \in N$ is the set of all embeddings of x in F. That is:

$$Q(T, x) := \{e(x) \mid e \text{ is an embedding from } Q \text{ to } T\}$$

The theoretical study of TPQs showed that TPQs are closely related to XPath. As mentioned in [93], tree patterns with a single returned node capture the XPath fragment $XP^{/,//,[]}$, where child and descendant edges correspond to the XPath axes $/$ and $//$ respectively. Hence, two tree pattern nodes connected by an edge represent a single location step in XPath, and the full path from the root to the returned node of a tree pattern constitutes a location path. Note that in tree patterns, the boolean operators $\{\texttt{or}, \texttt{not}\}$ are only used for value constraints, while XPath supports the general notion of boolean expressions.

6.3. Foundations of XML Data Exchange

The pioneering work on the general data exchange problem was the Clio project[6], which appeared in 1999 [94]. Since then, data exchange has been extensively studied for relational databases [28]. The results from the Clio project were integrated into several IBM products, including the IBM Relational Data Architect and the IBM InfoSphere FastTrack [29]. Advances of relational data exchange encouraged the study on XML-based data exchange problems [95, 96], where essential principles are adopted from the relational domain.

The theory of data exchange is grounded on the concept called *schema mapping* [28]. The first comprehensive work on XML schema mapping was reported by Arenas and Libkin [95], which was extended by Amano et al. for covering features including join relations and Skolem functions [96]. The basic ingredient of XML schema mapping is the so-called *tree-pattern* formulae, defined as follows:

Definition 6.3 (Tree-pattern formulae (TPF))**.** Tree-pattern formulae are given by the grammar:

$$\begin{aligned} \varphi &:= l(a_1 = t_1, \cdots, a_n = t_n)[\lambda] \\ \lambda &:= \epsilon \mid \varphi \mid //\varphi \mid \lambda, \lambda \end{aligned}$$

where:

- ϵ is the empty string;
- l is a label (an XML tag) or the wildcard $*$;
- $a_1, \cdots, a_n$ are XML attributes;
- $t_1, \cdots, t_n$ is a set of terms. Terms are defined inductively: each variable is a term; for a function symbol f of arity k, $f(t_1, \cdots, t_k)$ is also a term if $t_1, \ldots, t_n$ are terms;

[6]http://dblab.cs.toronto.edu/project/clio/index.php

- [ ] correspond to the XPath predicate and // correspond to the XPath descendant axis. □

The original formulation in [96] allows sequences (orders of the sub-patterns) and inequalities between terms, which are not required for AML data exchange and are neglected in this definition. Furthermore, equality between terms is represented by using the same term in place of different attributes. For example, $l(a_1 = t_1, a_2 = t_1)$ means that the attributes a_1 and a_2 shall have the same value. If the context is clear, then $\vec{t}$ is used in the place of the equations $(a_1 = t_1, \cdots, a_n = t_n)$, such that the formula $l(a_1 = t_1, \cdots, a_n = t_n)[\epsilon]$ is written as $l(\vec{t})$.

Recall Definition 6.1. It can be observed that TPFs are basically the same as tree patterns, besides the following three differences in terms of value constraints. First, TPFs do not consider text content of XML test nodes. Second, while tree patterns allow value comparisons using various operators, including $\{>, \geq, <, \leq\}$, TPFs for schema mapping only supports equality and inequality. Finally, TPFs use terms instead of constants to collect values from the source XML.

Let φ be a TPF. The set of all variables occurring in a tree pattern formula is denoted as $\mathsf{Var}\,\varphi$, while $\varphi(\vec{x})$ indicates that $\vec{x}$ are the variables in $\mathsf{Var}\,\varphi$. The semantics of a TPF w.r.t. an XML document tree is defined as follows:

Definition 6.4 (Semantics of Tree-pattern Formulae)**.** Let $T = (V, F)$ be an XML document tree, $\varphi(\vec{x})$ be a TPF, and F a valuation function that assigns to each $f(t_1 = b_1, \cdots, t_k = b_k)$ a value (where b_i is the concrete value of t_i). Let $\vec{a}$ be an interpretation of $\vec{x}$ (i.e., $x_i = a_i$), then $\varphi(\vec{a})$ is satisfiable in a node n of T, written as $(T, n, F) \models \varphi(\vec{a})$, when one of the following case is satisfied (each case handles one possible form of the TPF $\varphi(\vec{x})$ in Definition 6.3):

1. $(T, n, F) \models l(\vec{t})$, if $n.tag = l$ or $l = *$, and $\vec{t}$ interpreted under F and $\vec{a}$ correspond to the attributes of n. Note that if $\vec{t}$ does not contain any function symbol, i.e., each t_i is one element of $\vec{x}$, then the interpretation of $\vec{t}$ only depends on $\vec{a}$.
2. $(T, n, F) \models l(\vec{t})[\varphi']$, if $(T, n, F) \models l(\vec{t})$ and $(T, n', F) \models \varphi'$ for some child node n' of n;
3. $(T, n, F) \models l(\vec{t})[//\varphi']$, if $(T, n, F) \models l(\vec{t})$ and $(T, n', F) \models \varphi'$ for some descendant node n' of n;
4. $(T, n, F) \models l(\vec{t})[\lambda_1, \lambda_2]$, if $(T, n, F) \models l(\vec{t})[\lambda_1]$ and $(T, n, F) \models l(\vec{t})[\lambda_2]$.

Finally, if $\varphi(\vec{a})$ is satisfiable in the root node of T, then $(T, F) \models \varphi(\vec{a})$. □

Based on TPF, an XML data exchange setting can be defined in terms of source-to-target dependencies as follows:

Definition 6.5 (Source-to-Target Dependencies (STD))**.** A STD has the form:

$$\varphi(\vec{x}, \vec{y}) \rightarrow \psi(\vec{x}, \vec{z}) \tag{6.5}$$

where:

- $\vec{x}, \vec{y}, \vec{z}$ are vectors of variables, and $\vec{y}$ and $\vec{z}$ are disjoint. The variables $\vec{x}$ are called the bound variables in the STD.
- $\varphi(\vec{x}, \vec{y})$ and $\psi(\vec{x}, \vec{z})$ are TPFs over the source and the target XML schema definitions, respectively. By viewing a STD as a logic rule, $\varphi(\vec{x}, \vec{y})$ is called the rule body and $\psi(\vec{x}, \vec{z})$ is called the rule head.

Let S and T be two XML document trees. The pair (S, T) satisfies the above STD w.r.t. a valuation function F, if for each possible interpretation $\vec{a}$ for $\vec{x}$ and $\vec{b}$ for $\vec{y}$ with $(S, F) \models \varphi(\vec{a}, \vec{b})$, there is a tuple of values $\vec{c}$, such that $(T, F) \models \psi(\vec{a}, \vec{c})$. □

Definition 6.6 (Data Exchange Setting)**.** An XML data exchange setting is a triple $(D_\mathbf{S}, D_\mathbf{T}, \Sigma)$, where $D_\mathbf{S}$ and $D_\mathbf{T}$ are the source and target XML schema definitions (or DTDs), and Σ a set of STDs between $D_\mathbf{S}$ and $D_\mathbf{T}$.

Let S be a source XML tree conforming to $D_\mathbf{S}$. A target XML tree T that conforms to $D_\mathbf{T}$ and satisfies all STDs in Σ is a solution for S. □

Definition 6.6 provides a universal formalism for describing XML data exchange. In particular, because function symbols are allowed, the right-hand side of an STD is capable of expressing Skolem functions, which are necessary for grouping collected data from the source XML document [97]. However, as discussed above, neither the data comparison operators $\{>, \geq, <, \leq\}$ nor the content of XML text nodes is considered in Definition 6.3, both of which are essential for the filtering of the source XML document. The latter is of great importance in AML data exchange because the source and target data in AML-based engineering have the same XML schema (c.f. Fig. 2.5). Therefore, in Chapter 8, a more appropriate data exchange setting will be introduced for AML.

6.4. Query By Example

Zloof first introduced Query By Example (QBE) for querying relational databases using a visual paradigm [31]. In this work, the user can directly formulate data examples in appropriate database tables to describe the query result. Table 6.1 shows a query example involving two database tables, each of which contains one query example. The syntax of the query examples distinguishes between two kinds of entities: the underlined strings represent variables, and the normal strings represent constant values. The letter P stands for “print" and indicates the output of the query. Because the variable `ROD` appears in both the $\mathbf{SALES}$ and the $\mathbf{SUPPLY}$ table, the query looks for all departments which have at least one item from the supplier `PARKER`, or mathematically formulated as:

$$\{x : \exists y\,((x, y) \in \mathbf{SALES} \wedge (y, z) \in \mathbf{SUPPLY})\}$$

DEPARTMENT	ITEM
P.<u>TOY</u>	<u>ROD</u>

ITEM	SUPPLIER
<u>ROD</u>	PARKER

Table 6.1.: A query example in QBE [31]. The left table contains one entry for the relation $\mathbf{SALES}$ and the right one contains one entry for the relation $\mathbf{SUPPLY}$.

Given a relational database, a user can formulate the desired query by means of the above syntax, and an interpreter is responsible for translating the query example into concrete database query programs. The result of the query contains data elements from possibly different database tables. Therefore, QBE is primarily designed for data access.

Soon after the release of XQuery, a similar idea was adopted in [32] to develop the graphical language XQBE (XQuery By Example) for querying XML data. Because the general structure of an XML document is a tree, XQBE provides a tree-based syntax for building the examples. Moreover, XQBE is designed for data exchange such that a "target" XML document can be generated from the data in a "source" XML document. To this end, XQBE supports two kinds of tree models: a *source* tree for defining what data from the source XML file is needed and a *construct* tree for specifying the constitution of the target XML file. Fig. 6.2 shows the query example Q5 in [32]. The example has two parts, which are separated by the line in the middle. The left part represents the source tree and the right part describes the target tree. Each box is a so-called E-Node that stands for an XML element in the corresponding document. A target E-Node denotes a replication of a source XML element, either by a binding edge to the corresponding source E-Node, e.g., the two `book` E-Nodes in Fig. 6.2, or implied by an ancestor E-Node in the target tree which has a binding edge, e.g., the `title` and `author` elements.

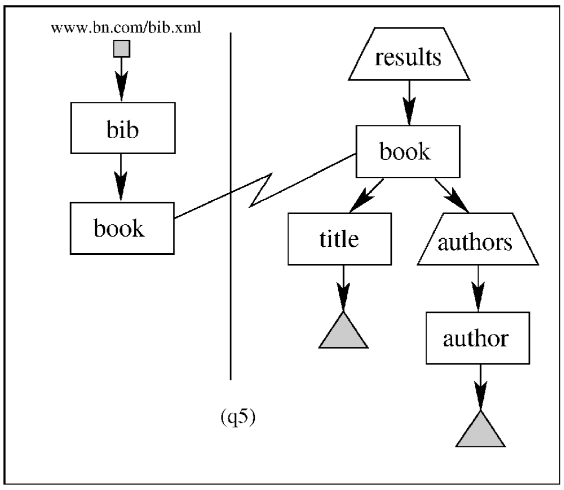


Figure 6.2.: The query example Q5 in XQBE [32].

The target tree can have single-tag Trapezoidal nodes (T-Nodes), represented with the shorter edge on the bottom, and set-tag T-Nodes, represented with the shorter edge on the top. A single-tag T-Node results into a new XML tag for each instance of its child nodes, while a set-tag T-Node will be constructed only once for all the instances below it. In Fig 6.2, both the `results` and the `author` elements are set-tag T-Nodes. Therefore, the `results` element will be constructed only once for all books from the source XML document. Similarly, one `authors` element will comprise all the authors of one book in the source XML document.

Now consider the query example Q6 from [32], as shown in Fig. 6.3. Q6 contains two single-tag T-Node `aBook` and `by`. Therefore, for each book from the source XML document, an `aBook` element will be constructed, in which a `by` element is generated for each author of the book.

One issue with single-tag T-Nodes is the computation of Cartesian products. Fig. 6.4 shows the query example Q7 from [32] that has two different E-Nodes `author` and

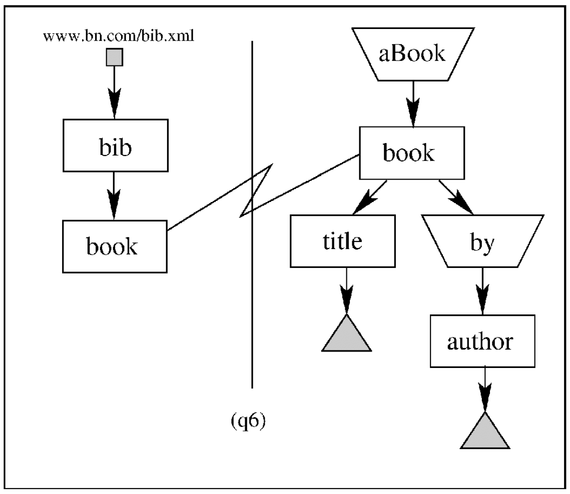


Figure 6.3.: The query example Q6 in XQBE [32].

`title` under the single-tag T-Node `result`. The query constructs a result element for each book, but only replicates the author and title of the book. For example, the book *Foundations of Semantic Web Technologies* has three authors *Pascal Hitzler, Markus Krötzsch*, and *Sebastian Rudolph*. Then three `result` elements will be constructed, each of which contains the book title and one of the three authors.

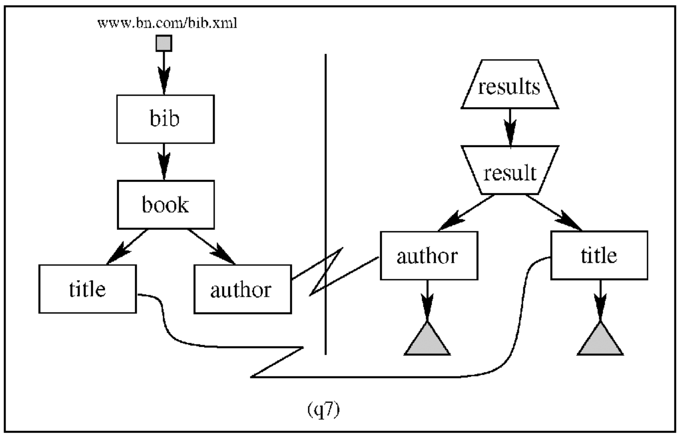


Figure 6.4.: The query example Q7 in XQBE [32].

While XQBE is relatively expressive and covers a large subset of XQuery, the more recent approach VXQ [33] focuses on the usability of the graphical language and concerns about constructing more efficient queries. The authors of VXQ argued that XQBE requires a large number of user interactions for composing a complex query, which is partially due to the rather verbose graphical syntax. Furthermore, VXQ employs query optimization techniques for improving run-time performance. Most notably, VXQ utilizes XPath predicates for filtering the source XML document such that the deep, nested XQuery programs will be evaluated over a smaller set of source XML data objects.

It is worth noting that both XQBE and VXQ are general QBE approaches that can be applied to all kinds of XML documents. However, these general-purpose approaches are not well suited for AML, because (i) they employ dedicated graphical notions which require additional learning efforts, and (ii) they cannot express AML-specific semantics.

In particular, point (ii) is essential for querying AML data. For example, the user might want to extract all internal elements referring to the role class `Resource`, while considering all the subclasses of `Resource`. Such queries can not be expressed with XQBE and VXQ.

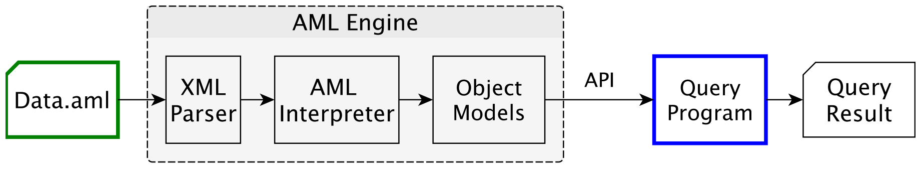
Recently, there has been attempts towards a dedicated QBE framework for AML, presented by Wimmer and Mazak [85]. The authors introduced the query language AQL (Automation Query Language) and its two sub-languages AQDL (AML Query Definition Language) and AQRL (AML Query Result Language) based on the Eclipse Modeling Framework (EMF) [7]. AQDL is used for creating a query model, and AQRL is used to present query results with some meta-information. However, AQRL is not comparable with the construct part of XQBE since it cannot specify data exchange rules, such as projection, join, renaming, etc. Based on the AML engine developed in [98], the main idea of AQL was to model queries using CAEX schema elements and to match them with the source AML data using an interpreter. Compared with XQBE and VXQ, AQL is more intuitive for AML users. Nevertheless, for constructing a query, extensive knowledge about EMF is necessary, since both the syntax and the user interface are based on EMF. Moreover, the semantics of AQDL is defined with informal notions which do not cover advanced AML modeling features, e.g., the class inheritance hierarchy and the connections between engineering objects. Finally, query execution is not separated from query interpretation and depends on the underlying AML engine.

In conclusion, XQBE and similar general-purpose visual languages are not well suited for AML, while AQL requires specific software. A better approach is to develop an AML-specific query language while adhering to the standardized techniques from the XML domain. To this end, Chapter 7 introduces the *AutomationML Query Template* (AQT) for accessing AML data using AML itself as the modeling framework, and Chapter 8 presents an extension of AQT to enable data exchange in AML-based engineering.

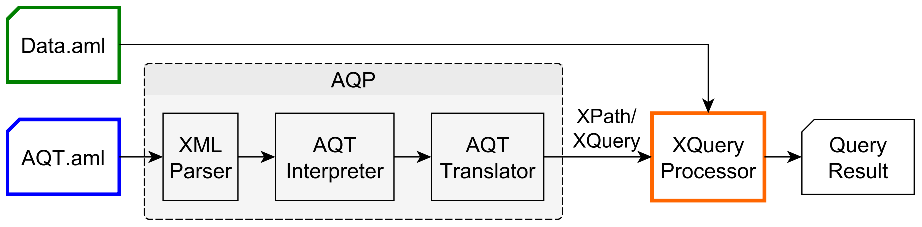
[7]`https://www.eclipse.org/modeling/emf/`

7. By-example Data Access for AutomationML

This chapter investigates the problem of accessing engineering data stored in AML instance hierarchies, which represent the topology of real production plants and hold the actual data values. Conventionally, extracting AML data makes use of an application programming interface (API) provided by a dedicated AML engine, as shown in Fig. 7.1a. The engine usually begins with a standard XML parser that parses a source AML document into an XML Document Object Model (DOM), which is then processed with an AML interpreter. The outputs of the interpreter are internal object models of the engine. Although the semantics of these internal models need to reflect the meaning of the AML terms as defined in the standard, the implementation depends on the employed programming techniques. As a consequence, the API is usually distinct among AML engines. For example, the "standard" C# implementation provided by the AML Consortium is based on the .NET framework[1], and the Java implementation published by Mayerhofer et al. [98] is based on the Eclipse Modeling Framework (EMF)[2], just to name a few. Thus, in order to develop AML query facilities in the engineering toolchain, the knowledge of AML, the knowledge of the internal object models, and the practical experiences of the API are necessary.



(a) The conventional pipeline for accessing AML data.



(b) The AQT based pipeline for accessing AML data.

Figure 7.1.: The conventional and AQT-based pipelines for accessing AML data (green). The blue components are provided by the user. The orange box in 7.1b represents any standard-conform XQuery processor.

Inspired by the Query by Example (QBE) approach [31][32], this chapter introduces the *AutomationML Query Template* (AQT) to foster efficient AML data queries for

[1]`https://www.nuget.org/packages/Aml.Engine`
[2]`https://github.com/amlModeling/caex-workbench`

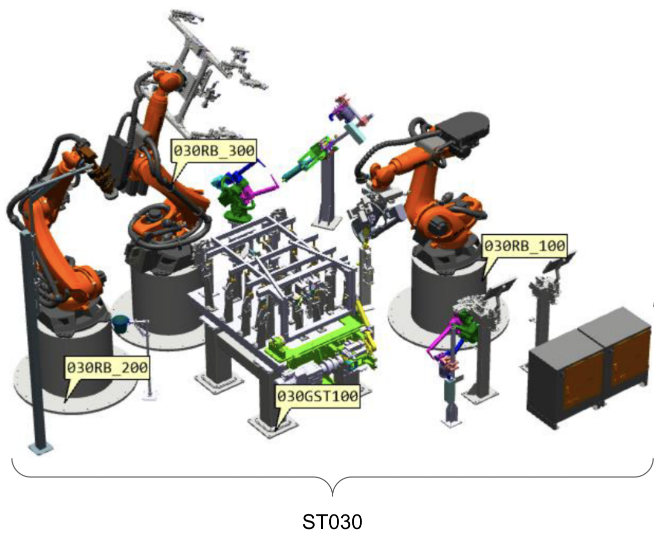


Figure 7.2.: An excerpt of the robot cell provided by the AML association. The object ST030 is the top-level internal element.

domain experts. AQT has two essential design principles. First, AQT exploits the modeling features of AML and employs the AML syntax for query construction. Second, the semantics of AQT is defined based on the notion of tree pattern queries (TPQ), as described in Section 6.2. Therefore, AQTs can be automatically translated to XPath [40] and XQuery [41] programs.

Fig. 7.1b shows the AQT-based pipeline for querying AML data. Instead of providing the query program as in Fig. 7.1a, the user supplies AQTs in an AML file while the so-called *AML Query Processor* (AQP) generates the corresponding XPath or XQuery program, that can be executed on any standard-conform XQuery processors. For convenience, the AML files “Data.aml” and “AQT.aml” are called the **source file** and the **query file**, respectively.

To exemplify the utilities of AQT, Fig. 7.2 shows an excerpt of a robot cell that is publicly available on the website of the AML association[3]. The original AML file has 323 internal elements, 277 external interfaces, and 1829 attributes. Fig. 7.3a shows the simplified XML serilization of the AML model. For the sake of brevity, the following simplifications are undertaken:

(a) Objects names are changed to indices. For example, the internal element names ST030, 030RB_100, 030RB_200, 030RB_300, and 030GST100 are simplified to $1, 2, 6, 10$, and 13, respectively;

(b) Full paths of role classes are replaced by class names. For example, the role class path `AutomationMLDMIRoleClassLib/DiscManufacturingEquipment/Robot` is replaced by `Robot`;

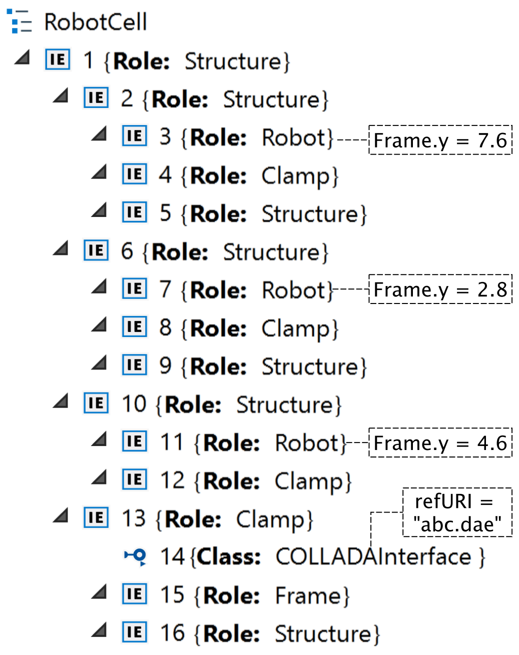
(c) Some nested XML elements are omitted in Fig. 7.3a, e.g., the internal elements under 030RB_100, 030RB_200, 030RB_300, and 030GST100.

Note that while engineering objects in AML are only enforced to be uniquely identifiable by their own UUIDs, each object in this excerpt is also given a unique name.

[3]`https://www.automationml.org/o.red/uploads/dateien/1506092067-AML_RobotCell.pdf`

```
<InstanceHierarchy ID="AF138E59-0000-17AC-556E-B1150000319B" Name="RobotCell">
   <InternalElement ID="AF138E59-0000-17AC-556E-B81A0000323E" Name="1">
      <InternalElement ID="AF138E59-0000-17AC-556E-B81A0000323F" Name="2">
         <InternalElement ID="AF138E59-0000-17AC-556E-B81A00003240" Name="3">
            <Attribute Name="Frame">
               <Attribute Name="y" Unit="m" AttributeDataType="xs:double">
                  <Value>7.6</Value>
               </Attribute>
               <!-- further coordinates of the Frame attribute as nested attributes-->
            </Attribute>
            <RoleRequirements RefBaseRoleClassPath="Robot" />
         </InternalElement>
         <!--internal elements 4 and 5 as nested objects-->
      </InternalElement>
      <InternalElement ID="AF138E59-0000-17AC-556E-B8BD000032A1" Name="6">
         <!--internal elements 7, 8, and 9 as nested objects-->
         <RoleRequirements RefBaseRoleClassPath="Structure" />
      </InternalElement>
      <InternalElement ID="AF138E59-0000-17AC-556E-B961000032DA" Name="10">
         <!--internal elements 11 and 12 as nested objects-->
         <RoleRequirements RefBaseRoleClassPath="Structure" />
      </InternalElement>
      <InternalElement ID="AF138E59-0000-17AC-556E-BC030000331D" Name="13">
         <ExternalInterface Name="14" RefBaseClassPath="COLLADAInterface"
            ID="32D1045E-5069-45BB-9DFB-DDA17BC7F27B">
            <Attribute Name="refURI" AttributeDataType="xs:anyURI">
               <Value>abc.dae</Value>
            </Attribute>
         </ExternalInterface>
         <!-- internal elements 15 and 16 as nested objects-->
         <RoleRequirements RefBaseRoleClassPath="Clamp" />
      </InternalElement>
      <RoleRequirements RefBaseRoleClassPath="Structure" />
   </InternalElement>
```

(a) The simplified XML serialization of the excerpt.



```
RobotCell
  ◢ IE 1 {Role: Structure}
    ◢ IE 2 {Role: Structure}
      ◢ IE 3 {Role: Robot}----Frame.y = 7.6
      ◢ IE 4 {Role: Clamp}
      ◢ IE 5 {Role: Structure}
    ◢ IE 6 {Role: Structure}
      ◢ IE 7 {Role: Robot}----Frame.y = 2.8
      ◢ IE 8 {Role: Clamp}
      ◢ IE 9 {Role: Structure}
    ◢ IE 10 {Role: Structure}
      ◢ IE 11 {Role: Robot}--Frame.y = 4.6
      ◢ IE 12 {Role: Clamp}
    ◢ IE 13 {Role: Clamp}                refURI = "abc.dae"
           14 {Class: COLLADAInterface }
      ◢ IE 15 {Role: Frame}
      ◢ IE 16 {Role: Structure}
```

(b) A screenshot of the excerpt in the AML Editor.

Figure 7.3.: The simplified AML model of the excerpt represented in: (a) its raw XML serialization (b) in the AML Editor.

Table 7.1.: Modeling features for querying AutomationML data. The Examples column contains examples of AQT that cover the corresponding modeling features in the same row. The examples q1-q8 will be presented in Section 7.2.2, and the handling of AutomationML advanced features will be presented in Section 7.3.3.

Modeling Features	**Examples**
AutomationML Structure Features	
(S1) structurally nested objects and interfaces	q1 - q8
(S2) class-instance relationship	q1 - q8
(S3) object identification via UUID	q2
AutomationML Property Features	
(P1) attribute name	q5 - q7
(P2) structurally nested attributes	q5
(P3) explicit (default) attribute value	q6
(P4) attribute unit	q5
(P5) attribute data type	q5, q6
(P6) attribute semantic reference	q7
(P7) attribute value requirements (nominal and ordinal)	q5
AutomationML Advanced Features	
(A1) class inheritance hierarchy via class-class relations	cf. Section 7.3.3
(A2) instance-instance relations (partner EI in the source file)	cf. Section 7.3.3
(A3) instance-instance relations (partner EI in the query file)	cf. Section 7.3.3
XML Data Query Features	
(X1) retrieving objects at any position in the XML structure	q1-q8
(X2) retrieving descendant objects in the XML structure	q5, q7
(X3) querying related data simultaneously (multi-return)	q4, q8
(X4) cardinality restrictions	q8

Fig. 7.3b is a screenshot of the excerpt in the AML editor that shows the structure of the instance hierarchy more compactly. For example, the labels {**Role**: `Structure`} in Fig. 7.3b correspond to the role requirements with the class reference `Structure` in Fig. 7.3a. Also, Fig. 7.3b does not explicitly contain any CAEX attributes. For explaining the query examples later in Section 7.2.2, some CAEX attributes are shown as dashed boxes with their names and value.

7.1. Requirements for querying AutomationML

Table 7.1 shows a categorization of the AML modeling features described in Section 2.1. Apparently, all structure (S1-S3) and property (P1-P7) features are essential for querying AML because they are used for storing the engineering data. For example, the user may be interested in the value of a CAEX attribute (P3) or the internal elements that refer to a particular role class (S2). Moreover, the class inheritance hierarchy (A1) is an advanced semantic feature of AML, which goes beyond a single class-class relation and allows the user to build a taxonomy of the domain concepts. Feature A1 thus allows expressing the query "*return all internal elements of the role class* `Resource` *or any of*

```
<InstanceHierarchy Name="Raw">
  <InternalElement Name="IE1" ID="fec46504-2eee-4b7e-b5df-9acc4a2c5393">
    <InternalElement Name="IE2" ID="3fb9fc26-1dbd-49b8-b3f1-e03920931a14" />
    <RoleRequirements RefBaseRoleClassPath="Structure" />
  </InternalElement>
</InstanceHierarchy>
```

Figure 7.4.: An AML model that can serve as a raw query template.

its descendant sub-classes". According to Fig. 2.4a, This query shall retrieve internal elements that refer to at least one of the role classes `Resource`, `Robot`, `Clamp`.

With the advanced features A2 and A3, the connections between external interfaces are determined, whereby a distinction is made whether the partner interface is located in the source file (A2) or in the query file (A3). These two features will be discussed more in Section 7.3.3. Finally, since AML is an XML-based data format, the common XML query features shall be supported. Informally, X1 devotes to specify the position of the data objects that shall be returned from the query, X2 supports the modeling of structural relations between the data objects, X3 allows retrieving a set of co-existing data objects simultaneously, and X4 supports counting the occurrence of data objects. More details and examples of these features will be presented in Section 7.2.2.

7.2. The AutomationML Query Template

The objective of the AutomationML Query Template (AQT) is to describe queries using AML models. Intuitively, a particular AML model can be used as a raw query template to extract data of that kind. Consider the query q1 that "*returns all internal elements that refer to the role class* `Structure` *and contain at least a nested internal element*". Fig. 7.4 shows the AML model that captures the structure of q1. However, the meaning of the raw query template is ambiguous because it does not specify which internal element in the template is the target of the query. In other words, while the raw query template can be used for pattern matching against a source AML file, it is unclear what is returned from the query: shall the outputs be source data objects that match to `IE1` or `IE2`? Indeed, the raw query template fails to capture the XML query features (X1-X4) and cannot be used for querying AML data.

7.2.1. Syntax of AQT

To tackle the problems of the raw query template, AQT defines the following CAEX attributes acting as query-specific configuration parameters.

- `returned`: whether the associated AML model belongs to the answer of the query. Intuitively, only returned AQT nodes are handled as targets of the query, while the others constitute the constraints of the query.
- `descendant`: whether the associated AML model is a descendant of its direct predecessor in the XML tree structure.

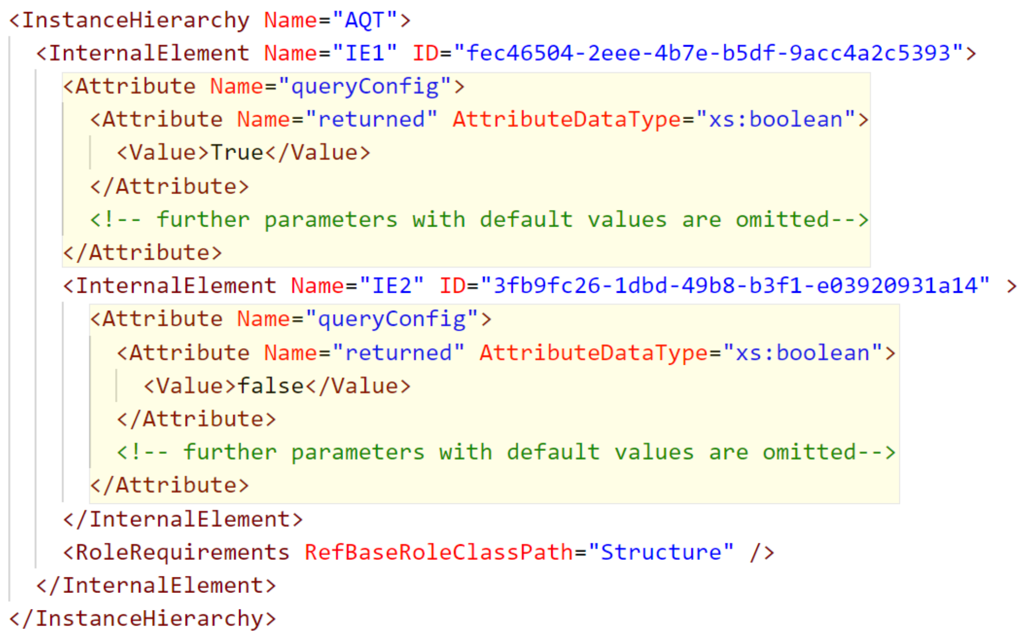
Table 7.2.: The configuration parameters of AQT.

Name	Type	Default	Features in Table 7.1	Implementation in XPath/XQuery
`returned`	`bool`	`false`	(X1, X3)	`return()`
`descendant`	`bool`	`false`	(X2)	descendant axis //
`identifiedById`	`bool`	`false`	(S3)	`@ID` = `id`
`identifiedByName`	`bool`	`false`	(P1)	`@Name` = `name`
`minCardinality`	`integer`	`1`	(X4)	`count()` >=
`maxCardinality`	`integer`	-1	(X4)	`count()` <=

- `identifiedById`: whether the associated AML model is uniquely identified by its UUID. This is practical for querying AML data, since each engineering object (internal element or external interface) in an AML file has a unique ID.
- `identifiedByName`: whether the associated AML model is uniquely identified by its Name. According to the AML standard, a CAEX attribute is identified by its name within the hierarchy level in a nested attribute structure.
- `minCardinality`: required minimum number of matches found for the associated AML model.
- `maxCardinality`: required maximum number of matches found for the associated AML model. Note that `maxCardinality` = `0` mimics a logical negation in query modeling, where the associated AML model shall not be matched in the source file.

Table 7.2 enumerates these parameters with their data type, default value, and implementation in XPath/XQuery. Now, reconsider the query q1 as described above. The user can first construct the raw query template (cf. Fig. 7.4) with the AML editor. The next step is to assign the configuration parameters to the raw query template. Because q1 aims to extract the outer internal element that refers to the role class `Structure`, the user sets `returned` = `true` for `IE1` and `returned` = `false` for `IE2`. Fig. 7.5 shows the XML text of the resulting AQT. Now, both internal elements are equipped with a CAEX attribute `queryConfig`, which comprises the configuration parameters as sub-attributes. This AQT therefore enforces `IE1` to be the answer of the query while the nested `IE2` represents a structural constraint of `IE1`. Note that because q1 does not restrict the UUID of the internal elements, the UUIDs in Fig. 7.5 are insignificant for the query and can be automatically generated from the AML editor. Later in Section 7.2.2, a query example that makes use of the UUID will be demonstrated.

Because AML has an XML-based syntax, each AQT can be formally represented by a tree $M = (V, F)$ where V is the set of nodes containing all XML elements excluding the `queryConfig` and its sub-elements, and F is the set of edges, i.e., the parent-child relationship between the nodes. The node with no parent is the root of the tree. Each node $v \in V$ is a 4-tuple (id, tag, A, B), where id is the identifier, and tag is the related XML tag. Note that id is different from a UUID since the latter one only applies to an AML IE or EI. A is the set of XML attributes (not the CAEX Attrs) and the XML text value associated with v, e.g., `Name` and `ID` in Fig. 7.5. Finally, B is the set of configuration parameters attached to v. Although these parameters are serialized as CAEX Attrs (cf. Fig. 7.5), they are not treated as nodes in V.



```
<InstanceHierarchy Name="AQT">
  <InternalElement Name="IE1" ID="fec46504-2eee-4b7e-b5df-9acc4a2c5393">
    <Attribute Name="queryConfig">
      <Attribute Name="returned" AttributeDataType="xs:boolean">
        <Value>True</Value>
      </Attribute>
      <!-- further parameters with default values are omitted-->
    </Attribute>
    <InternalElement Name="IE2" ID="3fb9fc26-1dbd-49b8-b3f1-e03920931a14" >
      <Attribute Name="queryConfig">
        <Attribute Name="returned" AttributeDataType="xs:boolean">
          <Value>false</Value>
        </Attribute>
        <!-- further parameters with default values are omitted-->
      </Attribute>
    </InternalElement>
    <RoleRequirements RefBaseRoleClassPath="Structure" />
  </InternalElement>
</InstanceHierarchy>
```

Figure 7.5.: The AQT for q1 in the AML syntax. The highlighted parts illustrate the difference to the raw query template in Fig. 7.4. The configuration parameters that take their default values are omitted for sake of brevity.

Moreover, B can be empty for some nodes in V, because CAEX Attrs can only be assigned to ICs, RCs, SUCs, IEs, EIs, and Attrs. Therefore, other $\mathrm{CAEXBasicObjects}$ (cf. Fig. 2.2), e.g., the RR element in Fig. 7.5, cannot carry these configuration parameters and are called *conditional* $\mathrm{CAEXBasicObjects}$. This design has several consequences. First, conditional $\mathrm{CAEXBasicObjects}$ can not be returned from the query. For example, the RR node in Fig. 7.5 cannot be defined as the answer to the query. However, this is not a severe limitation since conditional $\mathrm{CAEXBasicObjects}$ are not designed for storing engineering data but rather for stating constraints about the data. Second, conditional $\mathrm{CAEXBasicObjects}$ can not be defined as a descendant object in the XML structure. For example, queries that "*return* IE*s with a descendant* RR" are inexpressible with AQT. Nevertheless, such queries are not meaningful for AML because conditional $\mathrm{CAEXBasicObjects}$ are semantically bound to their direct parent in the AML document. Similarly, the rest of the configuration parameters are not relevant for conditional $\mathrm{CAEXBasicObjects}$. Therefore, the applicability of AQT is not severely restricted.

The formal representation M is called an *AQT tree* and each node in V is an *AQT node*. For each AQT supplied by the user, an AQT tree will be constructed, which is the input for the semantic interpretation. For the XML elements `IH`, `IE1`, `IE2`, and `RR` in Fig. 7.5, Formula 7.1 shows their corresponding AQT nodes (each v_i is only an auxiliary name for a node). The order of the configuration parameters corresponds to Table 7.2.

$$
\begin{aligned}
v_1 &= (\ id_1, \mathsf{IH}, \{\texttt{Name} = \text{“}\texttt{AQT}\text{”}\}, \emptyset\ )\\
v_2 &= (\ id_2, \mathsf{IE}, \{\texttt{Name} = \text{“}\texttt{IE1}\text{”}, \texttt{ID} = \text{“}\texttt{fec46504...}\text{”}\},\\
&\qquad \{true, false, false, false, 1, -1\}\ )\\
v_3 &= (\ id_3, \mathsf{IE}, \{\texttt{Name} = \text{“}\texttt{IE2}\text{”}, \texttt{ID} = \text{“}\texttt{3fb9fc26...}\text{”}\},\\
&\qquad \{false, false, false, false, 1, -1\}\ )\\
v_4 &= (\ id_4, \mathsf{RR}, \{\texttt{RefBaseRoleClassPath} = \text{“}\texttt{Structure}\text{”}\}, \emptyset\ )
\end{aligned}
\tag{7.1}
$$

Recall the structure and property features in Table 7.1. Since AQTs are essentially AML models, the features (S1-S2) and (P2-P7) are inherently supported, while (S3) and (P1) are covered by the parameters `identifiedById` and `identifiedByName`, respectively. Finally, the "Features" column in Table 7.2 clarifies that AQT also supports the XML query features (X1-X4) through the corresponding configuration parameters.

7.2.2. Semantics of AQT

Similar to XQBE [32], the semantics of AQT is defined based on the notion of TPQs (cf. Section 6.2).

Let $M = (V, F)$ be an AQT tree, as described in Section 7.2.1. It is always possible to construct a TPQ $Q = (N, E)$ from M. Besides the AttributeValueRequirement (AVR) nodes (see Section 2.1), each node $v \in V$ is mapped to a node $n \in N$ as follows (the dot operator denotes the membership of variables):

- $n.id = v.id$, $n.tag = v.tag$, $n.r = v.B.returned$.
- The set $n.C$ contains all constraints in $v.A$, except for `ID` and `Name`. The latter two are determined by the configuration parameters $v.B.identifiedById$ and $v.B.identifiedByName$, respectively.
- The set $n.D$ can either be empty or have two integer values $\{min, max\}$, which correspond to the configuration parameters $minCardinality$ and $maxCardinality$ in $v.B$, respectively.

Besides the edges between the Attr nodes and their AVR children, each edge $f = (v_i, v_j)$ in F is mapped to an edge $e = (n_i, n_j)$ in E, where v_i is mapped to n_i and v_j is mapped to n_j. If $v_j.B$ is empty or the value of $v_j.B.descendant$ is false, then $e.type = pc$. Otherwise, $e.type = ad$.

Consider the AQT node v_2 in Formula 7.1. Because both parameters `identifiedById` and `identifiedByName` of $v2$ are set to $false$, the following TPQ node is generated based on the above mapping:

$$(id_2, \mathsf{IE}, true, \emptyset, \{1, -1\}) \tag{7.2}$$

An AVR node with its descendant nodes in an AQT tree defines value constraints for the superordinate Attr node. Fig. 7.6 shows an excerpt of an AQT with the AVR node named $xRequiredValue$. The mapping from the entire CAEX attribute to TPQ starts with The following TPQ node of `x`:

$$\begin{aligned} n_x = \; (\; & id_x, \mathsf{Attr}, false, \{\texttt{Name} = \text{“}\texttt{x}\text{”}, \texttt{Unit} = \text{“}\texttt{mm}\text{”}, \\ & \texttt{AttributeDataType} = \text{“}\texttt{xs : double}\text{”}\}, \{1, -1\} \;) \end{aligned} \tag{7.3}$$

To compile the value constraint into the TPQ, a new TPQ node n_{xv} is generated from the AVR node as follows:

$$n_{xv} = \; (id_{xv}, \texttt{Value}, false, \{\texttt{value} \geq -5\}, \emptyset) \tag{7.4}$$

```
<Attribute Name="x" Unit="mm" AttributeDataType="xs:double">
    <Constraint Name="xRequiredValue">
        <OrdinalScaledType>
            <RequiredMinValue>-5</RequiredMinValue>
        </OrdinalScaledType>
    </Constraint>
    <Attribute Name="queryConfig">
        <Attribute Name="identifiedByName" AttributeDataType="xs:boolean">
            <Value>True</Value>
        </Attribute>
        <!--other parameters take default values-->
    </Attribute>
</Attribute>
```

Figure 7.6.: An excerpt of an AQT that contains an AVR node.

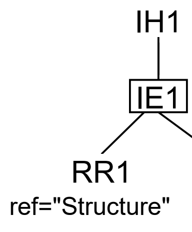
Note that the two “values” in Formula 7.4 are semantically different: the XML tag “`Value`” of the TPQ node n_{xv} is defined by the CAEX schema for storing the value of CAEX attributes (see the `Value` element of `Frame.y` in Fig. 7.3a), while “`value` ≥ -5” is a query constraint. Finally, the pc-edge (n_x, n_{xv}, pc) between n_x and n_{xv} is generated to represent their parent-child relation. Informally, this mapping applies a value constraint to the XML element `Value` of the embeddings of the CAEX attribute `x`.

There are two advantages of interpreting AQTs as TPQs. First, it fosters the automated translation to XPath programs, since TPQ can be seen as an abstraction of XPath [86]. Second, it enables a tree-like visualization of AQTs, as shown in Fig. 7.7. However, this visualization is neither used for query construction nor query translation. Instead, it is used to demonstrate the conceptualization of a query because it is more compact than the corresponding XML serializations (compare q1 with Fig. 7.5). For creating queries, the user only needs to construct the AQTs in the AML editor. In Fig. 7.7, each tree structure corresponds to one AQT that describes a distinct query for AML. For example, q1 represents the AQT in Fig. 7.5.

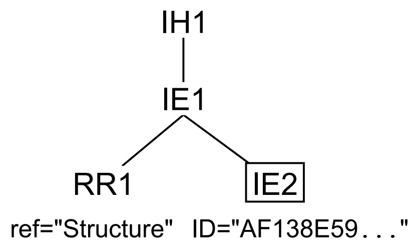
As AQT is used to query engineering data stored in AML instance hierarchies, the root of the tree is always an IH node. In the literature, TPQ nodes are usually represented by their XML tags [86][99]. However, Fig. 7.7 shows the ID of each node, e.g., `IH1` and `IE1`. The reason for this adaptation is that AML allows recursively nested XML elements (e.g., the nested IEs), which make the query examples hard to explain. A single line between two TPQ nodes depicts a pc-edge, and a double line represents an ad-edge. A *returned* node is marked with a box. Constraints in the set C of each TPQ node are shown as labels around the node. The term “`ref`” is used as a general name for all kinds of references in AML.

Consider the source AML file as shown in Fig. 7.3. The query results are illustrated by the numbers of the XML elements in the source AML file. For all the queries, the XML query feature X1 is essential because it allows specifying any inner tree node as the answer to the query.

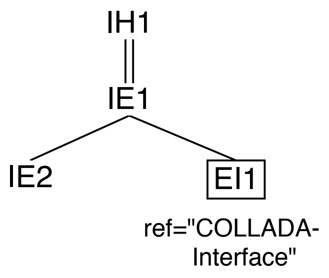
Query q1 selects the IEs that refer to the class `Structure` and have at least one child IE. It returns {1} by matching `IE1` to the element 1 and `IE2` to any of the elements 2, 6, 10, 13.



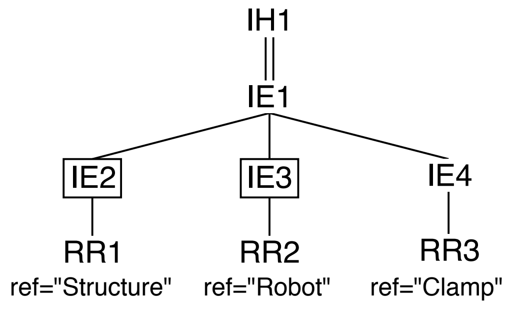
(a) (q1):{1}



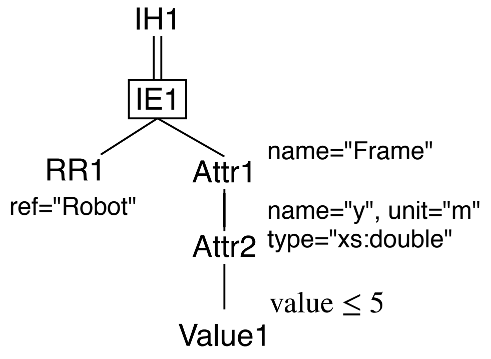
(b) (q2):{13}



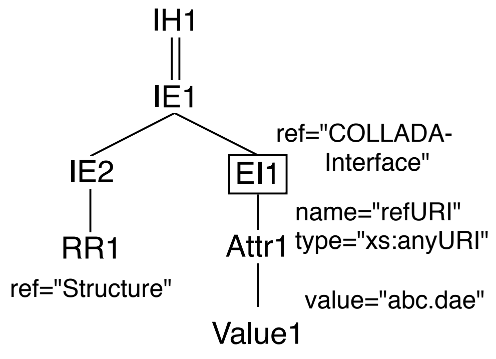
(c) (q3):{14}



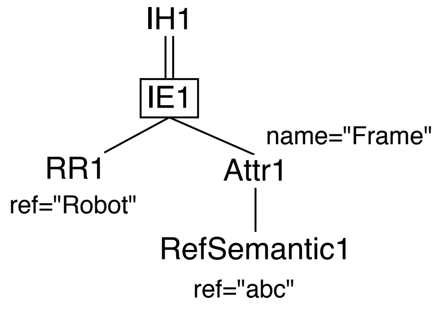
(d) (q4):{{5,3},{9,7}}



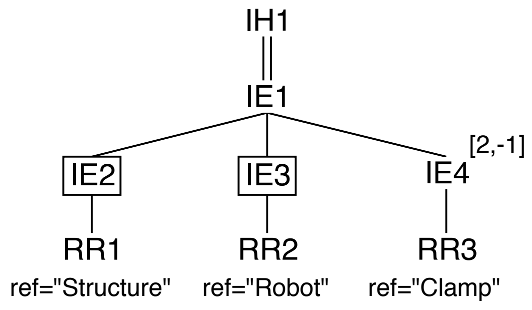
(e) (q5):{7,11}



(f) (q6):{14}



(g) (q7):{}



(h) (q8):{}

Figure 7.7.: Visualization of the TPQ structures of AQTs. Numbers in the curly brackets are the query results w.r.t. the AML instance hierarchy in Fig. 7.3.

Query q2 navigates to IE2 through IE1 that refers to the class Structure and selects IE2 if the UUID can be matched. It returns {13} by matching IE1 to the element 1.

Query q3 reads "from the descendant IE1 with at least one child IE2, select all COLLADA interfaces (EI1)". It returns {14} by matching IE1 to the element 13 and IE2 to the element 15 or 16.

Query q4 makes use of the XML query feature X3 for specifying two returned nodes. It states "from the descendant IE1 that has at least one child IE4 of the class Clamp, select all IE children that either refer to the class Structure or Robot". The result contains the set {5,3} by matching IE1 to the element 2 and the set {9,7} by matching IE1 to the element 6. Note that the coexistence of all three IE children is necessary.

Query q5 attaches value constraints to the Attrs. It means "select the descendant IE1 that refers to the class Robot and has a nested CAEX attribute Frame.y $\leq$ 5". The result of q5 contains {7,11} because the value constraints can be satisfied, as shown by the dashed boxes in Fig. 7.3b. This query employs the XML query feature X2 to select the descendant node IE1.

Query q6 extends q3 and reads "from the descendant IE1 with at least one child IE2 of the class Structure, select all COLLADA interfaces whose refURI attribute has the value $abc.dae$". The query returns {14} by matching IE1 and IE2 to the elements 13 and 16, respectively.

Query q7 enforces the node Attr1 to have a semantic reference to some path $"abc"$, which has no evidence in the source data. Therefore, q7 returns an empty set. Similar to q5, this query employs the XML query feature X2 to select the descendant node IE1.

Query q8 demonstrates the effect of cardinality restrictions (XML query feature X4), which are illustrated as a pair of numbers in square brackets. In contrast to q3, at least two matches are required for the class Clamp. The result of q8 is empty since no IE in Fig. 7.3 contains two distinct IE children of the class Clamp.

7.3. Translating AQTs to XPath and XQuery

Because AQTs are interpreted as TPQs, they can be translated into XPath and XQuery programs. This section first handles AQTs with a single returned node, for which XPath is sufficient to address the portion of the AML document that is described with the query. Then, this section continues with the translation of AQTs with several returned nodes, which requires the expressiveness of XQuery.

7.3.1. Translating AQTs with single returned node

Given an AQT tree $M = (V, F)$ with a single returned node, a TPQ $Q = (N, E)$ is first constructed according to the mapping in Section 7.2.2. Then Q is translated to XPath with Algorithm 7.1. The input of the algorithm contains the node $n \in N$ that is currently being visited and the XPath fragment e that is built so far. Initially, the parameter n is set to the root node, and e is empty.

Algorithm 7.1 TranslateToXPath

```
Input: Current node $n$, XPath fragment $e$
Output: XPath expression of the query
 1: Let $step$ be an empty XPath expression
 2: if ($n$ is the returned node, i.e., $n.r == true$) then
 3:     $step$ = GetLocationStepRec($n$)
 4:     append $step$ to $e$
 5: else
 6:     $step$ = GetLocationStep($n$)
 7:     Let $cc$ be the critical child of $n$
 8:     Let $ncc$ be all non-critical children of $n$
 9:     for $child \in ncc$ do
10:         add GetLocationStepRec($child$) to the predicate of $step$ conjunctively
11:     end for
12:     append $step$ to $e$
13:     TranslateToXPath($cc, e$)
14: end if
15: return $e$
```

The basic idea of the algorithm is to identify the so-called *critical nodes* in N. A critical node is either the returned node or one of its ancestors. Intuitively, the root node is always critical, and each inner-node has at most one critical child since only one returned node is allowed. Finally, non-critical nodes do not have any critical descendants.

The inner function GetLocationStep(n) generates a one-step XPath location step for the current TPQ node n. First, the axis is generated. If n is the root node or the type of the edge that connects n with its parent is pc, then the axis is $/$. Otherwise, the axis is $//$. Second, the XML tag is generated from $n.tag$. Finally, all value constraints in the set $n.C$ are translated into a conjunctive formula within the predicate of the location step. For example, the result of GetLocationStep(`EI1`) on the `EI1` node in q6 (Fig. 7.7) is:

$$/\texttt{EI}[\texttt{@ref=“COLLADAInterface”}] \tag{7.5}$$

The inner function GetLocationStepRec(n) computes a nested XPath location step for a node n by recursively translating its non-critical descendants into its predicate. Note that because non-critical nodes do not have any critical descendants, GetLocationStepRec(n) will traverse the entire sub-tree of n if n is non-critical. For the `EI1` node in q6, the function call GetLocationStepRec(`EI1`) extends Formula 7.5 by adding the nested location steps for `Attr1` and `Value1` as:

$$\begin{aligned}&/\texttt{EI}[\texttt{@ref=“COLLADAInterface”}\ \underline{\texttt{and}}\\ &\quad \texttt{child::Attr}[\texttt{@Name=“refURI”}\ \underline{\texttt{and}}\\ &\qquad \texttt{@AttributeDataType=“xs:anyURI”}\ \underline{\texttt{and}}\\ &\qquad \texttt{child::Value=“abc.dae”}]]\end{aligned} \tag{7.6}$$

If cardinality restrictions with non-default values present in the set $n.D$, then the XPath built-in function `count()` is used. For example, the node `IE4` in query q8 produces:

$$\texttt{count}(\texttt{child::IE}[\texttt{child::RR}[\texttt{@ref=“Clamp”}]]) \geq \texttt{2} \tag{7.7}$$

For the query q6 in Fig. 7.7, the translation starts with the initial values n=`IH1` and e="". Because `IH1` is not returned, GetLocationStep(`IH1`) returns "`/IH`" in line 6. `IH1` has only one critical child `IE1`. Therefore, e is extended to "`/IH`" in line 11 and TranslateToXPath(`IE1`, "`/IH`") is called in line 12. In the second iteration, $step$="`//IE`" is computed for the current node `IE1` in line 6. Because `IE1` has a non-critical child `IE2`, line 10 adds GetLocationStepRec(`IE2`) to the predicate of $step$ as:

```
//IE[child::IE[child::RR[@ref="Structure"]]]        (7.8)
```

In line 11, e="`/IH`" is extended with Formula 7.8 as:

```
/IH//IE[child::IE[child::RR[@ref="Structure"]]]        (7.9)
```

In line 12, the returned node `EI1` becomes the first parameter of the next recursive call. In the third iteration, Formula 7.6 is generated in line 3 and appended to the current XPath fragment (Formula 7.9) in line 4. Finally, line 13 returns the following XPath program as the result of the translation:

```
/IH//IE[child::IE[child::RR[@ref="Structure"]]]/
 EI[@ref="COLLADAInterface" and
  child::Attr[@Name="refURI" and                      (7.10)
      @AttributeDataType="xs:anyType" and
      child::Value="abc.dae"]]
```

7.3.2. Translating AQTs with multiple returned nodes

The translation of an AQT with multiple returned nodes needs to preserve the structural dependencies between the TPQ nodes to avoid redundant computation. Therefore, it is necessary to traverse the TPQ based on the hierarchical order of the lowest common ancestors (LCA) of each pair of the returned nodes. For each LCA, an XQuery variable is created and reused during the traversal. Syntactically, the translation always produces a nested XQuery for-return-clause.

For example, query q4 in Fig. 7.7 has two returned nodes `IE2` and `IE3`, whose LCA is the node `IE1`. Hence, an XQuery variable `$n0` is generated for `IE1` using the following XQuery for-clause:

```
for $n0 in /IH//IE[child::IE[
                                                      (7.11)
          child::RR[@ref="Clamp"]]]
```

Recall that a critical node is a returned node or an ancestor of a returned node. The location path after the keyword "`in`" in Formula 7.11 is generated by appending the result of GetLocationStepRec(`IE1`) to GetLocationStep(`IH1`) since `IE1` has a non-critical child `IE4` and `IH1` has no non-critical child. The translation proceeds with the critical children of the LCA (`IE1`), which are the returned nodes `IE2` and `IE3`. To enforce their coexistence, the variables `$n1` and `$n2` are created for `IE2` and `IE3`, respectively. Then, a combined XQuery for-clause is generated by appending $\mathrm{GetLocationStepRec}(\texttt{IE2})$ and $\mathrm{GetLocationStepRec}(\texttt{IE3})$ to the variable `$n0` as:

$$\begin{aligned}\underline{\texttt{for}}\ \texttt{\$n1}\ \underline{\texttt{in}}\ \texttt{\$n0/IE[child::RR[@ref=“Structure”]]},\\ \texttt{\$n2}\ \underline{\texttt{in}}\ \texttt{\$n0/IE[child::RR[@ref=“Robot”]]}\end{aligned} \tag{7.12}$$

The final XQuery program of q4 is constructed by nesting Formula 7.12 into Formula 7.11, and adding a return-clause for the variables `$n1` and `$n2`:

$$\begin{aligned}&\texttt{for \$n0 in /IH//IE[child::IE[child::RR[@ref=“Clamp”]]]}\\ &\quad\texttt{for \$n1 in \$n0/IE[child::RR[@ref=“Structure”]],}\\ &\qquad\texttt{\$n2 in \$n0/IE[child::RR[@ref=“Robot”]]}\\ &\quad\texttt{return(\$n1, \$n2)}\end{aligned} \tag{7.13}$$

7.3.3. Covering Advanced AML Modeling Features

The AutomationML advanced features (A1-A3) in Table 7.1 are not immediately covered by the translation above. Instead, they are handled as a global setting that can be switched on and off by the user.

For example, the user might want to consider subclasses in a query. To this end, dedicated XQuery functions are provided for traversing the inheritance hierarchy of AML classes (A1). Consider the query q8 in Fig. 7.7. If the node `IE4` with the cardinality restriction $[2, -1]$ were referring to the class `Resource`, then q8 would return {{5,3},{9,7}} as its result since the cardinality restriction is satisfied by the elements {3,4} and {7,8} through the subclasses `Robot` and `Clamp` of `Resource` (see Fig. 2.4a).

Internal links are used to model connections between EIs. One internal link has two endpoints A and B that contain the UUID of the connected interfaces. Recall that the AQT-based query pipeline requires two AML files, one that contains the source AML data (the source file) and another one that contains the AQTs (the query file). Depends on where the UUIDs can be found, the following three cases are distinguished when an internal link appears in the query file:

(a) If neither A nor B can be found in the query file, then a modeling error is discovered, and the internal link is ignored.

(b) If A is found in the query file, and B is found in the source file, then the embeddings of A shall be connected to the concrete source interface B (A2 in Table 7.1).

(c) If both A and B are found in the query file, then their embeddings shall be connected to each other (A3 in Table 7.1).

Fig. 7.8 shows a query file that has two AQTs and two links. The AQT with the root node `IH1` contains a returned node and needs to be translated. Let `EI1.ID`, `EI2.ID`, and `EI4.ID` be the UUID of `EI1`, `EI2`, and `EI4`, respectively. Then the following two internal links are found for the query:

link1:	(`EI1.ID`, “`72B7...`”)
link2:	(`EI2.ID`, `EI4.ID`)

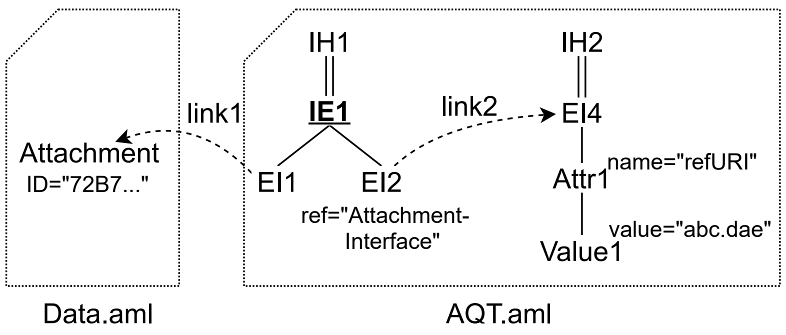


Figure 7.8.: A query file that has two AQTs and two internal links.

Link1 corresponds to case (b) where the partner interface with the UUID "7287..." is found in the source file. Link2 corresponds to case (c) where the partner interface `EI4` is found in the query file. For encoding the link constraints in both cases, the XQuery function `connectsTo($context, $partner)` is provided. The first parameter stands for the context EI, i.e., `EI1` for link1 and `EI2` for link 2. The second parameter is the XPath expression of the partner EI, which is either determined by its UUID (link 1) or its complete XPath location step (link 2). Given a query file with internal links, the AML Query Processor (AQP) will determine to which case each internal link belongs to and generates the corresponding XQuery code for it. Formula 7.14 and 7.15 show the XPath conditions that are built for links 1 and 2, respectively. In both cases, the context EI is represented using a dot "." and the XQuery variable `$root` stands for the root of the source file.

$$\texttt{connectsTo}(., \texttt{\$root//EI}[\texttt{@ID} = \text{“abc”}]) \tag{7.14}$$

$$\begin{aligned}&\texttt{connectsTo}(., \texttt{\$root//EI}[\texttt{Attr}[\texttt{@Name} = \text{“}\texttt{refURI}\text{”}\\ &\qquad\qquad \underline{\texttt{and}}\ \texttt{Value} = \text{“abc.ade”}]])\end{aligned} \tag{7.15}$$

For the query in Fig. 7.8, Formula 7.16 shows the result of the query translation that embeds Formula 7.14 and 7.15 into the XPath expression of node `IE1`.

$$\begin{aligned}&\texttt{IH//IE}[\texttt{EI}[\texttt{connectsTo}(., \texttt{\$root//EI}[\texttt{@ID} = \text{“abc”}])]\\ &\qquad \underline{\texttt{and}}\ \texttt{EI}[\texttt{connectsTo}(., \texttt{\$root//EI}[\texttt{Attr}[\texttt{@Name} = \text{“}\texttt{refURI}\text{”}\\ &\qquad\qquad \underline{\texttt{and}}\ \texttt{Value} = \text{“abc.ade”}]])]]\end{aligned} \tag{7.16}$$

7.4. Results and Discussions

This thesis provides an implementation of AQT and AQP in a Java framework [4]. While every standard-conform XQuery processor can be employed for query execution, the open-source library BaseX[5] is used, which is available on all mainstream operating systems and has an easy-to-use Java interface. The AutomationML advanced features (A1-A3) are implemented in a stand-alone XQuery function module and can be switched on and off by demand.

Fig. 7.9 shows the process of query translation and execution. The user can inspect the AutomationML files and the generated XPath/XQuery programs while the AQT tree and

[4]https://github.com/kit-hua/aml
[5]http://basex.org/

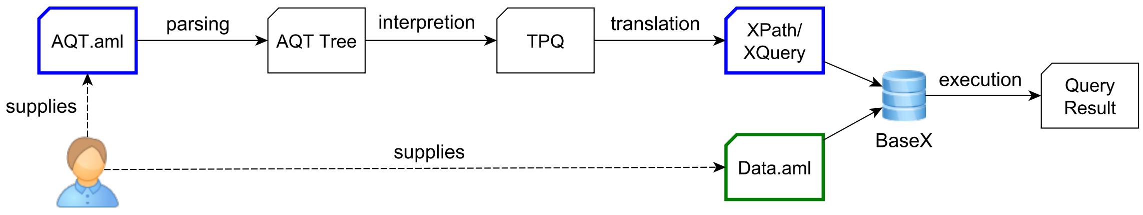
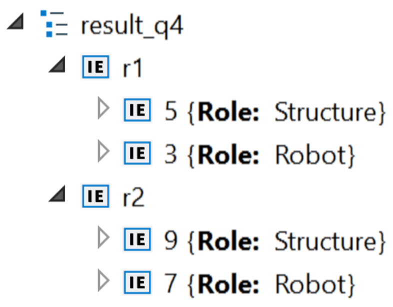


Figure 7.9.: An overview of query translation and execution.



```
◢ result_q4
  ◢ IE r1
      ▷ IE 5 {Role: Structure}
      ▷ IE 3 {Role: Robot}
  ◢ IE r2
      ▷ IE 9 {Role: Structure}
      ▷ IE 7 {Role: Robot}
```

Figure 7.10.: The query results of q4 (cf. Fig. 7.7d) regarding the source AML document as shown in Fig. 7.3b.

the TPQ only exist in the memory of the computer. It can be observed that the AQTs (models in AQT.aml) are simply the AutomationML representation of the TPQ and the desired query. After query execution, the user can either take the raw query outputs (i.e., XML text) from the console or request an AutomationML document that stores the query results in instance hierarchies. In the latter case, one AutomationML instance hierarchy is generated for each query, and auxiliary AutomationML objects are created for grouping query results if necessary. Fig. 7.10 illustrates the output AutomationML document for q4 in Fig. 7.7d. The nodes {5, 3} and {9, 7} are grouped by the auxiliary objects `r1` and `r2` respectively.

Table 7.3 shows the comparison of the expressive power between AQT and existing QBE approaches as described in Section 6.4. On the one hand, XQBE [32] and similar general-purposed approaches employ dedicated visual languages for query construction. Because the AML structure and property features are essentially tree structures of XML elements and attributes, (S1-S3) and (P1-P7) can be expressed with these visual languages. Moreover, these approaches allow the modeling of data exchange rules for the assembly of a target XML document, such as projection, join, and renaming. These features are not covered by AQT because they require the modeling of explicit value mapping and comparison (E1 in Table 7.3). For example, it is non-trivial to model `IE1.manufacturer` = `IE2.vendor` in AML, where `manufacturer` and `vendor` are Attrs, and `IE1` and `IE2` are arbitrary IEs. Value mapping is fundamental for data exchange, where the value of `IE1.manufacturer` needs to be copied to `IE2.vendor`.

On the other hand, AQL [85] adopts a similar idea of using AML models to query AML data. Nevertheless, AQL has a syntax defined within EMF and an informal notion of semantics. In terms of expressive power, the AML structure and property features (S1-S3) and (P1-P7) are supported since the meta-model of AQL is built on top of the CAEX schema. Furthermore, AQL has a set of boolean flags that support the XML data query features. For example, the *negated* and *multi* flag correspond to the cardinality restrictions (X4), the *returned* flag can be used for specifying the position of the returned data objects (X1, X3), and the *transitive* flag stands for the transitive closure of the XML

Table 7.3.: Comparison of the expressive power between AQT and the related work. It can be observed that AQT supports three AutomationML advanced features which are not considered in the related work. Also note that AQL has only proposed a modeling approach that was neither algorithmically investigated nor concretely implemented.

Modeling Features	**XQBE** [32]	**AQL** [85]	**AQT**
AutomationML Structure Features			
(S1) structurally nested objects and interfaces	yes	yes	yes
(S2) class-instance relationship	yes	yes	yes
(S3) object identification via UUID	yes	yes	yes
AutomationML Property Features			
(P1) attribute name	yes	yes	yes
(P2) structurally nested attributes	yes	yes	yes
(P3) explicit (default) attribute value	yes	yes	yes
(P4) attribute unit	yes	yes	yes
(P5) attribute data type	yes	yes	yes
(P6) attribute semantic reference	yes	yes	yes
(P7) attribute value requirements (nominal and ordinal)	yes	yes	yes
AutomationML Advanced Features			
(A1) class inheritance hierarchy via class-class relations	no	no	yes
(A2) instance-instance relations (partner in the source file)	no	no	yes
(A3) instance-instance relations (partner in the query file)	no	no	yes
XML Data Query Features			
(X1) retrieving objects at any position in the XML structure	yes	yes	yes
(X2) retrieving descendant objects in the XML structure	yes	yes	yes
(X3) querying related data simultaneously (multi-return)	yes	yes	yes
(X4) cardinality restrictions	yes	yes	yes
XML Data Exchange Features			
(E1) explicit value mapping and comparison	yes	no	no

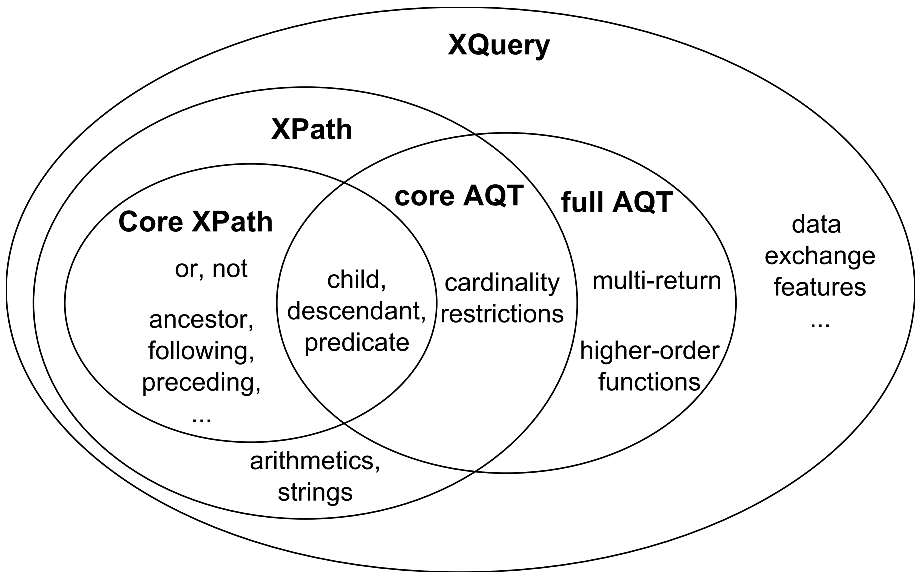


Figure 7.11.: Expressive power of AQT w.r.t. XPath and XQuery.

containment relation which is equivalent to the descendant axis in TPQ (X2). Nevertheless, AQL neither covers the AutomationML advanced features (A1-A3) nor supports the value mapping and comparison (E1).

In conclusion, AQT is designed to be intuitive for the AML users by exploiting the modeling features of AML itself. The expressiveness of AQT is sufficient to fulfill the requirements for querying AML data (cf. Section 7.1). Furthermore, the semantics of AQT is defined based on the notion of tree pattern queries, which are well-established in the field of XML and are closely related to XPath. Therefore, AQT can be automatically translated to XPath or XQuery programs.

Fig. 7.11 shows an overview of the expressive power of AQT with respect to XPath and XQuery. The subset called *core AQT* refers to queries with only one returned node and no AML advanced features. Because the semantics of AQT is defined based on the notion of TPQs, core AQTs also capture the XPath fragment $XP^{/,//,[]}$ (cf. Section 6.1.1), while additionally supports cardinality restrictions. Note that arithmetic operations, e.g. $+, -, *, /,$ `mod`, and string operations, e.g. `contains`, are neither supported by TPQ nor AQT. However, these features do not belong to the logical core of queries and can be considered as future work in the standardization of CAEX and AML.

In addition to core AQT, *full AQT* allows multiple returned nodes and supports AML advanced features. The latter one requires the definition of higher-order XQuery functions for traversing the inheritance hierarchy of AML class libraries, as shown in Listing 6.1. However, as illustrated in Table 7.3, AQT does not support the modeling of value mapping and comparison, which is a basis for more advanced data exchange features. In the next chapter, an extension of AQT is proposed to cover value mappings and enable AML data exchange.

8. By-example Data Exchange for AutomationML

This chapter extends AQT for exchanging AML data between engineering tools. Again, the design principle of the extension is to use AML itself for query modeling and to use XQuery for query translation. However, while the semantics of data access can be naturally defined based on the notion of tree pattern queries (cf. Section 7.2.2), a theoretical foundation for AML data exchange is yet to be defined. To this end, Section 8.1 first conducts a comprehensive analysis of functional requirements for AML data exchange, with a focus on industrial use cases. Afterwards, Section 8.2 develops a data exchange setting for AML, based on which the syntax and the semantics of the extended AQT are defined in Section 8.3. Finally, Section 8.4 presents algorithms for translating extended AQTs into XQuery programs.

8.1. Requirements for AML Data Exchange

To define the expressiveness of the extended AQT, this section reviews literature in three relevant yet independently developed research domains, including: (i) common industrial data exchange scenarios; (ii) semantic correspondence patterns; (iii) general XML data exchange use cases.

8.1.1. Common Industrial Data Exchange Scenarios

Drath discussed three kinds of typical data exchange scenarios in industrial engineering [100], including object mapping, structure mapping, and attribute mapping.

8.1.1.1. Object Mapping Scenarios

Object mapping (**OM**) refers to data transfer on the engineering object level, as shown in Fig. 8.1:

(OM1) 1-1 mapping: one source object A_1 is mapped to one target object B_1. Because this thesis does not consider value changes during the data exchange process, 1-1 mapping is simply a copy event that takes an object from the source to the target.

(OM2) 1-n mapping (decomposition): one source object A_1 is mapped (decomposed) to two target objects B_1 and B_2. In case that A_1 contains some attributes, then the 1-n mapping may involve several attribute mappings that specify which attributes of A_1 shall be taken to B_1 and B_2, respectively.

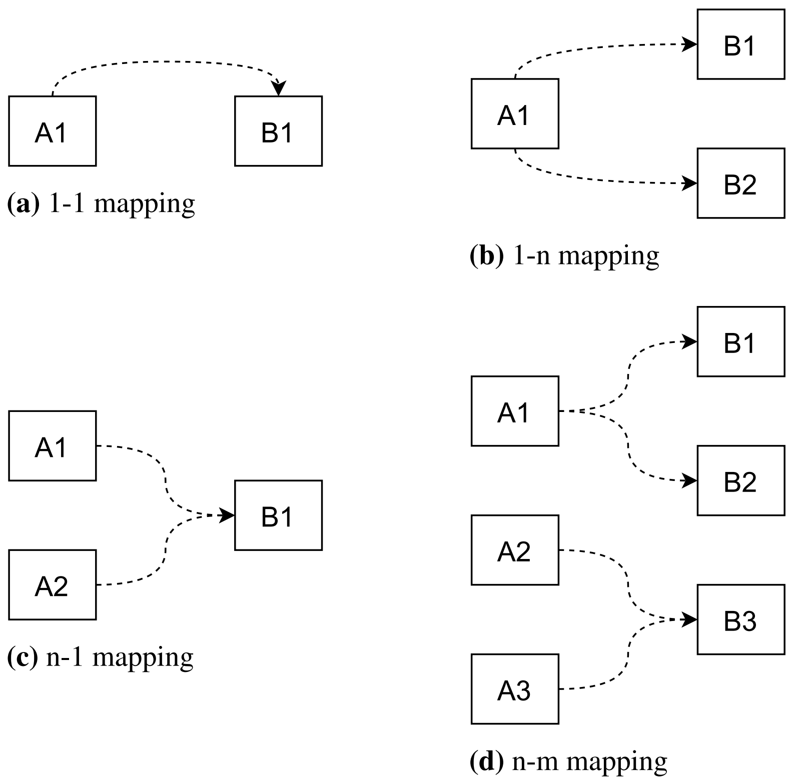


Figure 8.1.: Object mapping scenarios in industrial data exchange.

(OM3) n-1 mapping (aggregation): two source objects A_1 and A_2 are mapped (aggregated) to one target object B_1.

(OM4) n-m mapping: the source object A_1 is mapped to two target objects B_1 and B_2 (1-n mapping), and the source objects A_2 and A_3 are mapped to one target object B_3 (n-1 mapping). The complexity of n-m mapping is a mixture of the underlying n-1 and 1-n mappings.

8.1.1.2. Structure Mapping Scenarios

Object mappings described in Section 8.1.1.1 do not take into account structural dependencies between the objects. In the XML/AML context, data is organized as trees (cf. Section 6.1), where each object represents a tree node. Fig. 8.2 illustrates use cases in which the structural mappings (**SM**) of the objects need to be considered:

(SM1) Breadth projection[1]: the source object A_1 is mapped to the target object B_1 while the child object A_2 is ignored.

(SM2) Child-only breadth projection[2]: the source object A_1 itself is ignored in the data exchange. However, its child object A_2 is mapped to the target object B_1.

(SM3) Flattening[3]: the source object A_1 is mapped to the target object B_1, and the child object A_2 is mapped to the target object B_2. Note that the parent-child relation between A_1 and A_2 is resolved on the target side.

[1]It was called *BreakIncludingMe* in [100].
[2]It was called *Ignore* in [100].
[3]It was called *IgnoreHierarchy* in [100].

(SM4) Flattening and decomposition: a mixture of (SM3) and (OM2).

(SM5) Offset path: the source object A_1 and its child A_2 are mapped to the target objects B_1 and B_3, respectively. The parent-child relation between A_1 and A_2 is changed to a predecessor-descendant relation in the target tool because an intermediate new object B_2 is required in the target.

8.1.1.3. Attribute Mapping Scenarios

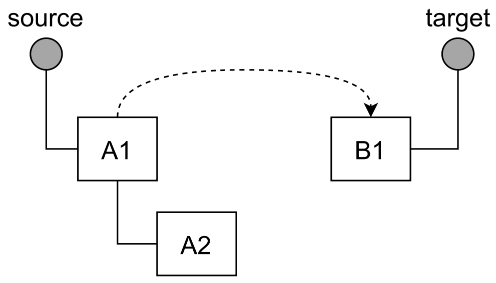
Attribute mapping refers to relations between properties defined in the source and target tool. For example, the source tool defines the `frame` attribute for the six-dimensional pose of an object, while the target tool uses the `coordinates` attribute for the same purpose. Because attributes are part of engineering objects, attribute mappings are always subject to object mappings, which in turn, can be associated with structure mappings. Drath described several common attribute mapping scenarios assuming that theses attributes are modeled as primitive data elements, for example, as XML attributes and thus shall be handled differently to the object mappings [100]. However, this thesis does not consider attribute mappings as a first-class mapping scenario for two reasons. First, as described in Section 1.1, this thesis does not cover value processing relations between engineering tools. Second, the externalization step in AML-based data exchange (cf. Fig. 2.5) transforms primitive data elements into complex XML elements, i.e., the CAEX attributes, which are handled similarly to other XML elements in an AML document, e.g., internal elements. Therefore, attribute mappings are treated as part of the object and structure mappings.

8.1.2. Semantic Correspondence Patterns

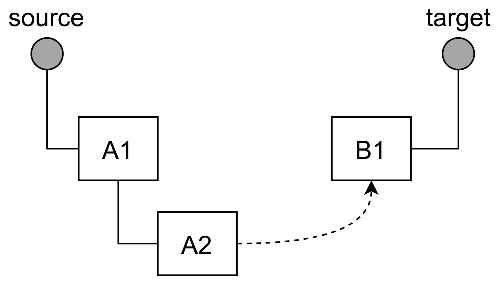
Semantic correspondence patterns were originally introduced for ontology alignment in Scharffe's Ph.D. thesis [101]. Kovalenko and Euzenat showed that some of these patterns represent typical *semantic matching* issues in multi-disciplinary engineering [34]. The term semantic matching (or ontology matching [102]) refers to finding relations between entities (i.e., objects, attributes, and classes) in different engineering systems. Although the original formulation of correspondence patterns was based on ontologies and thus semantically inconsistent with the XML setting, they provide guidelines for modeling complex relations between engineering systems and contribute to the design of query modeling. However, not all correspondence patterns are relevant for AML data exchange. For example, the *equivalent relation correspondence*, which states that one object relation is equivalent to another, is unnecessary, because both the source and the target tool use the same CAEX schema definition such that all object relations are predefined and homogeneous among the tools. In the following, the relevant correspondence patterns (**CP**) are summarized and adapted for AML data exchange:

(CP1) Equivalent class: a target AML class D is equivalent to a source AML class C, although they might be modeled differently.

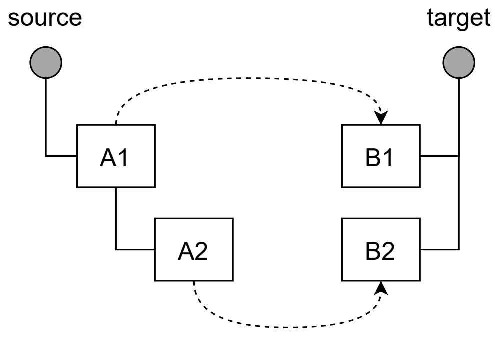
(CP2) Superclass: a target AML class D is a superclass of a source AML class C, i.e. $D \supset C$. Semantically, each internal element x that refers to C shall then be interpreted as an object of class D in the target tool, since $C(x) \rightarrow D(x)$.



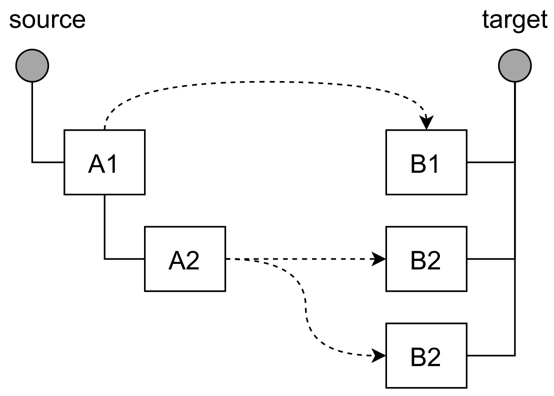
(a) Breadth projection



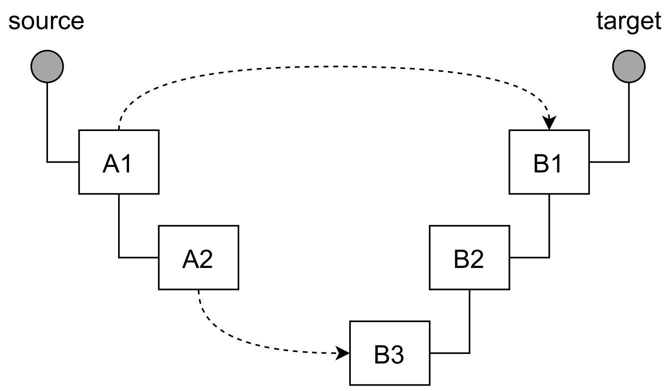
(b) Child-Only projection



(c) Flattening



(d) Flattening and decomposition



(e) Offset path

Figure 8.2.: Structure mapping scenarios in industrial data exchange.

(CP3) Subclass: a target AML class D is a subclass of a source AML class C, i.e. $D \subset C$. In this case, whether a source object of class C shall be imported to the target tool under the class D remains undefined and must be specified with additional constraints.

(CP4) Class by (path) attribute occurrence/value: a target AML class D corresponds to source objects which satisfy certain constraints on the associated (nested) attributes. In the context of AML, path attributes refer to nested CAEX attributes, e.g., the `x` coordinate of a `frame` attribute.

(CP5) Subclass defined by relation domain: in AML, the only essential relations are `hasIE` and `hasEI`. Therefore, this pattern means that a target AML class D corresponds to a set of source objects with certain sub internal elements or external interfaces.

(CP6) Equivalent property: in AML, properties are the CAEX attributes. Equivalent property means that a target CAEX attribute b has the same meaning as a source CAEX attribute a, and can copy the value from a. Note that the name of a and b can be different. For example, the source attribute `frame` is equivalent to the target attribute `coordinate`.

(CP7) Aggregation: several source objects shall be grouped together to formulate one target object. For example, the source internal elements ie_1, ie_2, and ie_3 shall be put under the target internal element ie_4 as child objects.

(CP8) Property-relation correspondence: a CAEX attribute in one tool might be represented as an object relation in another tool. For example, the CAEX attribute `manual` in the source tool stores the location of product manuals. The target tool might not have this attribute but an external interface of the class `Manual`, which holds a CAEX attribute `fileLocation`. During data exchange, the target tool needs to create an instance of the interface class `Manual`, and copy the value of the source attribute `manual` to the target attribute `fileLocation`.

Recall the data exchange workflow of AML, as illustrated by Fig. 2.5, and assume that the target tool defined some AML classes which do not exist in the source tool. CP1-CP5 suggests that the target-specific AML classes can be modeled as structure or value constraints of the source data. The design of a QBE-like approach for AML shall support these features to comply with the mixed model principle [4]. CP6 is the main functionality that is supported by early works on AML-based interoperability [1]. Finally, CP7 and CP8 represent two use cases for the construction of complex target instances.

8.1.3. General XML Data Exchange Use Cases

Section 8.1.1 and 8.1.2 summarized functional requirements for exchanging AML data which originated from industrial use cases. In terms of general XML data exchange, the following XQuery features (**XF**) [32] give a upper bound of the expressiveness that can be achieved by XQuery programs:

(XF1) Copy: the basic action in data exchange which copies data from source to target.

(XF2) Existential quantification: a source XML node shall contain a particular sub XML node or tree.

(XF3) Conjunction: a set of conjunctively connected logical expressions, i.e., several constraints that need to be satisfied together.

(XF4) Filtering: a constraint on the value of a source XML attribute or text node. Filtering can be combined with XF2 and XF3 for building complex filters over the source tree.

(XF5) Breadth projection: only a subset (can be empty) of sub-trees of a source node is taken into the target.

(XF6) Construction of new elements: the target side needs to construct some new XML elements which may interact with the data copied from the source side.

(XF7) Nesting: a set of source nodes are put in a particular nested structure in the target tree.

(XF8) Flattening: a set of nested source nodes become siblings in the target tree.

(XF9) Grouping: since XQuery 3.0, the `groupby` operator is available for grouping nodes on the target side. In practice, grouping can also be realized by restructuring the target XML document.

(XF10) Depth projection: suppose that a source XML tree node A has a child B, which in turn has a child C. Depth projection maps A and C to a target tree node A' and C' respectively, while ignoring the intermediate source node B. A special case of depth projection is that only C is copied into the target.

(XF11) Join: some XML attributes or text nodes of the source or target tree have the same value. Note that the actual value is not essential, as long as the equality holds.

(XF12) Aggregates: a target node aggregates data from several source nodes by means of the functions `min`, `max`, `count`, `avg`, `sum`. Note that aggregates in this context are different from the n-m mapping (Section 8.1.1) and the aggregation pattern (Section 8.1.2). The latter two refer to the "grouping" of several source nodes under one target node.

(XF13) Negation: represented by the XPath logical operator `not`. The expression $not(e)$ is true if e is false. Negations can be used to specify the opposite of a value constraint or the non-existence of a tree structure. For example, `Attr`$[$`not(Value` $>$ `0)`$]$ selects `Attrs` with `Value` $\leq$ `0`, and `IE`$[$`not(EI)`$]$ selects `IE`s which do not have any `EI`.

(XF14) Arithmetic computations: XQuery supports standard arithmetic computations such as $+, *, -, /$. As described in Section 7.4, AML per se does not support these features.

(XF15) Renaming: a source tree node A is copied to a target tree node A', and A' changes the original XML tag of A. This feature is irrelevant for AML since both the source and target AML file use the same XML schema definition.

(XF16) Cartesian product: let S_1 and S_2 be two sets of source tree nodes with $|S_1| = n$ and $|S_2| = m$. The Cartesian product $S_1 \times S_2$ produces $n \cdot m$ pairs, each of which contains one node from S_1 and one node from S_2. This feature is not considered in this thesis because the Cartesian product generates duplicated data on the target side.

Table 8.1.: Required XQuery features for the implementation of the common mapping scenarios described in Section 8.1.1. The features XF2-XF4 are implicitly used by all cases for describing complex structural and value constraints on the source nodes.

Common mapping scenarios	XQuery features
OM1 (1-1 object mapping)	XF1
OM2 (1-n object mapping)	XF1, XF5, XF6
OM3 (n-1 object mapping)	XF1 (for all qualified source objects)
OM4 (n-m object mapping)	XF1, XF5, XF6 (for all qualified source objects)
SM1 (breadth projection)	XF1, XF5
SM2 (child-only breadth projection)	XF1
SM3 (flattening)	XF1, XF8
SM4 (flattening and decomposition)	XF1, XF5, XF6, XF8
SM5 (offset path)	XF1, XF7

(XF17) Sorting: the XQuery operator `orderby` can be used to sort target nodes according to a value-based ordering. This feature is not essential for AML data exchange, because neither CAEX nor AML imposes an ordering over the objects in the instance hierarchy.

(XF18) Node order: since XML is an ordered tree in general terms, XQuery supports querying for a tree node in a particular position, e.g., $\texttt{IE}[\texttt{position}() = \texttt{1}]$ selects the first `IE` node. Similar to XF17, this feature is not considered in this thesis.

(XF19) Multiple documents: the target XML tree merges data from several sources. While this feature can be useful in general XML data exchange scenarios, it is not considered in this thesis, because the standard data exchange process as illustrated in Fig. 2.5 is based on one AML source file.

(XF20) Disjunction: represented by the XPath logical operator `or`. The expression `d or e` is true if either `d` or `e` is true. For example, $\texttt{Attr}[\texttt{Value} > \texttt{5} \texttt{ or } \texttt{Value} < \texttt{0}]$ selects `Attrs` whose `Value` is either larger than 5 or less than 0. As described in Section 7.4, AML per se does not support these features.

(XF21) Union: represents the logical operator $\cup$ and is denoted with the operator $|$ in XQuery. A union $A|B$ constructs a disjunctive combination of nodes that match A and B, respectively.

(XF22) Difference: for two sets of tree nodes S_1 and S_2, the difference between S_1 and S_2 (or the relative complement of S_1 w.r.t. S_2) are the nodes in S_1 but not in S_2.

(XF23) Universal quantification: a source XML node only contains sub-trees satisfying particular constraints.

It can be observed that features XF14 to XF23 are either inexpressible with AML (XF14, XF20-XF23) or unnecessary (XF15-XF19) for AML data exchange. Furthermore, by comparing the features above with the requirements described in Section 8.1.1, the features XF1-XF8 are sufficient for implementing the common industrial data exchange tasks, as shown in Table 8.1. In terms of semantic correspondence patterns, CP1-CP5 correspond to the features XF2-XF4 for modeling constraints in the source part. CP6 (equivalent property) is covered by XF1 and XF6 in the case of AML since all properties are represented as XML elements. Finally, CP7 corresponds to XF9 and CP8 can be realized with XF1-XF7.

In conclusion, XF1-XF9 are the essential features that must be supported by the extended AQT because they are necessary and sufficient to cover the requirements listed in Section 8.1.1 and 8.1.2.

8.2. A Data Exchange Setting for AML

The extension of AQT needs a theoretical foundation that defines the basic setting and the necessary notions for AML data exchange. Until now, two relevant formalisms are introduced, namely, the tree pattern query (TPQ) as described in Section 6.2 and the XML schema mapping as described in Section 6.3. Unfortunately, for the following reasons, neither formalism is appropriate for the data exchange task in AML-based engineering.

On the one hand, TPQ is a concept only for extracting data from a source XML file and has no means for specifying how to construct a target XML document. As mentioned in [95], a data exchange setting needs to collect data from the source and reproduce it in the target. Moreover, relationships between the source and target documents need to be defined. The requirement study presented in Section 8.1 suggested that two kinds of relationships are necessary for AML data exchange. The first one is the *full-copy* action that replicates a source data object in the target. The other one is the *partial-copy* action that projects parts of a source data object to the target.

On the other hand, XML schema mapping is primarily designed for transmitting data between XML databases, which conform to different XML schema definitions. Therefore, the XML data exchange setting, as shown in Definition 6.6, merely considers structural dependencies and value correspondences between the source and target side. The former is given by the structure of the tree-pattern formulae (TPF), while the latter is denoted by the identical variables. For carrying data from source to target, the TPFs in both sides must enumerate over all possible XML attributes and sub-trees that are defined. Suppose a simple STD $\varphi \rightarrow \psi$ that describes a copy of AML internal elements from source to target:

$$\texttt{IE}(\texttt{Name} = v1, \texttt{ID} = v2) \rightarrow \texttt{IE}(\texttt{Name} = v1, \texttt{ID} = v2)$$

According to Definition 6.5, for a pair of source and target internal elements s, t with $s.\texttt{Name} = t.\texttt{Name}$ and $s.\texttt{ID} = t.\texttt{ID}$, the above STD can be satisfied. However, the STD is not able to characterize the full-copy of an internal element, because the sub-trees of s and t are not specified. Suppose s has a supported role class, the STD that copies the supported role class to the target would be:

$$\texttt{IE}(\texttt{Name} = v1, \texttt{ID} = v2)[\texttt{SRC}(\texttt{ref} = v3)] \rightarrow \texttt{IE}(\texttt{Name} = v1, \texttt{ID} = v2)[\texttt{SRC}(\texttt{ref} = v3)]$$

Now, the target internal element t needs to have a corresponding supported role class for satisfying the STD. While this solution works for simple CAEX schema elements, it becomes subtle for complex, recursively defined ones. For example, AML internal elements can have nested internal elements, which in turn can have further nested internal elements. Therefore, a general STD that specifies the full-copy of any internal element would need infinitely many nested TPFs which can not be formulated.

Another challenge of the schema mapping approach is that common data exchange tasks in AML involve value comparisons. Recall the AQT examples in Fig. 7.7. It is evident that none of these tree patterns can be described with TPF. This problem has been reported under the term *contextual schema matching* in relational databases [37].

This thesis proposes a proper data exchange setting for AML by providing a notion for the target side, which adapts tree patterns in Definition 6.2. The basic idea of the setting is to have a source model Q and a target model P that describe patterns over the source and target AML documents, respectively. More specifically, Q is a *source tree pattern*[4] (STP) according to Definition 6.1, and P is a *target tree pattern* (TTP) defined as follows:

Definition 8.1 (Target Tree Pattern)**.** A target tree pattern is a node-labeled rooted tree $P = (N, E)$, where:

- N is the set of nodes, E is the set of edges, and the root of the tree is the node with no parent;
- Each node $x \in N$ can either be *bound* or *free*.
- A bound node t is represented as a tuple $(id, tag, name, sid, full)$, where id is the identifier of the node, tag is the related XML tag, $name$ is the desired name, sid is the identifier of a node s in an STP Q, and $full$ is a boolean flag. The node s is called the *source correspondence* of t. The relation between s and t is called a *binding*, which can either be a *full-copy* binding if $full = true$ or a *partial-copy* binding if $full = false$.
- A free node n is represented as a triple (id, tag, C), where id is the identifier of the node, tag is the related XML tag (the label), and C is a set of constraints on the value of n's text content or attributes, that is:

$$(n.\texttt{value} \mid n.\texttt{attr}) = c$$

  where `value` is the text content of n, `attr` is the name of an XML attribute, and c is a string constant.
- An edge in E is a pair (x, y) where x and y are identifiers of the connected nodes. In contrast to Definition 6.1, an edge in TTP always describes a parent-child relationship from x to y. In this context, TTPs are always *fully-specified* [95]. □

In Definition 8.1, a bound node indicates a full or partial copy from a source file, where sid specifies its source correspondence. The element $name$ is specifically designed for bindings between CAEX attributes, where the target engineering tool wants to import all content from a source CAEX attribute but change the attribute name. If the target node is not a CAEX attribute, then $name$ can be set arbitrarily. A free node is defined similarly to an STP node besides that only equality is allowed in the value constraints. Informally, free nodes are such data objects that only exist in the target side.

A pair (Q, P) with $Q = (N, E)$ being an STP and $P = (N', E')$ being a TTP can be visualized as two tree-like structures, between which binding edges exist, as shown in Fig. 8.3. The left part of the figure is the visualization of Q, as described in Section 7.2.2

[4]The terminology is only for convenience. A source tree pattern is indeed a tree pattern as described in Definition 6.1.

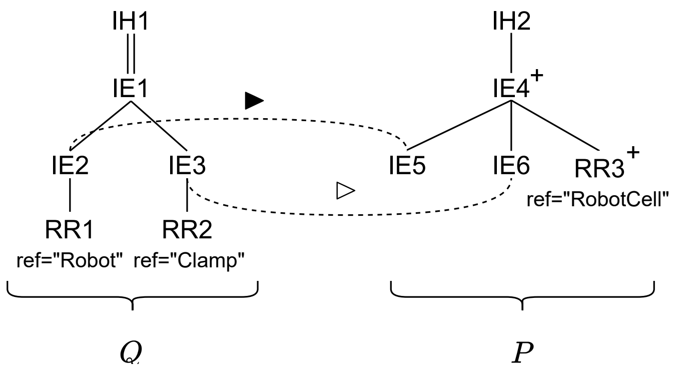


Figure 8.3.: The visualization of a pair (Q, P) where Q is an STP and P is a TTP.

(note that an STP has no returned node). The right part of the figure represents the structure of P where each node in N' is represented by its XML tag and ID, and each edge $e' = (n'_1, n'_2)$ is depicted with a line between the target nodes n'_1 and n'_2. A free target node is denoted by a $+$ symbol on the top right corner, e.g., the nodes `IE4` and `RR3`. Between a target bound node $n' \in N'$ and its source correspondence $n \in N$, there is a binding edge represented by a dashed line. Intuitively, $n.id = n'.sid$ is satisfied for a binding edge (n, n'). The symbol ▶ over a binding edge (n, n') denotes that (n, n') is a full-copy binding, i.e., $n'.full = true$, while $\rhd$ means that (n, n') is a partial-copy binding, i.e., $n'.full = false$. In Fig. 8.3, `IE5` and `IE6` are bound target nodes with `IE5`$.full = true$ and `IE6`$.full = false$.

Similar to a tree pattern, the semantics of a TTP is formally defined via *target embeddings*, as shown in Definition 8.2.

Definition 8.2 (Target Embedding)**.** Let $Q = (N, E)$ be a source tree pattern, $S = (V, F)$ be a source AML document, and $e \in Q(S)$ an embedding from Q to S, as described in Section 6.2. Let $P = (N', E')$ be a target tree pattern and $T = (V', F')$ be a target AML document. A target embedding from P to T that satisfies e is a function $embeddingT : N' \to V'$ such that:

1. for each free node $n' \in N'$, $n'.tag = embeddingT(n').tag$, and all value constraints in $n'.C$ are satisfied by $embeddingT(n')$.
2. for each bound node $t \in N'$, let $s \in N$ be the source correspondence of t:
   a) if $t.full = true$, then $embeddingT(t)$ is a full-copy of $e(s)$. Otherwise, $embeddingT(t)$ is a partial-copy of $e(s)$.
   b) if $t.tag =$ Attr, then $embeddingT(t)$.`Name` $= t.name$.
3. for each $(x, y) \in E'$, $(embeddingT(x), embeddingT(y)) \in F'$.

The XML node $embeddingT(n') \in V'$ is also called an instance of the TTP node n'. □

In Definition 8.2, case 1 describes the meaning of free nodes, which is similar to Definition 6.2. In case 2, the embedding of the bound nodes depends on the given source tree pattern and the source AML document, because bound target nodes (partially) reproduces their source correspondences. Finally, case 3 requires that for each edge in P there is a corresponding edge in the target AML document.

It remains to describe what exactly does a full and a partial-copy mean during target construction.

Definition 8.3 (partial-copy)**.** Let s be an XML node in a source AML document. A partial-copy of s is an XML node t in a target AML document that satisfies:

- $t.tag = s.tag$;
- If s is a CAEX attribute, then for each XML attribute a_i of s except for `Name`, there is an XML attribute a'_i of t such that $a'_i = a_i$[5];
- If s is not a CAEX attribute, then for each XML attribute a_i of s, there is an XML attribute a'_i of t such that $a'_i = a_i$;
- If s is a text node, then $s.\texttt{value} = t.\texttt{value}$. □

Definition 8.4 (full-copy)**.** Let s be an XML node in a source AML document. A full-copy of s is an XML node t in a target AML document that satisfies:

- t is a partial-copy of s;
- for each child node $sChild$ of s, t has a child node $tChild$ which is a full-copy of $sChild$. □

In other words, a full-copy of a source XML node s reproduces the entire tree of s in the target XML document. Consider the data exchange example, as shown in Fig. 8.3. This example describes a data exchange that, for all descendant internal elements `IE1` in the source data which contain at least one `Robot` (`IE2`) and one `Clamp` (`IE3`), construct a target `RobotCell` (`IE4`) that contains a full-copy of each `Robot` and a partial-copy of each `Clamp`.

One last challenge is the default existential quantification within the STP. For example, the existential quantification on the `IE2` and `IE3` in Fig. 8.3 indicates that there might be several `Robots` and `Clamps` under the embedding of `IE1`, as shown in the following example.

Example 8.1 (Possible embeddings for Q in Fig. 8.3)**.** Let $S = (V, F)$ be a source AML document, and $ie1, ie2, ie3, ie4$ be distinct tree nodes in V. Let e_1 and e_2 be two possible embeddings from Q to S with:

(i) $e_1(\texttt{IE1}) = ie1, e_1(\texttt{IE2}) = ie2, e_1(\texttt{IE3}) = ie3$;

(ii) $e_2(\texttt{IE1}) = ie1, e_2(\texttt{IE2}) = ie4, e_2(\texttt{IE3}) = ie3$.

Now consider possible target embeddings from P in Fig. 8.3 to a target AML document T. Definition 8.2 states that for any target embedding e' from P to T, if e' satisfies a source embedding e, then $e'(\texttt{IE5})$ must be a full-copy of $e(\texttt{IE2})$ because `IE2` is the source correspondence of the bound target node `IE5`. However, in the case of Example 8.1, there are two source embeddings e_1 and e_2 from Q to S. So the question is, shall e' satisfy both e_1 and e_2 or only one of them?

The answer to the above question relates to the two possible semantics of data exchange: Cartesian product [32] or grouping [97, 96]. Under Cartesian product semantics, one target embedding needs to satisfy only one source embedding. In other words, several instances of `IE4` are constructed, each of which according to one possible embedding from Q to S.

[5]The equality between XML attributes denotes the equality between their names and their values.

Example 8.2 (Cartesian Product)**.** Consider the source embeddings e_1 and e_2 in Example 8.1. Under the Cartesian product semantics, for each pair (x, y) of `Robot` and `Clamp`, an `IE4` is constructed that contains x and y as child nodes. Because `IE2` has two embeddings $ie2$ and $ie4$, two target internal elements corresponding to `IE4` are generated: `IE4-1` that contains $(ie2, ie3)$ as child nodes, and `IE4-2` that contains $(ie4, ie3)$ as child nodes.

Obviously, the Cartesian product semantics introduces duplication of source data in the target. In Example 8.2, the source XML node $ie3$ appeared twice in the target, each of which is put under a different instance of `IE4`. In engineering, it is desired to construct one instance of `IE4` under the grouping semantics such that ie_2, ie_3, ie_4 are copied as child nodes of the same `IE4` instance [100]. To formally define the grouping semantics, a *context* needs to be fixed for the corresponding grouping node.

Definition 8.5 (Context Node)**.** Let $Q = (N, E)$ be a source tree pattern and $P = (N', E')$ be a target tree pattern. Let $t \in N'$ be a TTP node and $DB(t) = \{d_1, \cdots, d_k\}$ be the set of descendant bound nodes of t. Let s be the source correspondence of t ($s = null$ if t is free) and $s_1, \cdots, s_l$ be the source correspondences of $d_1, \cdots, d_k$, with $l \leq k$. The context node of t, written as $context(t)$, is defined as:

(i) if t is free and $|DB(t)| = 0$, then $context(t) = null$;

(ii) if t is bound and $|DB(t)| = 0$, then $context(t)$ is the source correspondence of t;

(iii) if t is free and $|DB(t)| \neq 0$, then $context(t)$ is the least common ancestor (LCA) of $s_1, \cdots, s_l$;

(iv) if t is bound and $|DB(t)| \neq 0$, then $context(t)$ is the least common ancestor (LCA) of $s, s_1, \cdots, s_l$;

In the definition above, the total number of source correspondences $s_1, \cdots, s_l$ can be smaller than the total number of the descendant bound nodes of n, because it is allowed to bind different target nodes with the same source node. The listed four cases cover two situations that might occur. On the one hand, if the target node t has no bound descendant, i.e., case (i) and (ii), then grouping is not needed. On the other hand, if the target node t has bound descendants, then case (iii) and (iv) define the context of t to be the LCA of the source correspondences of the bound nodes.

Example 8.3 (Context Node)**.** For the node `IE4` in Fig. 8.3, its bound descendants are `IE5` and `IE6`, whose source correspondences are `IE2` and `IE3`, respectively. Therefore, $context($`IE4`$)$ is the LCA of `IE2` and `IE3`, that is, the node `IE1`.

Definition 8.6 (Grouping)**.** Let $S = (V, F)$ be a source AML document and $T = (V', F')$ be a target AML document. Let $Q = (N, E)$ be an STP, $P = (N', E')$ be a TTP, and let the node $s \in N$ be the context node of the node $t \in N'$. Let $e_1, \cdots, e_n$ be valid embeddings from Q to S with $e_1(s) = \cdots = e_n(s)$, and $e'_1, \cdots, e'_n$ be valid target embeddings from P to T that satisfy $e_1, \cdots, e_n$, respectively. Then T is constructed under the grouping semantics, if $e'_1(t) = \cdots = e'_n(t)$.

Definition 8.6 shows that under the grouping semantics, each instance of a target node t is constructed for a set of source embeddings $e_1, \cdots, e_n$ that matches $context(t)$ to the same source XML node. In other words, the construction of each distinct instance of t depends on the embedding of $context(t)$, which in turn depends on the source correspondences of t and its bound descendants. These dependencies can be represented by an XQuery for-return clause that restricts the instances of t to distinct embeddings of $context(t)$ (see Definition 8.9 in Section 8.4.1). Moreover, if t is a free node with the XML tag IE or EI, then a so-called *Skolem function* [97, 96] is used to create the UUID of the instances of t. The Skolem function takes as input the source correspondences $\{s, s_1, \cdots, s_l\}$ (recall that $s_1, \cdots, s_l$ are the source correspondences of the descendant bound nodes of t, cf. Definition 8.5) and outputs a UUID for each distinct set of them.

Example 8.4 (Grouping)**.** Consider the source embeddings e_1 and e_2 in Example 8.1. Under the grouping semantics, because $context(\texttt{IE4}) = \texttt{IE1}$ and $e_1(\texttt{IE1}) = e_2(\texttt{IE1})$, only one instance of `IE4` is constructed, which contains ie_2, ie_3, ie_4 as child nodes.

Note that making a choice between Cartesian product and grouping semantics is not required for the free target nodes that have no bound descendant node (case (i) in Definition 8.5), e.g., `RR3` in Fig. 8.3. Until now, the meaning of TTP and its relation to STP has been presented. The following definition provides a formal setting for AML data exchange.

Definition 8.7 (Data Exchange Setting for AML)**.** A data exchange setting for AML is a tuple (Q, P, F), where Q is an STP, P is a TTP, and F is a Skolem function. Let S be a source AML document and $Q(S)$ be the set of all source embeddings from Q to S. A target AML document T is a solution for S w.r.t (Q, P, F), if:

1. for each source embedding $e \in Q(S)$, there is a target embedding e' from P to T, such that e' satisfies e;
2. T is constructed under the grouping semantics. □

Case 1 in Definition 8.7 ensures the consistency between the source and target AML documents under the pair (Q, P), while case 2 additionally requires that T is constructed under the grouping semantics.

8.3. Extending AQT for Data Exchange

Based on the data exchange setting above, this section extends AQT for query modeling using AML. Similar to XQBE [32] and VXQ [33], the extended AQT consists of two parts: (i) a source AQT I that describes what is needed from the source AML document, and (ii) a target AQT J that specifies how to construct a target AML document.. Essentially, I corresponds to an STP and J corresponds to a TTP. A pair of (I, J) is called an st-AQT (**s**ource-to-**t**arget AQT).

Table 8.2.: The configuration parameters of a bound target tree node.

Name	Type	Default
`sourceID`	`string`	-
`mode`	`string`	`copy`

8.3.1. Syntax Extensions

Because the design principle of AQT is to use AML for query modeling, both the source AQT I and the target AQT J are AML models. For the source part, the original AQT syntax introduced in Section 7.2.1 can be reused. However, the `returned` parameter in Table 7.2 is not required anymore.

The main task of the extension is to develop a syntax for the target AQT J, which shall be interpreted as a TTP. According to Definition 8.1, a TTP node can be either *free* or *bound*. The former is target-specific while the latter relates to nodes in an STP via *bindings*. Intuitively, a free node can be directly represented in AML. For the bound nodes, two CAEX attributes are defined as query-specific configuration parameters, as shown in Table 8.2.

The parameter $\texttt{mode} \in \{\texttt{copy}, \texttt{projection}\}$ specifies the type of a binding between an STP and a TTP node as follows:

- $\texttt{mode} = \texttt{copy}$ refers to a *full-copy* binding as described in Definition 8.4.
- $\texttt{mode} = \texttt{projection}$ refers to an extended partial-copy that also takes into account the source RRs, SRCs, and ILs, because they are essential information in most engineering activities. Therefore, it is beneficial to simplify the data exchange models by handling them as part of the partial-copy by default. For convenience, a binding with $\texttt{mode} = \texttt{projection}$ is called a *projection* binding.

The parameter `sourceID` denotes the corresponding source node that is bound to the given target node. The value of `sourceID` adheres to the standard identification mechanism in AML, i.e., the UUID of the source internal element or external interface, or the full path of the corresponding source CAEX attribute. Because the target CAEX attribute can have a different name than its source correspondence, the binding between two CAEX attributes is called a *renaming* binding. The target configuration parameters `mode` and `sourceID` are embedded into the CAEX attribute `targetConfig` in the target AQT. Intuitively, a target node is bound if it is associated with a `targetConfig` and free otherwise.

Fig. 8.4 shows the raw XML serialization of an st-AQT (I, J). Both I and J are AML instance hierarchies. I describes an STP with the root node `IH1` which has a descendant node `IE1`. Furthermore, `IE1` should contain one `Robot` and one `Clamp`. J describes a TTP with the root node `IH2` which has a free child node `IE4`. `IE4` in turn, has two bound child nodes `IE5` and `IE6`. More specifically, `IE5` represents a full-copy of the node `IE2` in I because $\texttt{IE5.mode} = \texttt{copy}$ and `IE5.sourceID` is the UUID of `IE2`, while `IE6` denotes a partial-copy of the node `IE3` in I because $\texttt{IE6.mode} = \texttt{projection}$ and `IE6.sourceID` is the UUID of `IE3`.

In Section 7.2.1, a formal representation of the source AQT has been introduced, which is based on the notion of rooted trees. This formal representation also applies to the target AQT. The only difference is the set B of target tree nodes, which contains the

```xml
<InstanceHierarchy Name="IH1">
    <InternalElement Name="IE1" ID="a33e7848-b614-4fee-969a-6ec451a747ab">
        <Attribute Name="queryConfig">
            <Attribute Name="descendant" AttributeDataType="xs:boolean">
                <Value>True</Value>
            </Attribute>
        </Attribute>
        <InternalElement Name="IE2" ID="f5e47db0-79a6-46a4-8c97-40bc3e31c63b">
            <Attribute Name="queryConfig"></Attribute>
            <RoleRequirements RefBaseRoleClassPath="AutomationMLDMIRoleClassLib/
                DiscManufacturingEquipment/Robot" />
        </InternalElement>
        <InternalElement Name="IE3" ID="45ffc72a-81b4-4131-b5c0-80df7172aa85">
            <Attribute Name="queryConfig"></Attribute>
            <RoleRequirements RefBaseRoleClassPath="AutomationMLExtendedRoleClassLib/
                Clamp" />
        </InternalElement>
    </InternalElement>
</InstanceHierarchy>
```

(a) The source AQT I.

```xml
<InstanceHierarchy Name="IH2">
    <InternalElement Name="IE4" ID="2ef84bca-0c62-4be4-97e9-12adb3c4cdae">
        <InternalElement Name="IE5" ID="c3e5a38e-abfa-4133-bfd8-6f40e0774082">
            <Attribute Name="targetConfig">
                <Attribute Name="mode" AttributeDataType="xs:string">
                    <Value>copy</Value>
                </Attribute>
                <Attribute Name="sourceID" AttributeDataType="xs:string">
                    <Value>f5e47db0-79a6-46a4-8c97-40bc3e31c63b</Value>
                </Attribute>
            </Attribute>
        </InternalElement>
        <InternalElement Name="IE6" ID="deed94c5-28ed-429a-8cb3-1b4a3845f4ce">
            <Attribute Name="targetConfig">
                <Attribute Name="mode" AttributeDataType="xs:string">
                    <Value>projection</Value>
                </Attribute>
                <Attribute Name="sourceID" AttributeDataType="xs:string">
                    <Value>45ffc72a-81b4-4131-b5c0-80df7172aa85</Value>
                </Attribute>
            </Attribute>
        </InternalElement>
        <RoleRequirements RefBaseRoleClassPath="TargetRoleClassLib/RobotCell" />
    </InternalElement>
</InstanceHierarchy>
```

(b) The target AQT J.

Figure 8.4.: An st-AQT (I, J).

target configuration parameters `mode` and `sourceID`. Again, B can be empty for some nodes in the target tree because CAEX attributes can not be associated with conditional $\mathrm{CAEXBasicObjects}$. In other words, conditional $\mathrm{CAEXBasicObjects}$ cannot be bound to any source nodes. Nevertheless, since concrete engineering data is stored within CAEX attributes and organized in the nested structure of internal elements and external interfaces, only target internal elements, external interfaces, and CAEX attribute nodes need to be bound. Furthermore, if a target-specific conditional $\mathrm{CAEXBasicObjects}$ is required during the construction, it can be modeled as a free node in the target tree, such as the RR node in Fig. 8.4b.

8.3.2. Semantics Extensions

Let $I = (V, F)$ be a source AQT and $J = (V', F')$ be a target AQT. The following shows how to construct a pair $(Q = (N, E), P = (N', E'))$ where Q is an STP and P is a TTP:

1. Construct $Q = (N, E)$ from I following Section 7.2.2, because an STP is essentially a TPQ in which each node has `returned` $= false$.
2. Construct $P = (N', E')$ from J as follows:
   a) For each bound node $v \in V'$ with $v = (id, tag, A, B)$ and $B \neq \emptyset$, construct a bound TTP node $t = (id, tag, name, sid, full)$ in N, where $t.id = v.id$, $t.tag = v.tag$, $t.name = v.A.$`Name`, and $t.sid = v.B.$`sourceID`. If $v.B.$`mode` $==$ $copy$, then $t.full = true$, otherwise $t.full = false$.
   b) For each bound node $v \in V'$ with $v = (id, tag, A, B)$ and $B \neq \emptyset$, if $v.B.$`mode` $==$ `project` and $v.tag == \mathsf{IE}$, then add the following *default bindings* to P and Q. Let $s \in N$ be the source correspondence of $t \in N'$, then:
      i. add a node $t_{\mathsf{RR}} = (id, \mathsf{RR}, \text{“}rr\text{”}, sid, true)$ to N', and a node $s_{\mathsf{RR}} = (id, \mathsf{RR}, \emptyset, \emptyset)$ to N, with $t_{\mathsf{RR}}.sid = s_{\mathsf{RR}}.id$. Add (s, s_{RR}) as an edge to E and (t, t_{RR}) as an edge to E'.
      ii. add a node $t_{\mathsf{SRC}} = (id, \mathsf{SRC}, \text{“}src\text{”}, sid, true)$ to N', and a node $s_{\mathsf{SRC}} = (id, \mathsf{SRC}, \emptyset, \emptyset)$ to N, with $t_{\mathsf{SRC}}.sid = s_{\mathsf{SRC}}.id$. Add (s, s_{SRC}) as an edge to E and (t, t_{SRC}) as an edge to E'.
      iii. add a node $t_{\mathsf{IL}} = (id, \mathsf{IL}, \text{“}il\text{”}, sid, true)$ to N', and a node $s_{\mathsf{IL}} = (id, \mathsf{IL}, \emptyset, \emptyset)$ to N, with $t_{\mathsf{IL}}.sid = s_{\mathsf{IL}}.id$. Add (s, s_{IL}) as an edge to E and (t, t_{IL}) as an edge to E'.
   c) For each free node $w \in V'$ with $w = (id, tag, A, \emptyset)$, construct a free node $n' = (id, tag, C)$ in N', where $n'.id = w.id$, $n'.tag = w.tag$, and add each value constraint in $w.A$ to $n'.C$.
   d) For each edge $f' = (v'_i, v'_j)$ in F', add an edge $e' = (n'_i, n'_j)$ in E', where v'_i is mapped to n'_i and v'_j is mapped to n'_j.

It can be observed that the construction of P also influences the already constructed source tree pattern Q. More specifically, step 2b) enriches both P and Q with the default bindings for RRs, SRCs, and ILs. Note that the default bindings only exist in the pair (Q, P) but not in the pair (I, J). Also note that CAEX attributes are not considered as default bindings, because it is assumed that the source and target engineering tools may use different attribute names. Therefore, if one source CAEX attribute shall be conveyed to the target side under the `projection` mode, it must be explicitly modeled within the target AQT.

The semantics defined above allow the visualization of st-AQTs as tree structures, as specified in Section 8.2. In particular, Fig. 8.3 is the visualization of the example shown in Fig. 8.4. In the following, more examples of st-AQTs are presented.

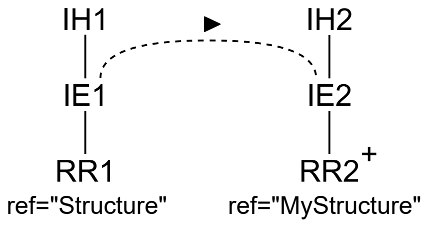


Figure 8.5.: (Q1) role class equality.

Fig. 8.5 shows the query Q1 that states "from each source IH, copy each child `Structure` into the target, and extend them with a RR of the class `MyStructure`." From the perspective of semantic correspondence patterns, Q1 describes the equivalence between the source role class `Structure` and the target role class `MyStructure`.

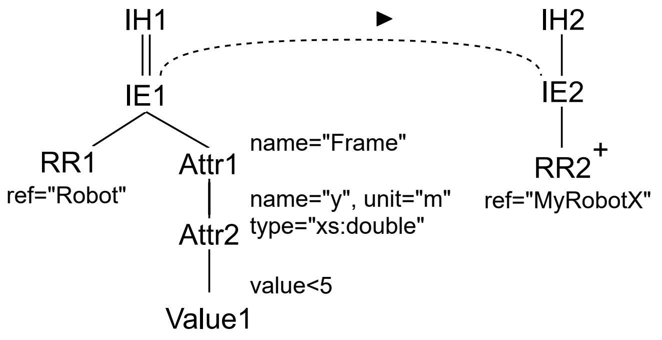


Figure 8.6.: (Q2) subclass.

Fig. 8.6 shows the query Q2 that states "from each source IH, copy all descendant `Robots` whose nested CAEX attribute `Frame.x` has a value smaller than 5." Essentially, Q2 extends Q1 with the value constraint on the CAEX attribute `Frame.y` of `IE1`. In terms of semantic correspondence patterns, the target role class `MyRobotX` is a subclass of the source role class `Robot`, because only the source robots that satisfy the value constraints are members of `MyRobotX`.

Fig. 8.7 shows the query Q3 that states "from all source IEs with at least one child `Robot` and one child `Clamp`, copy the `Robots` and `Clamps` and extend them with a RR of the class `MyResource`." Note that the coexistence of `IE2` and `IE3` is necessary. Regarding the semantic correspondence patterns, the target role class `MyResource` is a superclass of both `Robot` and `Clamp`.

Fig. 8.8 extends Q3 with cardinality restrictions on `IE2` and `IE3`. Now the coexistence of `IE2` and `IE3` is not mandatory because the minimum cardinality for both of them is 0. Q3 demonstrates how to model a binding without forcing the existence of the source

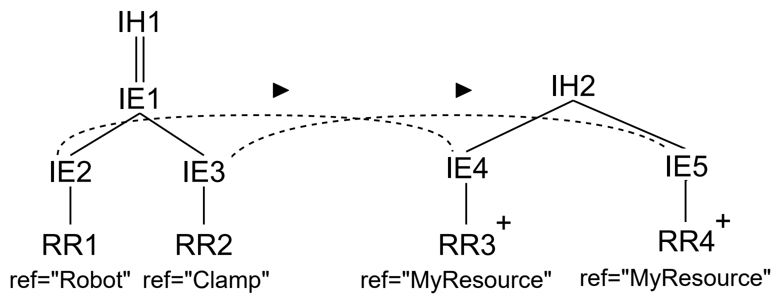


Figure 8.7.: (Q3) superclass.

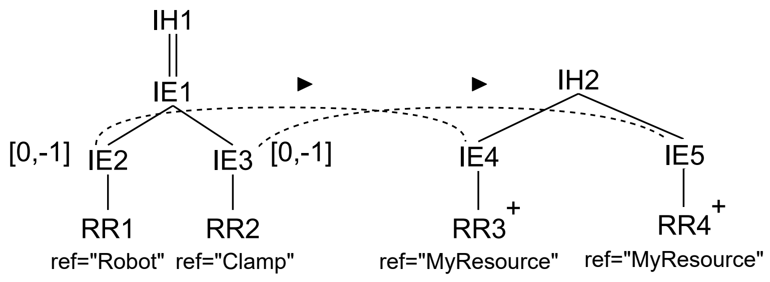


Figure 8.8.: (Q4) superclass with cardinality restrictions $[0, \infty]$.

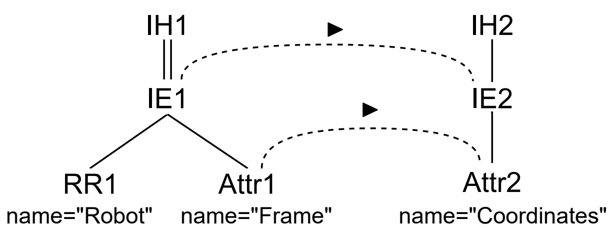


Figure 8.9.: (Q5) copy with attribute renaming.

correspondence and states “from all source IEs, copy the child Robots and Clamps and extend them with a RR of the class MyResource.”

Fig. 8.9 shows the query Q5 that states “for all source Robots with a CAEX attribute Frame, copy the Robots and change Frame to Coordinates”. Q5 demonstrates how to change CAEX attribute names during a full-copy.

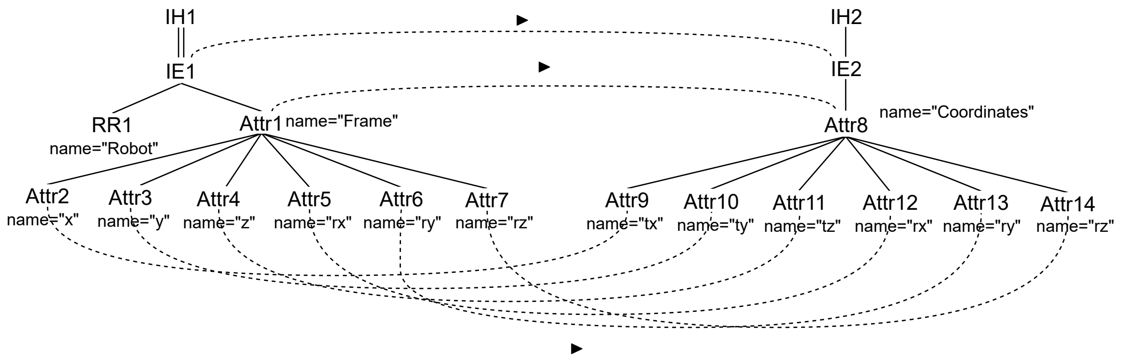


Figure 8.10.: (Q6) copy with nested attribute renaming.

Fig. 8.10 extends Q5 by specifying the data exchange behavior of the sub-attributes of the source XML attribute Frame. In particular, the name of Frame.x, Frame.y, and Frame.z are changed to Coordinates.tx, Coordinates.tx, and Coordinates.tz, respectively, while Frame.rx, Frame.ry and Frame.rz are copied without changes.

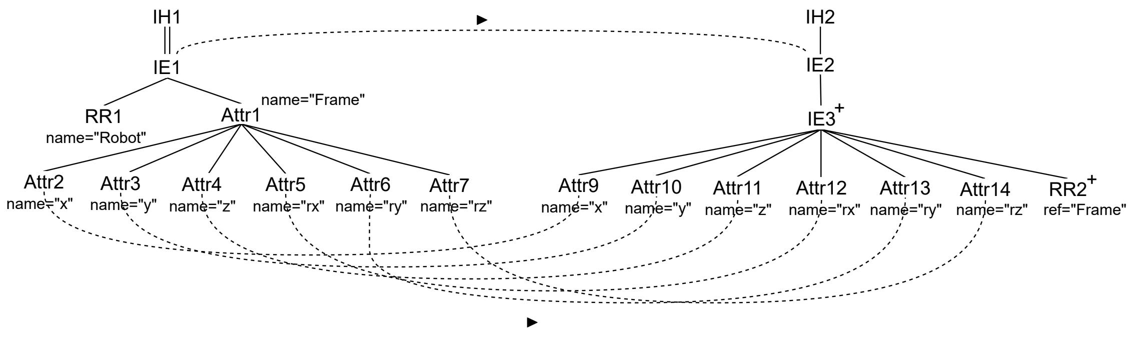


Figure 8.11.: (Q7) from attribute to object relation.

Fig. 8.11 shows the query Q7 that has a similar structure to Q6 but maps the source CAEX attribute Frame to a target internal element IE3 of type Frame. The sub-attributes of the source CAEX attribute Frame are copied as attributes of IE3. Q7 demonstrates the property to relation mapping (CP8 in Section 8.1.2).

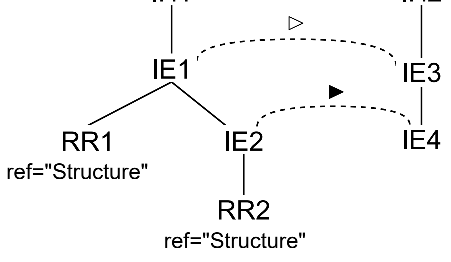


Figure 8.12.: (Q8) projection.

Fig. 8.12 shows the query Q8 that states "for all source IHs with a child `Structure`, which in turn has a child `Structure`, project the outer `Structure` to the target while keeping the nested one." Note that according to the semantics of st-AQTs, `RR1` belongs to the default bindings and will be copied to `IE3`.

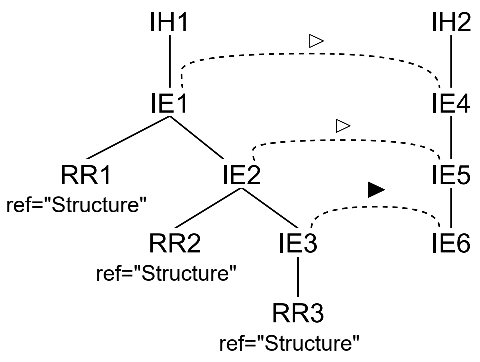


Figure 8.13.: (Q9) nested projection.

Fig. 8.13 extends Q8 with a nested projection of `IE1` and `IE2`. Now, `IE2` is not bound to the target with a full-copy binding but projected to the target node `IE5`. Consequently, the instances of `IE4` will only contain data from the embeddings of `IE1` that belong to the default binding, while the instances of `IE5` will comprise the embeddings of `IE3` in addition to the projection of `IE2`.

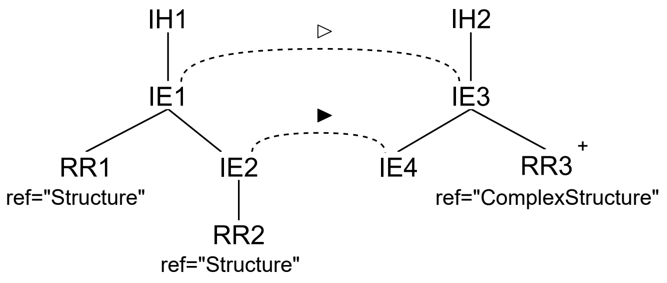


Figure 8.14.: (Q10) projection with new target elements.

Fig. 8.14 also represents an extension of Q8. The instances of `IE3` will be enriched with a new RR that refers to the class `ComplexStructure`. Q10 shows how to extend a projected node with target-specific data.

Fig. 8.15 has the similar structure to Q6. However, the full-copy bindings between $(\texttt{IE1}, \texttt{IE2})$ and $(\texttt{Attr1}, \texttt{Attr2})$ are changed to projection bindings, such that the instances of the TTP node `Attr8` will only have three sub-attributes, which are copied from the STP nodes `Attr2`, `Attr3` and `Attr4`.

Fig. 8.16 shows the query Q12 that changes the TTP of Q9 by deleting one TTP node. Q12 thus does not project data from the embeddings of `IE2` but nests the embeddings of `IE3` into the corresponding instance of `IE4`. Due to the grouping semantics, each instance of `IE4` corresponds to a distinct embedding $e(\texttt{IE1})$, under which all embeddings of `IE3` that share $e(\texttt{IE1})$ as a predecessor will be grouped together.

Fig. 8.17 shows the query Q13 that states "for all source `Structures` with at least one child `Robot` and one child `Clamp`, construct one IE4 that contains the full-copy of all the `Robots`, and one `IE5` that contains the full-copy of all the `Clamps`, and add a new RR to

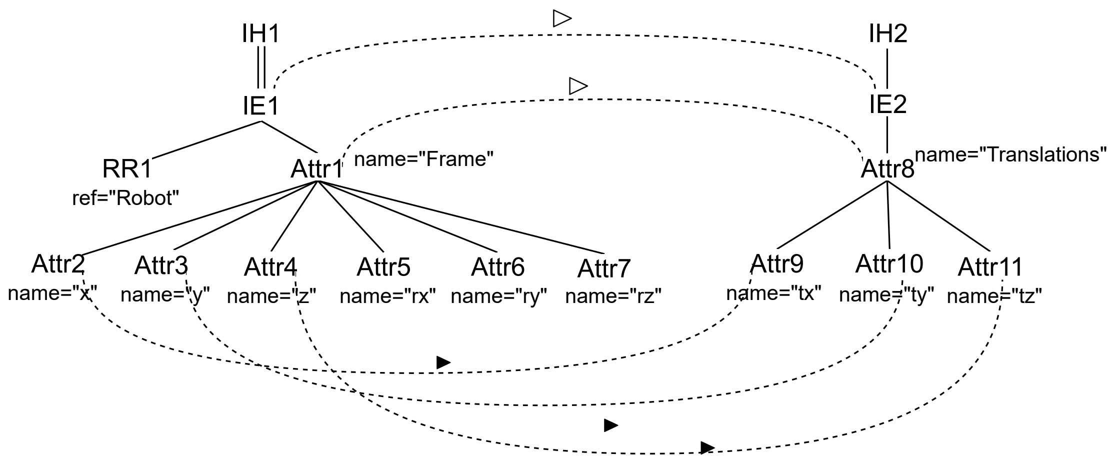


Figure 8.15.: (Q11) projection with attribute renaming.

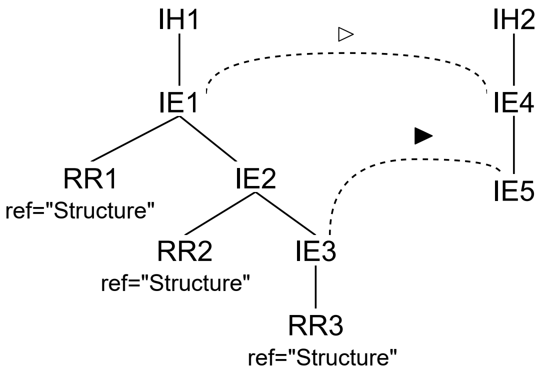


Figure 8.16.: (Q12) projection while ignoring intermediate data.

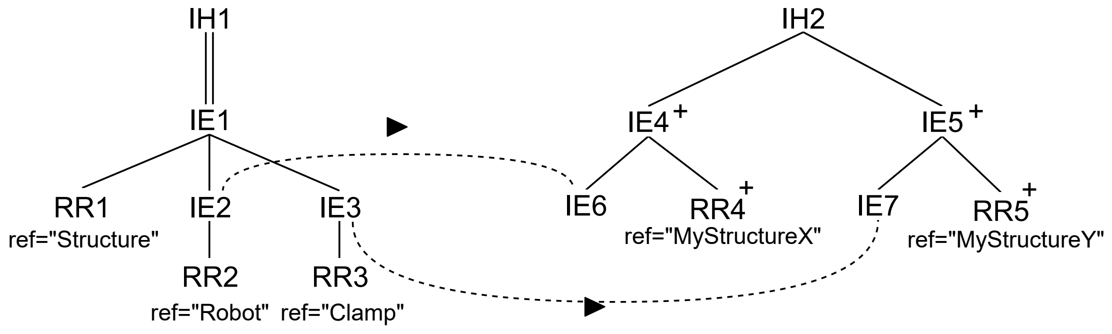


Figure 8.17.: (Q13) object decomposition.

IE4 and IE5, respectively." Under the grouping semantics, because the context of both IE4 and IE5 are the STP node IE1, each distinct embedding of IE1 will introduce one pair of instances for IE4 and IE5. Essentially, Q13 decomposes the STP node IE1 into two target nodes IE6 and IE7, which are restructured according to the requirements of the target AML document.

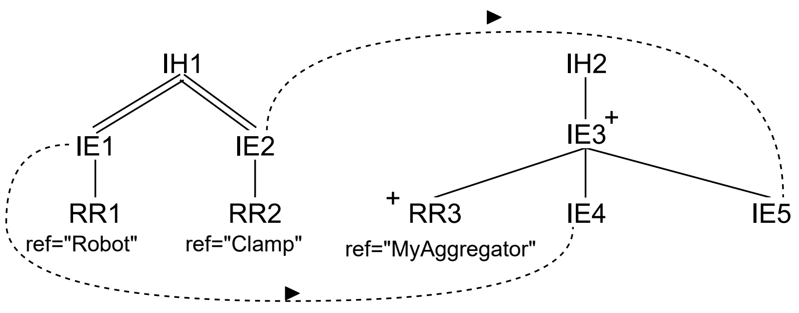


Figure 8.18.: (Q14) object aggregation.

Fig. 8.18 shows the query Q14 that states "aggregate all Robots and Clamps in the source file under a new object of the class MyAggregator in the target file." Again, because of the grouping semantics, no Cartesian product for each pair of (Robot, Clamp) is built. Instead, all embeddings of IE1 and IE2 under a given instance of IH1 are stored under one instance of IE3.

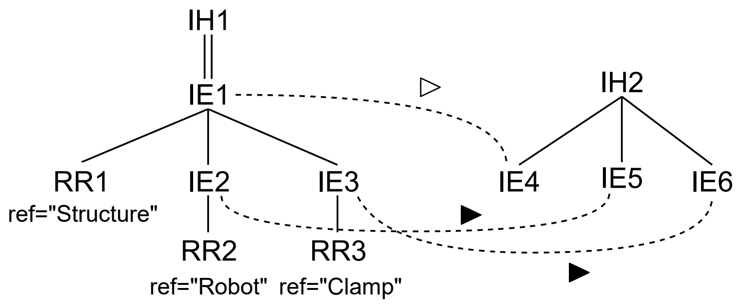


Figure 8.19.: (Q15) object flattening.

Fig. 8.19 shows the query Q15 that changes the TTP of Q13. Moreover, the STP node IE1 is now projected to the target node IE4, while its children IE2 and IE3 are copied as siblings of IE4. In other words, the nested structure of IE1(IE2, IE3) is flattened to a list (IE4, IE5, IE6).

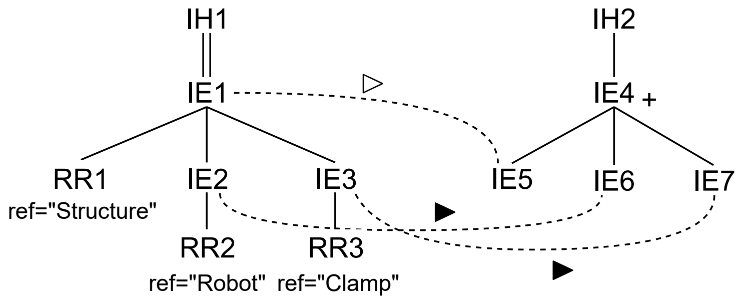


Figure 8.20.: (Q16) object flattening and grouping.

Fig. 8.20 extends Q15 with a free node IE4 in the target model. Q16 thus flattens the nested structure of IE1 and groups the flattened objects $(\mathtt{IE5}, \mathtt{IE6}, \mathtt{IE7})$ again into the target node IE4. The grouping semantics assures that for each distinct embedding of IE1, an instance of IE4 will be constructed.

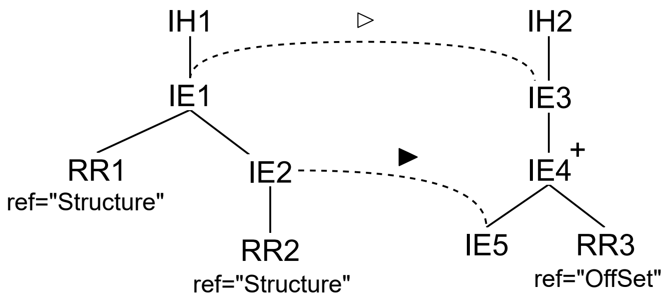


Figure 8.21.: (Q17) object offsetting and nesting.

Fig. 8.21 shows the query Q17 that states "for each source IH with nested Structures, project the outer Structure to the target, and add an intermediate IE4 of the class OffSet between the outer and the inner Structure in the target." Q16 intends to break the parent-child relation of $(\mathtt{IE1}, \mathtt{IE2})$ in the STP, and generate target-specific data between them.

Table 8.3.: Summary of the data exchange features covered by Q1-Q14.

Query Example	C.M. Scenarios	S.C. Patterns	XQuery Features
Q1	OM1	CP1	XF1-XF4,XF6
Q2	OM1	CP3,CP4	XF1-XF4,XF6
Q3	OM3,SM2	CP2	XF1-XF4,XF6
Q4	OM3,SM2	CP2	XF1-XF4,XF6
Q5	OM1	CP6	XF1-XF4,XF5
Q6	OM1	CP6	XF1-XF4,XF8
Q6	OM1	CP6	XF1-XF4,XF8
Q7	OM1	CP5, CP8	XF1-XF4,XF8
Q9	OM1,SM1	CP5	XF1-XF4,XF5
Q10	OM1,SM1	CP5	XF1-XF4,XF5,XF6
Q11	OM1,SM1	CP5,CP6	XF1-XF4,XF5
Q12	OM1,SM1	CP5	XF1-XF4,XF7,XF10
Q13	OM1,OM2	-	XF1-XF4,XF6
Q14	OM1,OM3	CP7	XF1-XF4,XF6,XF8,XF9
Q15	OM1,OM2,SM1,SM3,SM4	CP5,CP7	XF1-XF4,XF5,XF9
Q16	OM4,SM1,SM3,SM4	CP5,CP7	XF1-XF4,XF5,XF6,XF9
Q17	OM2,SM1,SM5	CP5	XF1-XF4,XF5,XF6,XF7

Table 8.3 summarizes the data exchange features covered by the query examples Q1-Q17. It can be seen that all requirements of the common mapping scenarios (OM1-OM4, SM1-SM5) and the semantic correspondence patterns (CP1-CP8) are covered. Consequently, XQuery features XF1-XF9 are supported by AQT. Furthermore, Q12 shows how the depth projection (XF10) is realized. In the next section, algorithms and implementations are presented for translating st-AQTs into XQuery programs.

8.4. Extending AQP for Data Exchange

This section extends the AML Query Processor (AQP, cf. Fig. 7.1b) for AML data exchange based on the extended AQT. More concretely, given an AML data exchange setting (Q, P, F) where Q is an STP, P is a TTP, and F is a Skolem function, construct an XQuery program that generates a solution T for every possible source AML document S.

8.4.1. Code Templates

The extension of the AQP is based on the so-called *identity transformation*, which is shown as the XQuery function fCopy in Listing 8.1.

Listing 8.1 Implementation of full-copy in XQuery

```
1 declare function caex:fCopy($element as element()){
2     element {node-name($element)}
3     {
4         $element/@*,
5         for $child in $element/node()
6         return
7             if ($child instance of element())
8             then caex215:fCopy($child)
9             else $child
10    }
11 };
```

The function fCopy takes as input one XML element $\$element$ and outputs a new XML element that copies the content of $\$element$. It starts by constructing the XML tag of the new element in line 2, using the XML tag of $\$element$. Then it adds all the XML attributes of $\$element$ to the new element in line 4. From line 5 to line 9, fCopy adds the results of the recursive calls on the children of $\$element$ into the new element in a depth-first manner. Recall Definition 8.4, fCopy implements the full-copy from an input source XML element to an output target XML element.

Identity transformation is a very flexible XQuery code structure that can be adapted for specific purposes. For example, the following variation of the identity transformation implements the partial-copy in Definition 8.3. In contrast to Listing 8.1, The function pCopy does not traverse the child nodes but copies the text content of $\$element$.

Listing 8.2 Implementation of partial-copy in XQuery

```
declare function caex:pCopy($element as element()){
    element {node-name($element)}
    {
        $element/@*,
        $element/text()
    }
};
```

The above two code examples can be used as pure XQuery functions and are implemented in the `caex` XQuery module. However, for the construction of a target AML document, they are not sufficiently expressive, because neither fCopy nor pCopy provides means for handling potential child nodes of a target node in P. For example, a full-copied target node may contain some child nodes that need to be renamed, e.g., Q5 in Fig. 8.9. In the following, the so-called *code templates* are presented, which serve as the blueprint of three basic types of XQuery programs that may be constructed in AQP, i.e., copy, projection, and renaming.

8.4.1.1. Code Template for Copy

Listing 8.3 Template for a full-copy binding (sn, tn)

```
 1 element {node-name($sn)}
 2 {
 3     $sn/@*,
 4     for $child in $sn/node()
 5     return(
 6     (:cascaded if-then-else clause for internal children of tn
          :)
 7     )
 8     (:for-return clauses for external children of tn:)
 9     (:raw XML text for simple free children of tn:)
10 }
```

Listing 8.3 shows the code template for a full-copy binding between a STP node sn and a TTP node tn. The notion $\$sn$ refers to the XQuery variable of sn and corresponds to sn's embeddings $e(sn)$ in a source AML document. Intuitively, a target AML document needs to copy data from the sub-tree of $e(sn)$ to the position of tn. However, tn might have the following three kinds of child nodes, which also need to be covered by the copy:

- Simple free children: free child nodes of tn which have no bound descendants. Simple free children do not need any particular translation. Their raw XML text can be used by XQuery for target construction (line 9 in Listing 8.3).
- Internal children: internal children are bound child nodes of tn whose source correspondences are also child nodes of tn's source correspondence. Consider Q9 in Fig. 8.13. The target node `IE5` is an internal child of `IE4` because the source node `IE2` is a child of `IE1`. On the contrary, `IE5` in Fig. 8.16 is not an internal child of its parent `IE4`, because the source correspondence of `IE5` is not a child of the source correspondence of `IE4`. Internal children are a specific set of target nodes that are translated to cascaded if-then-else clauses. The term “internal” refers to the fact that the XQuery program of such nodes constitutes the return clause of the XQuery program of their parent node (line 6 in Listing 8.3).
- External children: external children are those child nodes of tn that are neither simple free nor internal. Essentially, external children include (i) complex free children which contain any bound descendants, and (ii) bound children whose source correspondence are not children of the source correspondence of tn. Note that if tn is free, then all its bound children must be external since tn has no source

correspondence. External children are translated to additional for-return clauses outside of the return clause of tn (line 8 in Listing 8.3).

Let $tChild$ be an **internal child** of t and $sChild$ the source correspondence of $tChild$ in the STP Q. The function call $\mathrm{BuildIfThenElse}(tChild, Q)$ (Definition 8.8) constructs an if-then-else clause for transferring data from the $sChild$ to $tChild$. The if clause selects the $sChild$ from the source document, the then clause contains the XQuery program for the full-copy binding $(sChild, tChild)$, and the else clause is a placeholder for further children of tn. Listing 8.4 shows an example of an if-then-else clause that copies the source CAEX attribute `Frame` to the target side.

Definition 8.8 ($\mathrm{BuildIfThenElse}(tn, Q)$)**.** Let tn be a TTP node and Q be the associated STP. The function $\mathrm{BuildIfThenElse}(tn, Q)$ builds an XQuery if-then-else clause as follows:

1. let $xpath$ be the XPath expression of the source correspondence of tn
2. build an if clause xif with $xpath$ as the condition
3. build an empty then clause $xthen$
4. build an empty else clause $xelse$
5. return $xif, xthen, xelse$

Listing 8.4 An if-then-else clause of a source child node

```
if ($child[self::Attr[@Name="Frame"]])
then ( caex:fCopy($child) )
else ( )
```

A cascaded if-then-else clause is a concatenated XQuery if-then-else clause, where each if clause is concatenated to the else clause of the previous one. For example, a two-level cascaded if-then-else clause looks like: if-then-else if-then-else. The generated else if has the standard meaning in common programming languages. The cascaded if-then-else clause intends to enumerate over all bound children of tn within the for-return clause in Listing 8.3. If tn has no external child, then the expression $\mathrm{caex:fCopy}(\$child)$ is added to the return clause for copying all descendant information from sn to the target, as shown in Listing 8.5:

Listing 8.5 Template for copy with no external child

```
element {node-name($sn)}
{
    $sn/@*,
    for $child in $sn/node()
    return(
        caex:fCopy($child)
    )
    (:raw XML text for simple free children of tn:)
}
```

For an **external** child, a further for-return clause is generated using the function BuildForReturn.

Definition 8.9 ($\text{BuildForReturn}(tn, tp, Q)$)**.** Let tn be a TTP node, tp be the parent of tn, and Q be the associated STP. The function $\text{BuildForReturn}(tn, tp, Q)$ builds an XQuery for-return clause as follows:

1. let ctn be the context node of tn and var be the XQuery variable assigned to ctn.
2. let $base := \text{GetBaseReference}(tn, tp, Q)$ and $baseVar$ be the XQuery variable assigned to $base$.
3. compute the XPath expression xtn between ctn and $base$.
4. let $xfor :=$ **<u>for</u>** $\$var$ **<u>in</u>** $\$baseVar/xtn$
5. let $xret$ be an empty return clause
6. return $xfor, xret$

Recall that the desired solution of an AML data exchange setting is a target AML document that is constructed under the grouping semantics. Definition 8.6 showed that grouping is achieved by fixing a context node. Therefore, BuildForReturn first computes the context nodes ctn for the target node tn. The second step is to find the so-called *base reference* of tn, which is the starting point for navigating to ctn. Intuitively, the root node of Q can always be used as such a starting point, because any node in Q (including ctn) is reachable from the root of Q. This simple intuition, however, will not work for queries that contain a hierarchy of bindings. Consider Q12 in Fig. 8.16. A for clause for `IE5` starting from the `root` of Q would be:

$$\begin{aligned}\underline{\textbf{for}}\ \$var\ \underline{\textbf{in}}\ \$root/&\texttt{IE}[\texttt{RR}[@\texttt{ref} = \text{“}\texttt{Structure}\text{”}]]/\\ &\texttt{IE}[\texttt{RR}[@\texttt{ref} = \text{“}\texttt{Structure}\text{”}]]/\\ &\texttt{IE}[\texttt{RR}[@\texttt{ref} = \text{“}\texttt{Structure}\text{”}]]\end{aligned} \tag{8.1}$$

The problem of the for clause in Formula 8.1 is that the context of the target node `IE4` is disregarded. Suppose a source AML document $T = (V, F)$ such that two embeddings e_1 and e_2 exist with $e_1(\texttt{IE1}) = v1, e_1(\texttt{IE3}) = \text{v}2$ and $e_2(\texttt{IE1}) = v3, e_2(\texttt{IE3}) = \text{v}4$, where v_1, v_2, v_3, v_4 are distinct XML nodes in V. According to the grouping semantics, two different instances of `IE4` shall be constructed because the context node for `IE4` is `IE1` and $e_1(\texttt{IE1}) \neq e_2(\texttt{IE1})$. However, the for clause in Formula 8.1 would append $v2$ and $v4$ to both instances of `IE4`, since the XPath expression in Formula 8.1 navigates to both $v2$ and $v4$ from the root. In other words, the hierarchy of the bindings $(\texttt{IE1}, \texttt{IE4})$ and $(\texttt{IE3}, \texttt{IE5})$ is not considered within the for clause. Therefore, step 2 in Definition 8.9 calls the function GetBaseReference to find the appropriate starting point for the navigation. The rest of BuildForReturn computes the XPath expression xtn starting from the base reference to the context node of tn and constructs a for clause using xtn.

Definition 8.10 ($\text{GetBaseReference(tn, tp, Q)}$)**.** Let tn be a TTP node, tp be a predecessor of tn, and Q be the associated STP. The function $\text{GetBaseReference(tn, tp, Q)}$ returns a node in Q as follows:

1. let scp be the source correspondence of tp and ctp be the context of tp.
2. let ctn be the context of tn.
3. If scp is the same as or a predecessor of ctn, return scp.

4. If ctp is the same as or a predecessor of ctn, return ctp.

The principle of $\mathrm{GetBaseReference}$ is to use either the source correspondence scp or the context node ctp of tn's predecessor as the base reference for tn, both of which must have appeared once in the generated XQuery program and can be used as an anchor point for ctn. For Q12 in Fig. 8.16, the base reference of `IE5` is the node `IE1`, which is the source correspondence and the context node of `IE4`. Let $ie1$ be the XQuery variable assigned to `IE1` in Q12. The for clause of `IE5` generated by $\mathrm{BuildForReturn}$ would be:

$$\underline{\textbf{for}}\ \$var\ \underline{\textbf{in}}\ \$ie1/\texttt{IE}[@\texttt{ref} = \text{``}\texttt{Structure}\text{''}]/\texttt{IE}[@\texttt{ref} = \text{``}\texttt{Structure}\text{''}] \tag{8.2}$$

Notably, the for clause in Formula 8.2 is simpler than Formula 8.1, because the tree pattern of `IE1` itself is eliminated. This mechanism also works correctly for uncommon situations where the target nodes have a reversed hierarchy w.r.t. their source correspondences. In this case, ctp is preferred over scp because the source correspondence of tn can be higher than the source correspondence of tp (i.e., scp) in Q. Fig. 8.22 shows such an example: the target node `IE3` is higher than `IE4` in the TTP, but the source correspondence of `IE3` is lower than the source correspondence of `IE4` in the STP. Therefore, the context node of `IE4` is `IE1` which is the same as the context node of `IE3`. Accordingly, the following for clause is generated by $\mathrm{BuildForReturn}$:

$$\underline{\textbf{for}}\ \$var\ \underline{\textbf{in}}\ \$ie1 \tag{8.3}$$

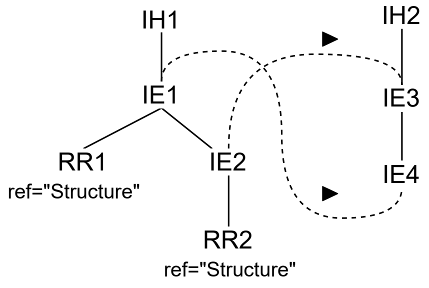


Figure 8.22.: Reversed hierarchy of target nodes w.r.t. their source correspondences.

8.4.1.2. Code Template for Projection

Listing 8.6 shows the code template for a projection binding between (sn, tn). The basic form of the template inherits from the function pCopy and the copy template. The $\mathrm{if\text{-}then\text{-}else}$ clause from line 7 to 11 copies child nodes that belong to the default bindings, as described in Section 8.3.2. The else clause is kept empty by default and can be extended with the cascaded $\mathrm{if\text{-}then\text{-}else}$ clause for the internal children of tn.

Listing 8.6 Template for a projection binding (sn,tn)

```
element {node-name($sn)}
{
    $sn/@*,
    $sn/text(),
    for $child in $sn/node()
    return
        if (caex:isProjectDefault($child$))
        then caex215:fCopy($child)
        else (
        (:cascaded if-then-else clause for internal children
           of tn:)
        )
    (:for-return clauses for external children of tn:)
    (:raw XML text for simple free children of tn:)
}
```

8.4.1.3. Code Template for Renaming

The last code template is designed for the renaming of CAEX attributes during the copy or projection from sn to tn. In contrast to previous templates, the XML attribute `Name` is not copied to the target element in line 3 but set to the value of the input parameter $\$name$ in line 4. In case sn is a nested CAEX attribute with subordinate CAEX attributes, a target CAEX attribute that is bound to sn might only take a subset of the subordinate ones. Therefore, lines 5 to 8 prepare a for-return clause for handling the internal children of tn. If tn would contain any external free children, then they are also handled outside of the return clause.

Listing 8.7 Template of renaming for a binding (sn,tn)

```
 1 element {node-name($sn)}
 2 {
 3     $sn/@*[not(name(.)="Name")],
 4     attribute{"Name"}{$name},
 5     for $child in $sn/node()
 6     return(
 7     (:cascaded if-then-else clause for internal children of tn
          :)
 8     )
 9     (:for-return clauses for external children of tn:)
10     (:raw XML text for simple free children of tn:)
11 }
```

8.4.2. Query Translation

Algorithm 8.1 shows the main body of the query translation. The algorithm takes an AML data exchange setting $S = (Q, P, F)$ and outputs the translated XQuery program. The translation begins with the variable invention and assignment for each node in the

Algorithm 8.1 TranslateToXQuery

```
Input: AML data exchange setting $S = (Q, P, F)$
Output: XQuery program $prog$
1: for each node $n$ in $Q.N$ do
2:     assign a distinct XQuery variable to $n$
3: end for
4: let $prog := \emptyset$, and $tr$ be the root node of $P$
5: let $forRet$ be a for-return clause for $tr$
6: add Construct(S, tr) to the return clause of $forRet$
7: add $forRet$ to $prog$
8: return $prog$
```

Algorithm 8.2 Construct

```
Input: AML data exchange setting $S = (Q, P, F)$, current TTP node $tn$
Output: XQuery program $prog$
1: if $tn$ is a free node then
2:     $prog = \text{ConstructNew}(S, tn)$
3: else
4:     $prog = \text{ConstructBound}(S, tn)$
5: end if
6: return $prog$
```

STP (line 2). In line 5, an XQuery for-return clause is constructed for P's root node tr. Essentially, the for clause selects all appropriate IH nodes from the source AML document, and the return clause is a place holder for the XQuery program of tr. Because tr's context node is always Q's root node sr, the for clause evaluates over each XML node in the source AML document that satisfies the XPath expression of sr, such as:

$$\begin{aligned}&\underline{\textbf{for}}\ \$root\ \underline{\textbf{in}}\ \texttt{doc(“source.aml”)/CAEXFile/IH[...]}\\&\underline{\textbf{return}}\ ()\end{aligned} \tag{8.4}$$

The XPath predicate in Formula 8.4 represents the complete structure of Q. As studied by Choi et al. [33], using the predicate of sr helps to filter the source AML document in the early phase of query evaluation and can significantly increase run-time performance. The translation proceeds with a function call construct(S, tr), which builds two different kinds of XQuery programs depending on the current TTP node, as shown in Algorithm 8.2. The result of construct(S, tr) is added to the return clause of tr in line 6. Finally, line 7 adds the for-return clause of tr to the program. Because tr is always a free IH node, construct starts with ConstructNew(S, tr) in line 2.

Since free nodes are target-specific data, identity transformation can not be used as a basis for their translation, and thus no code template is available for them. Instead, Algorithm 8.3 shows the construction of an XQuery program $prog$ for a free node tn. In line 2, the opening XML tag of tn is added to $prog$, e.g., `<InstanceHierarchy>` in the case of the root node tr. If tn was generated from an AQT internal element or external interface node, then a UUID is invented and added to the opening XML tag using the Skolem function F (line 4). If tn has XML attributes, e.g., class reference of a RR, then the XML attributes are added to the opening XML tag in line 6. Afterward, ConstructNew proceeds with the child nodes of tn. If $child$ is simple free, then the

Algorithm 8.3 ConstructNew

```
Input: AML data exchange setting $S = (Q, P, F)$, current TTP node $tn$
Output: XQuery program $prog$
 1: let $prog := \emptyset$
 2: add the opening XML tag of $tn$ to $prog$
 3: if $tn.tag ==$ IE or $tn.tag ==$ EI then
 4:     invent a UUID for $tn$ using $F$ and adds it to the opening XML tag
 5: end if
 6: add all XML attributes in $tn.C$ to the opening XML tag
 7: for each child node $child$ of $tn$ do
 8:     if $child$ is simple free then
 9:         add the XML text of $child$ to $prog$
10:     else
11:         let $fr :=$ buildForReturn($child, tn, Q$)
12:         add Construct($S, child$) to the return clause of $fr$
13:         add $fr$ to $prog$
14:     end if
15: end for
16: add the closing XML tag of $tn$ to $prog$
17: return $prog$
```

Algorithm 8.4 ConstructBound

```
Input: AML data exchange setting $S = (Q, P, F)$, current TTP node $tn$
Output: XQuery program $prog$
 1: let $prog$ be the corresponding code template for $tn$
 2: let $cascade$ be an empty if-then-else clause
 3: for each $child$ of $tn$ do
 4:     if $child$ is internal then
 5:         let $ite :=$ BuildIfThenElse($child, Q$)
 6:         add Construct($S, child$) to the then clause of $ite$
 7:         concatenate $cascade$ with $ite$
 8:     else
 9:         if $child$ is simple free then
10:             add raw XML text of $child$ to $prog$
11:         else
12:             let $fr :=$ buildForReturn($child, tn, Q$)
13:             add Construct($S, child$) to the return clause of $fr$
14:             add $fr$ to $prog$
15:         end if
16:     end if
17: end for
18: add $cascade$ to the return clause of $prog$
19: return $prog$
```

raw XML text of $child$ is added to $prog$ in line 9. Otherwise, line 11 generates a for-return clause for $child$ using the function BuildForReturn (cf. Definition 8.9), and line 12 fills the corresponding return clause with the recursive call $\mathrm{Construct}(S, child)$. Each individual for-return clause is added to $prog$ sequentially in line 13. Finally, the closing XML tag of tn is added to $prog$ in line 16.

For a bound target node tn, Algorithm 8.4 shows how to extend its corresponding code template towards a complete XQuery program. The first step of the translation is the construction of the appropriate code template for tn, as described in Section 8.4.1. From line 2 to 17, the child nodes of tn are translated. For each internal child, an if-then-else clause ite is generated (line 5) whose then part is filled with the recursive call $\mathrm{Construct}(S, child)$ (line 6). According to the code templates, all ites are concatenated together in line 7. If the child is simple free, its raw XML text is directly added to the code template (line 10). If the child is external, a for-return clause is generated where the return part is filled with the recursive call $\mathrm{Construct}(S, child)$ (line 13). Finally, line 18 adds the cascaded if-then-else clause to the return clause of the code template and line 19 returns the XQuery program.

Listing 8.8 XQuery program for Q8

```
 1  declare namespace uuid = 'java.util.UUID';
 2  import module namespace caex215 = "http://ipr.kit.edu/caex
       " at "src/main/resources/caex.xqy";
 3  for $root in doc("src/test/resources/data.aml")/CAEXFile/
       IH[IE[RR[@ref="Structure"] and IE[RR[@ref="Structure
       "]]]]
 4  return
 5  (
 6      <InstanceHierarchy>
 7      {
 8      for $n1 in $root/IE[RR[@ref="Structure"] and IE[RR[
           @ref="Structure"]]]
 9      return
10      element{node-name($n1)}
11      {
12          $n1/@*,
13          for $n2 in $n1/node()
14          return
15              if (caex215:isProjectDefault($n2))
16              then caex215:copy($n2)
17              else
18                  if ($n2[self::IE[RR[@ref="Structure"]]])
19                  then caex:fCopy($n2)
20                  else()
21      }
22      }
23      </InstanceHierarchy>
24  )
```

Listing 8.8 exemplifies the translation procedure for Q8 in Fig. 8.12. Lines 1 and 2 include the necessary declarations of namespaces, which contain the XQuery module functions required by the program. Lines 3 and 4 are the for-return clause generated for the node IH2 in Q8, where the XPath expression:

$$\texttt{IH[IE[RR[@ref = "Structure"]} \underline{\texttt{and}} \texttt{ IE[RR[@ref = "Structure"]]]]}$$

represents the complete structure of IH1. Lines 6 and 23 are the opening and closing XML tags of IH2, which are constructed by $\mathrm{ConstructNew}(S, \texttt{IH2})$. Between the XML tags, $\mathrm{ConstructNew}(S, \texttt{IH2})$ also generates a for-return clause for IE3, which is the only child of IH2. Because the context node of IH2 is IH1, $\mathrm{GetBaseReference}(\texttt{IE3}, \texttt{IH2}, Q)$ returns IH1 as the base reference for selecting the context node of IE3. Consequently, the XPath expression between IH1 and IE1 is used inside the for clause in line 8. The body of the return clause is filled with the result of $\mathrm{Construct}(S, \texttt{IE3})$, which in turn, calls the function $\mathrm{ConstructBound}(S, \texttt{IE3})$ since IE3 is a bound node. The binding between IE3 and its source correspondence IE1 is a projection binding, therefore the code template in Listing 8.6 is adopted. Finally, because IE4 is the only child of IE3, a single if-then-else clause for IE4 is added to the else clause through line 18 to 20.

The following shows that for an AML data exchange setting (Q, P, F) and an arbitrary AML document S, the presented translation constructs a solution T for S w.r.t. (Q, P, F).

According to Definition 8.7, the following two conditions must hold:

(i) for each source embedding $e \in Q(S)$, there is a target embedding e' from P to T, such that e' satisfies e;

(ii) T is constructed under the grouping semantics.

For condition (i), Definition 8.2 distinguishes following three cases:

1. for each free node $n' \in N'$, $n'.tag = embeddingT(n').tag$, and all value constraints in $n'.C$ are satisfied by $embeddingT(n')$. This case is covered by line 2 to 6 and line 16 of ConstructNew.
2. for each bound node $t \in N'$, let $s \in N$ be the source correspondence of t:
   a) if $t.full = true$, then $embeddingT(t)$ is a full-copy of $e(s)$. Otherwise, $embeddingT(t)$ is a partial-copy of $e(s)$. In ConstructBound, the instances of t are constructed based on the code templates, which implement the full- and the extended partial-copy of a source XML node $e(s)$.
   b) if $t.tag =$ Attr, then $embeddingT(t).\texttt{Name} = t.name$. This case is covered by lines 3 and 4 in the renaming template (Listing 8.7).
3. for each $(x, y) \in E'$, $(embeddingT(x), embeddingT(y)) \in F'$. Because all child nodes are handled within the scope of the code templates, their instances are nested into the instances of their parent nodes.

For condition (ii), the for-return clause constructed by the function BuildForReturn ensures that target instances are created following the grouping semantics. Furthermore, the invention of UUIDs is controlled by the Skolem function F such that each distinct instance of a free node receives a unique UUID if one is required.

8.5. Results and Discussions

The extended AQT and AQP are implemented based on the existing QBE framework described in Section 7.4. To construct a query, the user specifies an st-AQT model with a

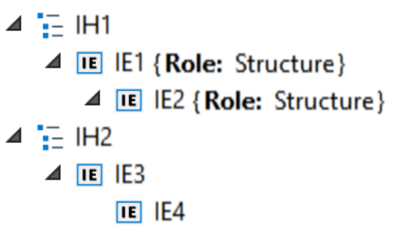
Table 8.4.: The supported XQuery features of XQBE, VXQ, and AQT. The greyed out features (XF11-XF15) are non-essential for AML data exchange becauses they were not mentioned in the requirements study in Section 8.1.

Modeling Features	**XQBE** [32]	**VXQ** [33]	**AQT**
(XF1) Copy	yes	yes	yes
(XF2) Existential quantification	yes	yes	yes
(XF3) Conjunction	yes	yes	yes
(XF4) Filtering	yes	yes	yes
(XF5) Breadth projection	yes	yes	yes
(XF6) New element	yes	yes	yes
(XF7) Nesting	yes	yes	yes
(XF8) Flattening	yes	yes	yes
(XF9) Grouping	yes	yes	yes
(XF10) Depth projection	yes	yes	yes
(XF11) Join	yes	yes	partially
(XF12) Aggregates	yes	yes	partially
(XF13) Negation	partially	partially	partially
(XF14) Arithmetic computation	yes	yes	no
(XF15) Renaming	yes	yes	no
(XF16) Cartesian product	yes	yes	no
(XF17) Sorting	yes	yes	no
(XF18) Node order	yes	yes	no
(XF19) Multiple documents	yes	yes	no
(XF20) Disjunction	no	no	no
(XF21) Union	no	no	no
(XF22) Difference	no	no	no
(XF23) Universal quantification	no	no	no

conventional AML tool, e.g., the AutomationML Editor, as shown in Fig. 8.23. Then the user triggers the query translation by feeding the st-AQT model to the AQP. The result of the query translation is an XQuery program, which is apparently much more laborious to implement than the corresponding st-AQT model (compare Listing. 8.8 with Fig. 8.23). One might argue that query implementation in a conventional programming language, such as Java or C#, would be more intuitive than the XQuery variant (cf. Fig. 7.1a). However, the manual implementation of the query program requires sufficient programming skills and knowledge of the underlying AML engine. Furthermore, manual implementations are often not reusable because different engineering projects and systems would require different software infrastructures. On the contrary, the AQT-based approach generates a query program that can be executed on all mainstream operating systems.

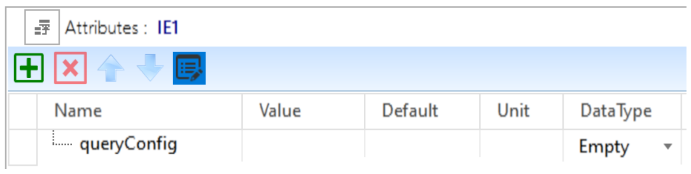
In terms of the expressive power of query models, Table 8.4 shows the supported XQuery features of XQBE [32], VXQ [33], and AQT. Obviously, both XQBE and VXQ cover more XQuery features than AQT:

- Features XF11 (join) and XF12 (aggregates) are only partially supported by AQT. For XF11, the internal link support of AQT (cf. Section 7.3.3) represents a special case of join that compares the UUIDs of external interfaces. For XF12, only the



```
◢ IH1
    ◢ IE1 {Role: Structure}
        ◢ IE2 {Role: Structure}
◢ IH2
    ◢ IE3
        IE4
```

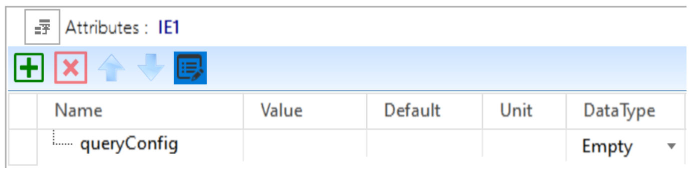
(a) The source AQT with the root node `IH1` and target AQT with the root node `IH2` of Q8.



Attributes : IE1

Name	Value	Default	Unit	DataType
queryConfig				Empty

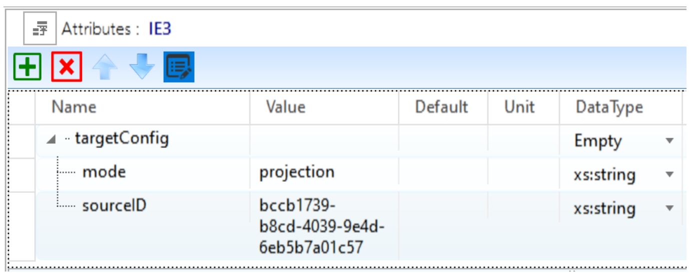
(b) The configuration parameters of IE1.



Attributes : IE1

Name	Value	Default	Unit	DataType
queryConfig				Empty

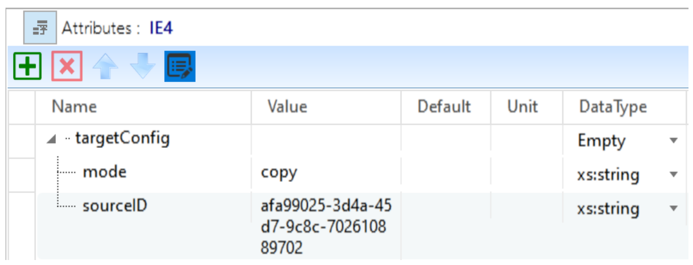
(c) The configuration parameters of IE2.



Attributes : IE3

Name	Value	Default	Unit	DataType
targetConfig				Empty
mode	projection			xs:string
sourceID	bccb1739-b8cd-4039-9e4d-6eb5b7a01c57			xs:string

(d) The configuration parameters of IE3.



Attributes : IE4

Name	Value	Default	Unit	DataType
targetConfig				Empty
mode	copy			xs:string
sourceID	afa99025-3d4a-45d7-9c8c-702610889702			xs:string

(e) The configuration parameters of IE4.

Figure 8.23.: The st-AQT of the query Q8 as shown in Fig. 8.12). The instance hierarchies `IH1` and `IH2` in sub-figure (a) represent the source and target AQT of Q8, respectively. The configuration parameters of `IE1` (sub-figure (b)) and `IE2` (sub-figure (c)) take default values and are thus omitted in the source AQT, as described in Section 7.2.1. The value of `sourceID` in sub-figure (d) corresponds to the UUID of `IE1`, while the value of `sourceID` in sub-figure (e) corresponds to the UUID of `IE2`.

function `count` is covered by AQT through the parameters `minCardinality` and `maxCardinality`.

- Feature XF13 (negation) is partially supported in all three approaches through the zero-cardinality. In the case of AQT, it is expressed using `maxCardinality`=0.
- Features XF14 to XF19 are inexpressible with AQT. However, XF15 to XF19 are not relevant for AML data exchange, while AML does not support arithmetic computation (XF14).

There are two reasons for the restricted expressiveness. First, the features XF11-XF23 are not essential for AML-based engineering because they did not appear in the requirements study in Section 8.1. Second, fully supporting these features would lead to complex query templates, which in turn, would require additional modeling efforts or an extension of the AML standard. On the other hand, as described in Section 7.4, AQT covers domain-specific modeling features that are not considered in XQBE and VXQ.

In terms of the modeling complexity, AQT is more efficient in query construction than XQBE and VXQ:

- AQT allows updating the source data during a copy or projection. For example, query Q5 in Fig. 8.9 copies the source internal element `IE1` to the target internal element `IE2` while changing the name of the CAEX attribute `Frame` to `Coordinates`. In contrast, XQBE can either copy all data without any changes or enumerate individual objects that need to be copied or updated explicitly. For Q5, one needs to draw one target node for renaming the attribute and one target node for receiving data from the source node `RR1`. Consider the embeddings of `IE1` that would contain further sub-elements, e.g., internal elements and CAEX attributes. Then XQBE requires to construct one target node for each of these sub-elements.
- The default bindings of AQT reduces the modeling efforts dramatically for projection bindings. For example, query Q10 in Fig. 8.14 projects `IE1` to `IE3` and adds a new role requirements `RR3` to `IE3`. For modeling this query with XQBE and VXQ, one needs to add a target node `RR4` and the default binding for $(\mathtt{RR1}, \mathtt{RR4})$. In the case that `IE1` has further child SRCs and ILs, their default bindings also need to be added manually.

In conclusion, AQT is designed for AML data exchange by finding a balance between expressive power and the modeling complexity. Furthermore, in contrast to a dedicated graphical query language as XQBE and VXQ, the AML-based syntax of AQT is more intuitive for the AML users and allows creating queries with conventional AML editors.

9. Conclusions and Future Work

This thesis investigated the problem of semantic interoperability in AML-based engineering from two perspectives. The first part tackled the challenges of ontology building in the industrial context, and the second part developed QBE-like approaches for accessing and exchanging data stored in AML files. This chapter summarizes the thesis by revisiting the research questions and discussing future work.

9.1. Contributions

In Section 1.1, the motivation and the background of the work were presented. In particular, two research questions have been proposed that define the scope of the thesis.

RQ1: How to facilitate ontology building in industrial environments and incorporate domain experts into the modeling procedure?

This research question is addressed in the first part of the thesis as follows:

- Because creating sophisticated ontological concepts can be overwhelming for domain experts, Chapter 4 aimed at developing advanced machine learning algorithms for inducing OWL complex class expressions from labeled data. Section 4.1 studied the state-of-the-art learning algorithm CELOE and discussed its drawbacks in both theoretical and practical aspects. Section 4.2 presented the algorithm Rapid Restart Hill Climbing (RRHC) that tackles the problems of CELOE by traversing the search tree in a hill climbing fashion. For learning from AML data, Section 4.3 proposed the AML-specific refinement operator ρ^{aml} that reduces the search space by utilizing the syntactic constraints of the CAEX schema. Finally, the effectiveness of RRHC and ρ^{aml} is evaluated in Section 4.4. On the one hand, RRHC outperformed CELOE in 11 out of 16 learning problems for OWL and had a comparative result in the remaining 5 cases. On the other hand, ρ^{aml} showed superior performance in all learning problems for AML.
- The raw outputs of the concept learning algorithms are OWL complex class expressions, which can be unintuitive for domain experts. To make the learner more transparent, Chapter 5 introduced the AMLLEARNER framework that assists the user during concept induction. AMLLEARNER interacts with the user based on bidirectional communication via the AML Concept Model (ACM). ACMs build the front-end of the system by representing OWL complex class expressions in the AML syntax. Therefore, the results of the learner can be inspected and modified by the user. The user feedback, in turn, is exploited by the learner for improving the results. By studying the literature on interactive machine learning (IML) systems, Section 5.4 showed that AMLLEARNER supports the most desired interactive features.

RQ2: How to adapt database-based approaches for AML-based engineering so that they become AutomationML-centric while adhering to the foundations of database theory?

The motivation of RQ2 was driven by the recent advances from database research. The second part of the thesis provides an answer to RQ2 following the Query by Example (QBE) paradigm.

- Chapter 7 investigated the problem of AML data access. In contrast to the conventional approach that relies on the API of an AML engine, Section 7.2 introduced the AML Query Template (AQT) for describing the desired query using native AML models, and Section 7.3 presented algorithms for translating AQTs into XPath and XQuery programs. AQT has two design principles. First, it aims at domain experts who are familiar with AML but unskilled in programming. Second, AQT is inspired by the QBE approaches for XML data exchange and has a tree pattern based semantics. The study of essential requirements for querying AML showed that AQT has rich modeling features while being intuitive to AML users.
- Based on the results of Chapter 7, Chapter 8 extended AQT for exchanging AML data using the same methodology. First, a systematic analysis of the requirements for AML data exchange is conducted in Section 8.1. Second, a data exchange setting is developed for AML, which is based on the foundations of database theories. The theoretical study provided necessary notions for the design of the extension and a guideline for the implementation. Based on the formal setting, Section 8.3 presented the syntax and semantics extensions of AQT. Finally, algorithms for query translation are developed in Section 8.4. The comparison with the related work showed that AQT is more suitable for data exchange tasks in AML-based engineering.

9.2. Future Work

Despite the achievements as described above, future work in several aspects can be considered.

Better concept learning algorithms. First, the learning algorithms presented in Chapter 4 are based on top-down refinement operators. However, bottom-up approaches might be more effective to learn complex concepts from a small set of examples [39]. Moreover, bottom-up approaches do not necessarily require negative examples which can be more suitable for learning from engineering data. In the literature of learning in DL, mixed approaches that combine top-down and bottom-up refinements have also been proposed, such as YINGYANG [70] and DL-FOIL [73][74]. Future work following this direction can again utilize the structure properties of the CAEX schema for efficient space traversal.

Second, while the experiments of RRHC showed promising results, this thesis did not look into the “myopia” problem of hill climbing [103]. One reason is that the heuristic of RRHC has a dynamic nature, i.e., the score of a tree node changes after its concept is refined. In future work, it is worth investigating how the myopia problem affects RRHC for learning DL concepts, and techniques from Inductive Logic Programming (ILP) can be borrowed for improving the performance of RRHC. For example, *macros* were introduced for tackling the problem of non-discriminating relations [104]. A similar adaptation of the refinement operators can thus require the consumer in any quantification (filler of an object property) to be a specialization of the top concept $\top$.

Third, more sophisticated strategies for space traversing can be considered. In the field of ILP, randomized searching has been proposed [105][106]. In the same sequel, Mahsa et al. proposed a reinforcement learning (RL) based approach for concept learning in $\mathcal{EL}^{++}$ [107]. The core idea thereby, is to replace the deterministic top-down refinement with a stochastic sampling of hypotheses. To this end, the refinement operator is interpreted as actions, and a reward function is defined which has a similar effect to the heuristic. Future work in this direction includes widening the RL-based approach for more expressive languages, including OWL, and the investigation of better reward functions.

Finally, techniques from parallel computing can be adopted. In particular, the learning algorithms PARCEL [71] and SPARCEL [72] proposed by Tran et al. seem very promising because they are built on top of DL-Learner. In other words, the top-down learning algorithm RRHC and the refinement operator ρ^{aml} proposed in this thesis can be combined with the parallel techniques for potentially better learning performance.

Employing machine learning techniques for tackling various semantic interoperability problems. The first part of the thesis was completely based on the ontology language OWL. However, recent advances from the machine learning field showed appealing research directions based on different semantic formalisms, including learning associational rules from RDF triple stores or OWL ontologies [108][109][110], onto-relational learning in hybrid knowledge bases [111][112][113], and rule mining from knowledge graphs [114][115][116]. The objective of these approaches is to learn rules that reveal logical relations in the data, which can be used for achieving a higher level of semantic interoperability. However, different approaches are based on the underlying semantic foundation and have advantages or drawbacks regarding the expressiveness and performance. Therefore, future work in this direction may consider various formalisms for the semantic lifting of engineering data and choose the appropriate techniques for solving individual interoperability problems.

Increasing the expressive power of AQT and inducing tree pattern queries. As described in Section 7.4 and 8.5, the expressiveness of AQT can be improved by considering common arithmetic and String operations. These features are necessary for describing value processing relations [34]. Because XPath and XQuery provide dedicated notions and functions for the implementation of value processing, only a modeling approach is required for covering these features in AQT. On the one hand, an extension of the AML standard can be discussed in the scope of Best Practice Recommendations (BPR). On the other hand, the modeling approach proposed by Bihani et al. [51] can be adopted as an ad-hoc solution. Finally, a further step based on AQT is to employ machine learning techniques for query induction. For example, Cohen and Weiss studied the complexity of learning tree patterns from example graphs [117]. In this regard, one interesting question is whether AQTs for data access can be learned because ACM and AQT share a very similar structure. However, future work needs to investigate the complexity of learning the descendant axis, which could lead to performance issues during inference. Another related research direction is the reverse-engineering of

SPARQL queries [118][119], which can be seen as a more advanced solution to the induction of tree patterns.

Bibliography

[1] Prerna Bihani and Rainer Drath. Concept for AutomationML-based interoperability between multiple independent engineering tools without semantic harmonization: Experiences with AutomationML. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Limassol, September 2017. IEEE. ISBN 978-1-5090-6505-9. doi: 10.1109/ETFA.2017.8247586. URL `http://ieeexplore.ieee.org/document/8247586/`.

[2] Rainer Drath, Alexander Fay, and Mike Barth. Interoperabilität von Engineering-Werkzeugen. *at - Automatisierungstechnik*, 59(7):451–460, July 2011. ISSN 0178-2312. doi: 10.1524/auto.2011.0938. URL `http://www.degruyter.com/doi/10.1524/auto.2011.0938`.

[3] Fachbereich Engineering und Betrieb automatisierter Anlagen. VDI/VDE 3695 Blatt 4 - Engineering of industrial plants; Evaluation and optimization; Subject tools. *VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik*, 2010.

[4] Rainer Drath and Mike Barth. Concept for interoperability between independent engineering tools of heterogeneous disciplines. In *ETFA2011*, pages 1–8, Toulouse, France, September 2011. IEEE. ISBN 978-1-4577-0017-0. doi: 10.1109/ETFA.2011.6058975. URL `http://ieeexplore.ieee.org/document/6058975/`.

[5] Joachim Burlein and Matthias Rassl. Introducing AutomationML in a heterogeneous software tool landscape – a success story. In *5th AutomationML User Conference*, Sweden, 2018. AutomationML e.V.

[6] Rainer Drath, Michael John, Arndt Lüder, Johanna Pauly, Josef Prinz, Markus Rentschler, Matthias Riedl, Miriam Schleipen, Nicole Schmidt, Jannis Stecken, Anton Strahilov, and Bernhard Wally. AutomationML in a Nutshell. *AutomationML e.V.*, page 32, 2018. URL `http://www.unserebroschuere.de/automationml/WebView/`.

[7] Markus Rentschler and Rainer Drath. Vendor-Independent modeling and exchange of Fieldbus Topologies with AutomationML. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 956–963, Turin, September 2018. IEEE. ISBN 978-1-5386-7108-5. doi: 10.1109/ETFA.2018.8502630. URL `https://ieeexplore.ieee.org/document/8502630/`.

[8] Peter Adolphs, Heinz Bedenbender, Dagmar Dirzus, Martin Ehlich, Ulrich Epple, Martin Hankel, Roland Heidel, Michael Hoffmeister, Haimo Huhle, Bernd Kärcher, Heiko Koziolek, Reinhold Pichler, Stefan Pollmeier, Frank

Schewe, Armin Walter, Bernd Waser, and Martin Wollschlaeger. Reference Architecture Model Industrie 4.0 (RAMI4.0). *VDI/VDE, ZVEI*, 2015. URL https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2016/januar/GMA_Status_Report__Reference_Archtitecture_Model_Industrie_4.0__RAMI_4.0_/GMA-Status-Report-RAMI-40-July-2015.pdf.

[9] Implementation Strategy Industrie 4.0. *Plattform Industrie 4.0*, 2016.

[10] German Standardization Roadmap Industrie 4.0 – Version 3. 2018.

[11] Erich Barnstedt, Heinz Bedenbender, Meik Billmann, Birgit Boss, Erich Clauer, Michael Fritsche, Kai Garrels, Martin Hankel, Oliver Hillermeier, Michael Hoffmeister, Michael Jochem, Heiko Koziolek, Christoph Legat, Marco Mendes, Jörg Neidig, Manuel Sauer, Marc Schier, Michael Schmitt, Tizial Schröder, and Constantin Ziesche. Details of the Asset Administration Shell. Part 1 –The exchange of information between partners in the value chain of Industrie 4.0. 2018. doi: 10.13140/RG.2.2.18347.28968.

[12] Rainer Drath and Mike Barth. Concept for managing multiple semantics with AutomationML: Maturity level concept of semantic standardization. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, pages 1–8, Krakow, Poland, September 2012. IEEE. doi: 10.1109/ETFA.2012.6489538. URL http://ieeexplore.ieee.org/document/6489538/.

[13] Jacob Nilsson and Fredrik Sandin. Semantic Interoperability in Industry 4.0: Survey of Recent Developments and Outlook. In *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, pages 127–132, Porto, July 2018. IEEE. ISBN 978-1-5386-4829-2. doi: 10.1109/INDIN.2018.8471971. URL https://ieeexplore.ieee.org/document/8471971/.

[14] Lorenz Hundt. Draft Libraries for Whitepaper - Description of AutomationML Components, December 2019. URL https://www.automationml.org/o.red/uploads/dateien/1575989332-WP%20Compo%20-%20AutomationComponentLibrary_v1_0_0.zip.

[15] Irlán Grangel-González. *A Knowledge Graph Based Integration Approach for Industry 4.0.* PhD thesis, University Bonn, Boon, Germany, 2019.

[16] S. Runde, K. Güttel, and A. Fay. Transformation von CAEX-Anlagenplanungsdaten in OWL: Eine Anwendung von Technologien des Semantic Web. In *Automation 2009, Der Automatisierungskongress in Deutschland, VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik (GMA)*, pages 175–178, June 2009.

[17] S. Runde, A. Fay, and S. Böhm. Konvertierung von OWL-Planungsergebnissen nach CAEX. In *Automation 2010, Der 11. Branchentreff der Mess- und Automatisierungstechnik, VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik (GMA)*, pages 1–12, June 2010.

[18] Lisa Abele, Christoph Legat, Stephan Grimm, and Andreas W. Muller. Ontology-based validation of plant models. In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 236–241, Bochum, Germany, July 2013. IEEE. ISBN 978-1-4799-0752-6. doi: 10.1109/INDIN.2013.6622888. URL `http://ieeexplore.ieee.org/document/6622888/`.

[19] Anders Björkelund, Jacek Malec, Klas Nilsson, and Pierre Nugues. Knowledge and Skill Representations for Robotized Production. *IFAC Proceedings Volumes*, 44(1): 8999–9004, January 2011. ISSN 14746670. doi: 10.3182/20110828-6-IT-1002.01053. URL `https://linkinghub.elsevier.com/retrieve/pii/S1474667016450561`.

[20] Olga Kovalenko, Manuel Wimmer, Marta Sabou, Arndt Luder, Fajar J. Ekaputra, and Stefan Biffl. Modeling AutomationML: Semantic Web technologies vs. Model-Driven Engineering. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–4, Luxembourg, Luxembourg, September 2015. IEEE. ISBN 978-1-4673-7929-8. doi: 10.1109/ETFA.2015.7301643. URL `http://ieeexplore.ieee.org/document/7301643/`.

[21] Yingbing Hua, Stefan Zander, Mirko Bordignon, and Bjorn Hein. From AutomationML to ROS: A model-driven approach for software engineering of industrial robotics using ontological reasoning. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Berlin, Germany, September 2016. IEEE. ISBN 978-1-5090-1314-2. doi: 10.1109/ETFA.2016.7733579. URL `http://ieeexplore.ieee.org/document/7733579/`.

[22] Stefan Biffl and Marta Sabou. *Semantic Web Technologies for Intelligent Engineering Applications*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3-319-41488-7 978-3-319-41488-1.

[23] Evgeny Kharlamov, Nina Solomakhina, Özgür Lütfü Özçep, Dmitriy Zheleznyakov, Thomas Hubauer, Steffen Lamparter, Mikhail Roshchin, Ahmet Soylu, and Stuart Watson. How Semantic Technologies Can Enhance Data Access at Siemens Energy. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble, editors, *The Semantic Web – ISWC 2014*, pages 601–619, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11964-9.

[24] Fajar J Ekaputra, Marta Sabou, Estefanía Serral, Elmar Kiesling, and Stefan Biffl. Ontology-Based Data Integration in Multi-Disciplinary Engineering Environments: A Review. 4(1):26, 2017.

[25] Marta Sabou, Stefan Biffl, Alfred Einfalt, Lukas Krammer, Wolfgang Kastner, and Fajar J. Ekaputra. Semantics for Cyber-Physical Systems: A Cross-Domain Perspective. *Semantic Web*, 2019.

[26] C. Hildebrandt, S. Torsleff, B. Caesar, and A. Fay. Ontology Building for Cyber-Physical Systems: A domain expert-centric approach. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*,

pages 1079–1086, Munich, August 2018. IEEE. ISBN 978-1-5386-3593-3. doi: 10.1109/COASE.2018.8560465. URL https://ieeexplore.ieee.org/document/8560465/.

[27] Evgeny Kharlamov, Bernardo Cuenca Grau, Ernesto Jiménez-Ruiz, Steffen Lamparter, Gulnar Mehdi, Martin Ringsquandl, Yavor Nenov, Stephan Grimm, Mikhail Roshchin, and Ian Horrocks. Capturing Industrial Information Models with Ontologies and Constraints. In Paul T. Groth, Elena Simperl, Alasdair J. G. Gray, Marta Sabou, Markus Krötzsch, Freddy Lécué, Fabian Flöck, and Yolanda Gil, editors, *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part II*, volume 9982 of *Lecture Notes in Computer Science*, pages 325–343, 2016. doi: 10.1007/978-3-319-46547-0_30. URL https://doi.org/10.1007/978-3-319-46547-0_30.

[28] Pablo Barceló. Logical foundations of relational data exchange. *ACM SIGMOD Record*, 38(1):49, June 2009. ISSN 01635808. doi: 10.1145/1558334.1558341. URL http://portal.acm.org/citation.cfm?doid=1558334.1558341.

[29] Ronald Fagin, Laura M. Haas, Mauricio Hernández, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema Mapping Creation and Data Exchange. In Alexander T. Borgida, Vinay K. Chaudhri, Paolo Giorgini, and Eric S. Yu, editors, *Conceptual Modeling: Foundations and Applications*, volume 5600, pages 198–236. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-02462-7 978-3-642-02463-4. doi: 10.1007/978-3-642-02463-4_12. URL http://link.springer.com/10.1007/978-3-642-02463-4_12.

[30] Iovka Boneva, Angela Bonifati, and Radu Ciucanu. Graph Data Exchange with Target Constraints. In *EDBT/ICDT Workshops - Querying Graph Structured Data (GraphQ)*, pages 171–176, Bruxelles, Belgium, 2015. URL https://hal.inria.fr/hal-01095838.

[31] Moshé M. Zloof. Query by Example. In *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*, AFIPS '75, pages 431–438, New York, NY, USA, 1975. ACM. doi: 10.1145/1499949.1500034. URL http://doi.acm.org/10.1145/1499949.1500034. event-place: Anaheim, California.

[32] Daniele Braga, Alessandro Campi, and Stefano Ceri. XQBE (XQuery By Example): A visual interface to the standard XML query language. *ACM Transactions on Database Systems*, 30(2):398–443, June 2005. ISSN 03625915. doi: 10.1145/1071610.1071613. URL http://portal.acm.org/citation.cfm?doid=1071610.1071613.

[33] Ryan H. Choi and Raymond K. Wong. VXQ: A visual query language for XML data. *Information Systems Frontiers*, 17(4):961–981, August 2015. ISSN 1387-3326, 1572-9419. doi: 10.1007/s10796-013-9480-3. URL http://link.springer.com/10.1007/s10796-013-9480-3.

[34] Olga Kovalenko and Jérôme Euzenat. Semantic Matching of Engineering Data Structures. In Stefan Biffl and Marta Sabou, editors, *Semantic Web Technologies*

for Intelligent Engineering Applications, pages 137–157. Springer International Publishing, Cham, 2016. ISBN 978-3-319-41490-4. doi: 10.1007/978-3-319-41490-4_6. URL https://doi.org/10.1007/978-3-319-41490-4_6.

[35] Bogdan Alexe, Laura Chiticariu, Renee J. Miller, and Wang-Chiew Tan. Muse: Mapping Understanding and deSign by Example. In *2008 IEEE 24th International Conference on Data Engineering*, pages 10–19, Cancun, Mexico, April 2008. IEEE. ISBN 978-1-4244-1836-7 978-1-4244-1837-4. doi: 10.1109/ICDE.2008.4497409. URL http://ieeexplore.ieee.org/document/4497409/.

[36] A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, and G. Summa. Schema Mapping Verification: The Spicy Way. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '08, pages 85–96, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-926-5. doi: 10.1145/1353343.1353358. URL http://doi.acm.org/10.1145/1353343.1353358. event-place: Nantes, France.

[37] Philip Bohannon, Eiman Elnahrawy, Wenfei Fan, and Michael Flaster. Putting Context into Schema Matching. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 307–318. VLDB Endowment, 2006. URL http://dl.acm.org/citation.cfm?id=1182635.1164155. event-place: Seoul, Korea.

[38] Guohui Ding and Guoren Wang. Discovering Implicit Categorical Semantics for Schema Matching. In Jeffrey Xu Yu, Myoung Ho Kim, and Rainer Unland, editors, *Database Systems for Advanced Applications*, volume 6588, pages 179–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-20151-6 978-3-642-20152-3. doi: 10.1007/978-3-642-20152-3_14. URL http://link.springer.com/10.1007/978-3-642-20152-3_14.

[39] Jens Lehmann. *Learning OWL Class Expressions*. PhD thesis, Universität Leipzig, 2009.

[40] Robie Jonathan, Dyck Michael, and Spiegel Josh. XML Path Language (XPath) 3.1, 2017. URL https://www.w3.org/TR/xpath-31/.

[41] Robie Jonathan, Dyck Michael, and Spiegel Josh. XQuery 3.1: An XML Query Language, 2017. URL https://www.w3.org/TR/xquery-31/.

[42] Rainer Drath, Arndt Luder, Jorn Peschke, and Lorenz Hundt. AutomationML - the glue for seamless automation engineering. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pages 616–623, Hamburg, Germany, September 2008. IEEE. ISBN 978-1-4244-1505-2. doi: 10.1109/ETFA.2008.4638461. URL http://ieeexplore.ieee.org/document/4638461/.

[43] Miriam Schleipen, Rainer Drath, and Olaf Sauer. The system-independent data exchange format CAEX for supporting an automatic configuration of a production monitoring and control system. In *2008 IEEE International Symposium on Industrial Electronics*, pages 1786–1791, Cambridge, UK, June 2008. IEEE. ISBN 978-1-4244-1665-3. doi: 10.1109/ISIE.2008.4676932.

[44] OPC Unified Architecture Information Model for AutomationML. *AutomationML consortium, OPC Foundation*, 2016. URL `https://www.automationml.org/o.red/uploads/dateien/1485865685-WP_OPCUAforAutomationML_V1.0.0.zip`.

[45] Olaf Graeser, Lorenz Hundt, Michael John, Gerald Lobermeier, Arndt Lüder, Stefan Mülhens, Nikolaus Ondracek, Mario Thron, and Josef Schmelter. AutomationML and eCl@ss Integration. *Common Working Group of AutomationML e.V and eCl@ss e.V.*, page 57, 2017.

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

[47] Stefan Biffl, Olga Kovalenko, Arndt Luder, Nicole Schmidt, and Ronald Rosendahl. Semantic mapping support in AutomationML. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–4, Barcelona, Spain, September 2014. IEEE. ISBN 978-1-4799-4845-1. doi: 10.1109/ETFA.2014.7005276. URL `http://ieeexplore.ieee.org/document/7005276/`.

[48] Y. Hua and B. Hein. Concept Learning in AutomationML with Formal Semantics and Inductive Logic Programming. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1542–1547, August 2018. doi: 10.1109/COASE.2018.8560541.

[49] Lorenz Hundt and Josef Prinz. AutomationML-Datenaustausch. *SPS Magazin*, 4, 2013.

[50] Pouria G. Bigvand, Alexander Fay, Rainer Drath, and Pablo Rodriguez Carrion. Concept and development of a semantic based data hub between process design and automation system engineering tools. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Berlin, Germany, September 2016. IEEE. ISBN 978-1-5090-1314-2. doi: 10.1109/ETFA.2016.7733734. URL `http://ieeexplore.ieee.org/document/7733734/`.

[51] Prerna Bihani, Rainer Drath, and Aniket Kadam. Towards Meaningful Interoperability for Heterogeneous Engineering Tools via AutomationML. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1286–1290, Zaragoza, Spain, September 2019. IEEE. ISBN 978-1-72810-303-7. doi: 10.1109/ETFA.2019.8869532. URL `https://ieeexplore.ieee.org/document/8869532/`.

[52] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, page 4, May 2001.

[53] Marta Sabou, Olga Kovalenko, Fajar Juang Ekaputra, and Stefan Biffl. Semantic Web Solutions in Engineering. In Stefan Biffl and Marta Sabou, editors, *Semantic Web Technologies for Intelligent Engineering Applications*, pages 281–296. Springer International Publishing, Cham, 2016. ISBN 978-3-319-41488-1 978-3-319-41490-4. doi: 10.1007/978-3-319-41490-4_11. URL `http://link.springer.com/10.1007/978-3-319-41490-4_11`.

[54] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of semantic web technologies*. Chapman & Hall/CRC textbooks in computing. CRC Press, Boca Raton, Fla. London New York, 2010. ISBN 978-1-4200-9050-5. OCLC: 845661932.

[55] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: the making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26, December 2003. ISSN 15708268. doi: 10.1016/j.websem.2003.07.001. URL `https://linkinghub.elsevier.com/retrieve/pii/S1570826803000027`.

[56] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003. ISBN 0-521-78176-0.

[57] M. Ross Quillian. Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12(5):410–430, 1967. doi: 10.1002/bs.3830120511. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/bs.3830120511`.

[58] Marvin Minsky. A Framework for Representing Knowledge. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1974.

[59] Manfred Schmidt-Schaubß and Gert Smolka. Attributive Concept Descriptions with Complements. *Artif. Intell.*, 48(1):1–26, 1991. ISSN 0004-3702. doi: 10.1016/0004-3702(91)90078-X. URL `http://dx.doi.org/10.1016/0004-3702(91)90078-X`.

[60] Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017. doi: 10.1017/9781139025355.

[61] J. Persson, A. Gallois, A. Bjoerkelund, L. Hafdell, M. Haage, J. Malec, K. Nilsson, and P. Nugues. A Knowledge Integration Framework for Robotics. In *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, pages 1–8, June 2010.

[62] Marta Sabou, Olga Kovalenko, and Petr Novák. Semantic Modelling and Acquisition of Engineering Knowledge. In Stefan Biffl and Marta Sabou, editors, *Semantic Web Technologies for Intelligent Engineering Applications*, pages 105–136. Springer International Publishing, Cham, 2016. ISBN 978-3-319-41490-4. doi: 10.1007/978-3-319-41490-4_5. URL `https://doi.org/10.1007/978-3-319-41490-4_5`.

[63] Jens Lehmann and Pascal Hitzler. Concept learning in description logics using refinement operators. *Machine Learning*, 78(1-2):203–250, January 2010. ISSN 0885-6125, 1573-0565. doi: 10.1007/s10994-009-5146-2. URL `http://link.springer.com/10.1007/s10994-009-5146-2`.

[64] Lorenz Bühmann, Jens Lehmann, Patrick Westphal, and Simon Bin. DL-Learner Structured Machine Learning on Semantic Web Data. In *Companion of the*

The Web Conference 2018 on The Web Conference 2018 - WWW '18, pages 467–471, Lyon, France, 2018. ACM Press. ISBN 978-1-4503-5640-4. doi: 10.1145/3184558.3186235. URL `http://dl.acm.org/citation.cfm?doid=3184558.3186235`.

[65] Jens Lehmann, Sören Auer, Lorenz Bühmann, and Sebastian Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, 9(1): 71 – 81, 2011. ISSN 1570-8268. doi: https://doi.org/10.1016/j.websem.2011.01.001. URL `http://www.sciencedirect.com/science/article/pii/S1570826811000023`.

[66] Tristan Cazenave. Iterative Widening. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'01, pages 523–528, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-812-5 978-1-55860-812-2. URL `http://dl.acm.org/citation.cfm?id=1642090.1642162`. event-place: Seattle, WA, USA.

[67] Liviu Badea and Shan Hwei Nienhuys-Cheng. A Refinement Operator for Description Logics. In G. Goos, J. Hartmanis, J. van Leeuwen, James Cussens, and Alan Frisch, editors, *Inductive Logic Programming*, volume 1866, pages 40–59. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. ISBN 978-3-540-67795-6 978-3-540-44960-7. doi: 10.1007/3-540-44960-4_3. URL `http://link.springer.com/10.1007/3-540-44960-4_3`.

[68] N. Fanizzi, S. Ferilli, L. Iannone, I. Palmisano, and G. Semeraro. Downward Refinement in the ALN Description Logic. In *Fourth International Conference on Hybrid Intelligent Systems (HIS'04)*, pages 68–73, Kitakyushu, Japan, 2004. IEEE. ISBN 978-0-7695-2291-3. doi: 10.1109/ICHIS.2004.39. URL `http://ieeexplore.ieee.org/document/1409983/`.

[69] Floriana Esposito, Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro. Knowledge-Intensive Induction of Terminologies from Metadata. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *The Semantic Web – ISWC 2004*, volume 3298, pages 441–455. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-23798-3 978-3-540-30475-3. doi: 10.1007/978-3-540-30475-3_31. URL `http://link.springer.com/10.1007/978-3-540-30475-3_31`.

[70] Luigi Iannone, Ignazio Palmisano, and Nicola Fanizzi. An algorithm based on counterfactuals for concept learning in the Semantic Web. *Applied Intelligence*, 26 (2):139–159, March 2007. ISSN 0924-669X, 1573-7497. doi: 10.1007/s10489-006-0011-5. URL `http://link.springer.com/10.1007/s10489-006-0011-5`.

[71] An C. Tran, Jens Dietrich, Hans W. Guesgen, and Stephen Marsland. Two-way Parallel Class Expression Learning. In Steven C. H. Hoi and Wray Buntine, editors, *Proceedings of the Asian Conference on Machine Learning*, volume 25 of *Proceedings of Machine Learning Research*, pages 443–458,

Singapore Management University, Singapore, November 2012. PMLR. URL `http://proceedings.mlr.press/v25/tran12c.html`.

[72] An C. Tran, Jens Dietrich, Hans W. Guesgen, and Stephen Marsl and. Parallel Symmetric Class Expression Learning. *Journal of Machine Learning Research*, 18 (64):1–34, 2017. URL `http://jmlr.org/papers/v18/14-317.html`.

[73] Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. DL-FOIL Concept Learning in Description Logics. In Filip Železný and Nada Lavrač, editors, *Inductive Logic Programming*, volume 5194, pages 107–121. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-85927-7 978-3-540-85928-4. doi: 10.1007/978-3-540-85928-4_12. URL `http://link.springer.com/10.1007/978-3-540-85928-4_12`.

[74] Nicola Fanizzi, Giuseppe Rizzo, Claudia d'Amato, and Floriana Esposito. DLFoil: Class Expression Learning Revisited. In Catherine Faron Zucker, Chiara Ghidini, Amedeo Napoli, and Yannick Toussaint, editors, *Knowledge Engineering and Knowledge Management*, volume 11313, pages 98–113. Springer International Publishing, Cham, 2018. ISBN 978-3-030-03666-9 978-3-030-03667-6. doi: 10.1007/978-3-030-03667-6_7. URL `http://link.springer.com/10.1007/978-3-030-03667-6_7`.

[75] J. R. Quinlan and R. M. Cameron-Jones. Induction of logic programs: FOIL and related systems. *New Generation Computing*, 13(3-4):287–312, December 1995. ISSN 0288-3635, 1882-7055. doi: 10.1007/BF03037228. URL `http://link.springer.com/10.1007/BF03037228`.

[76] Antonio De Nicola and Michele Missikoff. A lightweight methodology for rapid ontology engineering. *Communications of the ACM*, 59(3):79–86, February 2016. ISSN 00010782. doi: 10.1145/2818359. URL `http://dl.acm.org/citation.cfm?doid=2897191.2818359`.

[77] Jens Lehmann and Pascal Hitzler. Foundations of Refinement Operators for Description Logics. In Hendrik Blockeel, Jan Ramon, Jude Shavlik, and Prasad Tadepalli, editors, *Inductive Logic Programming*, volume 4894, pages 161–174. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-78468-5 978-3-540-78469-2. doi: 10.1007/978-3-540-78469-2_18. URL `http://link.springer.com/10.1007/978-3-540-78469-2_18`.

[78] Lorenz Bühmann, Jens Lehmann, and Patrick Westphal. DL-Learner—A framework for inductive learning on the Semantic Web. *Journal of Web Semantics*, 39:15–24, August 2016. ISSN 15708268. doi: 10.1016/j.websem.2016.06.001. URL `https://linkinghub.elsevier.com/retrieve/pii/S157082681630018X`.

[79] Motik Boris, F. Patel-Schneider Peter, and Cuenca Grau Berardo. OWL 2 Web Ontology Language Direct Semantics (Second Edition), 2012. URL `https://www.w3.org/TR/owl2-direct-semantics/`.

[80] Lisa Abele and Stephan Grimm. Knowledge-based integration of industrial plant models. In *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, pages 4392–4397, Vienna, Austria, November 2013. IEEE.

ISBN 978-1-4799-0224-8. doi: 10.1109/IECON.2013.6699842. URL `http://ieeexplore.ieee.org/document/6699842/`.

[81] Marta Sabou, Fajar Ekaputra, Olga Kovalenko, and Stefan Biffl. Supporting the engineering of cyber-physical production systems with the AutomationML analyzer. In *2016 1st International Workshop on Cyber-Physical Production Systems (CPPS)*, pages 1–8, Vienna, Austria, April 2016. IEEE. ISBN 978-1-5090-1156-8. doi: 10.1109/CPPS.2016.7483919. URL `http://ieeexplore.ieee.org/document/7483919/`.

[82] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. Power to the People: The Role of Humans in Interactive Machine Learning. *AI Magazine*, 35(4):105, December 2014. ISSN 0738-4602, 0738-4602. doi: 10.1609/aimag.v35i4.2513. URL `https://aaai.org/ojs/index.php/aimagazine/article/view/2513`.

[83] John J. Dudley and Per Ola Kristensson. A Review of User Interface Design for Interactive Machine Learning. *ACM Transactions on Interactive Intelligent Systems*, 8(2):1–37, June 2018. ISSN 21606455. doi: 10.1145/3185517. URL `http://dl.acm.org/citation.cfm?doid=3232718.3185517`.

[84] Markus Krötzsch. *Description Logic Rules*. PhD Thesis, KIT, 2010.

[85] Manuel Wimmer and Alexandra Mazak. From AutomationML to AutomationQL: A By-Example Query Language for CPPS Engineering Models. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1394–1399, Munich, August 2018. IEEE. ISBN 978-1-5386-3593-3. doi: 10.1109/COASE.2018.8560448. URL `https://ieeexplore.ieee.org/document/8560448/`.

[86] Gerome Miklau and Dan Suciu. Containment and equivalence for a fragment of XPath. *Journal of the ACM*, 51(1):2–45, January 2004. ISSN 00045411. doi: 10.1145/962446.962448. URL `http://portal.acm.org/citation.cfm?doid=962446.962448`.

[87] Laks V. S. Lakshmanan, Hui Wang, and Zheng Zhao. Answering Tree Pattern Queries Using Views. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB ’06, pages 571–582. VLDB Endowment, 2006. event-place: Seoul, Korea.

[88] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0, November 1999. URL `https://www.w3.org/TR/1999/REC-xpath-19991116/`.

[89] Georg Gottlob, Christoph Koch, and Reinhard Pichler. The complexity of XPath query evaluation. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS ’03*, pages 179–190, San Diego, California, 2003. ACM Press. ISBN 978-1-58113-670-8. doi: 10.1145/773153.773171. URL `http://portal.acm.org/citation.cfm?doid=773153.773171`.

[90] Zhiyuan Chen, H.V. Jagadish, Flip Korn, N. Koudas, S. Muthukrishnan, R. Ng, and D. Srivastava. Counting twig matches in a tree. In *Proceedings 17th International Conference on Data Engineering*, pages 595–604, Heidelberg, Germany, 2001. IEEE Comput. Soc. ISBN 978-0-7695-1001-9. doi: 10.1109/ICDE.2001.914874. URL `http://ieeexplore.ieee.org/document/914874/`.

[91] Sihem Amer-Yahia, SungRan Cho, Laks V. S. Lakshmanan, and Divesh Srivastava. Minimization of Tree Pattern Queries. *SIGMOD Rec.*, 30(2):497–508, 2001. ISSN 0163-5808. doi: 10.1145/376284.375730. URL `https://doi.org/10.1145/376284.375730`.

[92] H. V. Jagadish, Laks V. S. Lakshmanan, Divesh Srivastava, and Keith Thompson. TAX: A Tree Algebra for XML. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Giorgio Ghelli, and Gösta Grahne, editors, *Database Programming Languages*, volume 2397, pages 149–164. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN 978-3-540-44080-2 978-3-540-46093-0. doi: 10.1007/3-540-46093-4_9. URL `http://link.springer.com/10.1007/3-540-46093-4_9`.

[93] Junhu Wang, Jiang Li, and Jeffrey Xu Yu. Answering tree pattern queries using views: a revisit. In *Proceedings of the 14th International Conference on Extending Database Technology - EDBT/ICDT '11*, page 153, Uppsala, Sweden, 2011. ACM Press. ISBN 978-1-4503-0528-0. doi: 10.1145/1951365.1951386. URL `http://portal.acm.org/citation.cfm?doid=1951365.1951386`.

[94] Renée J. Miller, Mauricio A. Hernández, Laura M. Haas, Lingling Yan, C. T. Howard Ho, Ronald Fagin, and Lucian Popa. The Clio project: managing heterogeneity. *ACM SIGMOD Record*, 30(1):78–83, March 2001. ISSN 01635808. doi: 10.1145/373626.373713. URL `http://portal.acm.org/citation.cfm?doid=373626.373713`.

[95] Marcelo Arenas and Leonid Libkin. XML data exchange: Consistency and query answering. *Journal of the ACM*, 55(2):1–72, May 2008. ISSN 00045411. doi: 10.1145/1346330.1346332. URL `http://portal.acm.org/citation.cfm?doid=1346330.1346332`.

[96] Shun'ichi Amano, Claire David, Leonid Libkin, and Filip Murlak. XML Schema Mappings: Data Exchange and Metadata Management. *Journal of the ACM*, 61 (2):1–48, April 2014. ISSN 00045411. doi: 10.1145/2590773. URL `http://dl.acm.org/citation.cfm?doid=2605175.2590773`.

[97] Ariel Fuxman, Mauricio A. Hernandez, Howard Ho, Renee J. Miller, Paolo Papotti, and Lucian Popa. Nested Mappings: Schema Mapping Reloaded. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB '06, pages 67–78. VLDB Endowment, 2006. event-place: Seoul, Korea.

[98] Tanja Mayerhofer, Manuel Wimmer, Luca Berardinelli, and Rainer Drath. A Model-Driven Engineering Workbench for CAEX Supporting Language Customization and Evolution. *IEEE Transactions on Industrial Informatics*, 14(6):2770–2779, June 2018. ISSN 1551-3203, 1941-0050. doi: 10.1109/TII.2017.2786780. URL `https://ieeexplore.ieee.org/document/8239624/`.

[99] Laks V. S. Lakshmanan. XML Tree Pattern, XML Twig Query. In LING LIU and M. TAMER ÖZSU, editors, *Encyclopedia of Database Systems*, pages 3637–3640. Springer US, Boston, MA, 2009. ISBN 978-0-387-39940-9. doi: 10.1007/978-0-387-39940-9_797. URL `https://doi.org/10.1007/978-0-387-39940-9_797`.

[100] Rainer Drath. Bäumchen wechsle Dich - Tücken beim automatischen Abgleich hierarchischer Objektstrukturen. *Softwaretechnik-Trends*, 26(4), 2006. URL `http://pi.informatik.uni-siegen.de/stt/26_4/01_Fachgruppenberichte/ORA2006/04_drath-final.pdf`.

[101] François Scharffe. *Correspondence Patterns Representation*. Dissertation, Science and Physics of the University of Innsbruck, Innsbruck, March 2009.

[102] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2nd edition, 2013. ISBN 978-3-642-38720-3. URL `http://book.ontologymatching.org`.

[103] Lourdes Peña Castillo and Stefan Wrobel. A comparative study on methods for reducing myopia of hill-climbing search in multirelational learning. In *Twenty-first international conference on Machine learning - ICML '04*, page 19, Banff, Alberta, Canada, 2004. ACM Press. doi: 10.1145/1015330.1015334. URL `http://portal.acm.org/citation.cfm?doid=1015330.1015334`.

[104] Lourdes Pena Castillo and Stefan Wrobel. Macro-Operators in Multirelational Learning: A Search-Space Reduction Technique. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Machine Learning: ECML 2002*, volume 2430, pages 357–368. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN 978-3-540-44036-9 978-3-540-36755-0. doi: 10.1007/3-540-36755-1_30. URL `http://link.springer.com/10.1007/3-540-36755-1_30`.

[105] Filip Železný, Ashwin Srinivasan, and David Page. A Monte Carlo Study of Randomised Restarted Search in ILP. In Rui Camacho, Ross King, and Ashwin Srinivasan, editors, *Inductive Logic Programming*, pages 341–358, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-30109-7.

[106] Filip Železný, Ashwin Srinivasan, and C. David Page. Randomised restarted search in ILP. *Machine Learning*, 64(1-3):183–208, September 2006. ISSN 0885-6125, 1573-0565. doi: 10.1007/s10994-006-7733-9. URL `http://link.springer.com/10.1007/s10994-006-7733-9`.

[107] Mahsa Chitsaz, Kewen Wang, Michael Blumenstein, and Guilin Qi. Concept Learning for EL++ by Refinement and Reinforcement. In *Proceedings of the 12th Pacific Rim International Conference on Trends in Artificial Intelligence*, PRICAI'12, pages 15–26, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-32694-3. doi: 10.1007/978-3-642-32695-0_4. URL `https://doi.org/10.1007/978-3-642-32695-0_4`. event-place: Kuching, Malaysia.

[108] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World*

Wide Web - WWW '13, pages 413–422, Rio de Janeiro, Brazil, 2013. ACM Press. ISBN 978-1-4503-2035-1. doi: 10.1145/2488388.2488425. URL `http://dl.acm.org/citation.cfm?doid=2488388.2488425`.

[109] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast rule mining in ontological knowledge bases with AMIE+. *The VLDB Journal*, 24(6): 707–730, December 2015. ISSN 0949-877X. doi: 10.1007/s00778-015-0394-1. URL `https://doi.org/10.1007/s00778-015-0394-1`.

[110] Claudia d'Amato, Steffen Staab, Andrea G. B. Tettamanzi, Tran Duc Minh, and Fabien Gandon. Ontology enrichment by discovering multi-relational association rules from ontological knowledge bases. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing - SAC '16*, pages 333–338, Pisa, Italy, 2016. ACM Press. ISBN 978-1-4503-3739-7. doi: 10.1145/2851613.2851842. URL `http://dl.acm.org/citation.cfm?doid=2851613.2851842`.

[111] Francesca A. Lisi. Principles of Inductive Reasoning on the Semantic Web: A Framework for Learning in AL-Log. In François Fages and Sylvain Soliman, editors, *Principles and Practice of Semantic Web Reasoning*, pages 118–132, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-32028-9.

[112] Francesca A. Lisi. Inductive Logic Programming in Databases: from Datalog to DL+log. *arXiv:1003.2586 [cs]*, March 2010. URL `http://arxiv.org/abs/1003.2586`. arXiv: 1003.2586.

[113] Francesca A. Lisi. AL-QuIn: An Onto-Relational Learning System for Semantic Web Mining. *International Journal on Semantic Web and Information Systems*, 7(3):1–22, July 2011. ISSN 1552-6283, 1552-6291. doi: 10.4018/jswis.2011070101. URL `http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jswis.2011070101`.

[114] Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. Completeness-Aware Rule Learning from Knowledge Graphs. In Claudia d'Amato, Miriam Fernandez, Valentina Tamma, Freddy Lecue, Philippe Cudré-Mauroux, Juan Sequeda, Christoph Lange, and Jeff Heflin, editors, *The Semantic Web – ISWC 2017*, pages 507–525, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68288-4.

[115] Vinh Thinh Ho, Daria Stepanova, Mohamed H. Gad-Elrab, Evgeny Kharlamov, and Gerhard Weikum. Rule Learning from Knowledge Graphs Guided by Embedding Models. In Denny Vrandečić, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl, editors, *The Semantic Web – ISWC 2018*, pages 72–90, Cham, 2018. Springer International Publishing. ISBN 978-3-030-00671-6.

[116] Daria Stepanova, Mohamed H. Gad-Elrab, and Vinh Thinh Ho. Rule Induction and Reasoning over Knowledge Graphs. In Claudia d'Amato and Martin Theobald, editors, *Reasoning Web. Learning, Uncertainty, Streaming, and Scalability: 14th International Summer School 2018, Esch-sur-Alzette, Luxembourg, September 22–26, 2018, Tutorial Lectures*, pages 142–172. Springer International Publishing, Cham, 2018. ISBN 978-3-030-00338-8. doi: 10.1007/978-3-030-00338-8_6. URL `https://doi.org/10.1007/978-3-030-00338-8_6`.

[117] Sara Cohen and Yaacov Y. Weiss. Learning Tree Patterns from Example Graphs. In Marcelo Arenas and Martín Ugarte, editors, *18th International Conference on Database Theory (ICDT 2015)*, volume 31 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 127–143, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-79-8. doi: 10.4230/LIPIcs.ICDT.2015.127. URL `http://drops.dagstuhl.de/opus/volltexte/2015/4981`. ISSN: 1868-8969.

[118] Gonzalo Diaz, Marcelo Arenas, and Michael Benedikt. SPARQLByE: Querying RDF Data by Example. *Proc. VLDB Endow.*, 9(13):1533–1536, 2016. ISSN 2150-8097. doi: 10.14778/3007263.3007302. URL `https://doi.org/10.14778/3007263.3007302`. Publisher: VLDB Endowment.

[119] Marcelo Arenas, Gonzalo I. Diaz, and Egor V. Kostylev. Reverse Engineering SPARQL Queries. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 239–249, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4143-1. doi: 10.1145/2872427.2882989. URL `https://doi.org/10.1145/2872427.2882989`. event-place: Montréal, Québec, Canada.

Appendix

A. Publications

[1] **Hua, Y.**, Hein, B. (2020). *AQT - A Query Template for AutomationML*. IEEE Transaction on Industrial Informatics, April 2020. doi: 10.1109/TII.2020.2989125.

[2] **Hua, Y.**, Hein, B. (2019). *Interactive Learning Engineering Concepts in AutomationML*. 24st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2019). Zaragoza.

[3] **Hua, Y.**, Hein, B. (2019). *Interpreting OWL Complex Classes in AutomationML based on Bidirectional Translation*. 24th IEEE Conference on Emerging Technologies and Factory Automation. Zaragoza.

[4] **Hua, Y.**, Hein, B. (2019). *Rapid Restart Hill Climbing for Learning Description Logic Concepts*. 29th International Conference on Inductive Logic Programming. Plovdiv.

[5] **Hua, Y.**, Hein, B. (2018). *Concept Learning in AutomationML with Formal Semantics and Inductive Logic Programming*. IEEE 14th International Conference on Automation Science and Engineering (CASE). München.

[6] **Hua, Y.**, Hein, B. (2018). *Concept Learning in Engineering based on Refinement Operator*. 28th International Conference on Inductive Logic Programming. Ferrara.

[7] **Hua, Y.**, Mende, M., Hein, B. (2017). *Modulare und Wandlungsfähige Robotersysteme*. Industrie 4.0 Management, 6, 33-37.

[8] **Hua, Y.**, Zander, S., Bordignon, M., Hein, B. (2016). *From AutomationML to ROS: A model-driven approach for software engineering of industrial robotics using ontological reasoning*. IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA). Berlin: IEEE.

[9] Reiser, U., Müller, U., Ludwig, M., Lüdtke, M., **Hua, Y.** (2018). *ReApp–Wiederverwendbare Roboterapplikationen für flexible Roboteranlagen*. In Zukunft der Arbeit–Eine praxisnahe Betrachtung (pp. 133-146). Berlin, Heidelberg: Springer Vieweg.

[10] Schleipen, M., Henßen, R., Schmidt, N., D'Agostino, N., **Hua, Y.** (2017). *AutomationML auf höheren Automatisierungsebenen - Eine Auswahl relevanter Anwendungsfälle*. Leitkongress der Mess- und Automatisierungstechnik. Baden-Baden.

[11] Zander, S., **Hua, Y.** (2016). *Utilizing ontological classification systems and reasoning for cyber-physical systems*. Karlsruhe Service Summit Research Workshop. Karlsruhe.

[12] Zander, S., Ahmed, N., **Hua, Y.** (2016). *Empowering the model-driven engineering of robotic applications using ontological semantics and reasoning*. International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management.

List of Figures

List of Tables

Listings

List of Algorithms