Performance of Paradigms for Storing and Querying Multidisciplinary Engineering Models

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering / Internet Computing

eingereicht von

Michael Wapp, BSc
Matrikelnummer 01325702

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao. Univ.Prof. Dr. Stefan Biffl
Mitwirkung: Dipl.-Ing. Kristof Meixner, BSc

Wien, 2. Dezember 2019

Michael Wapp

Stefan Biffl

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Performance of Paradigms for Storing and Querying Multidisciplinary Engineering Models

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering / Internet Computing

by

Michael Wapp, BSc

Registration Number 01325702

to the Faculty of Informatics

at the TU Wien

Advisor: Ao. Univ.Prof. Dr. Stefan Biffl
Assistance: Dipl.-Ing. Kristof Meixner, BSc

Vienna, 2nd December, 2019

Michael Wapp Stefan Biffl

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

Erklärung zur Verfassung der Arbeit

Michael Wapp, BSc
Bahnstraße 16-18/2/11 7000 Eisenstadt

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

Wien, 2. Dezember 2019

Michael Wapp

Danksagung

Ich möchte mich hiermit bei meinen Betreuern Dr. Stefan Biffl, Dr. Dietmar Winkler und Dipl.-Ing. Kristof Meixner für die Unterstützung bei der Entstehung dieser Diplomarbeit bedanken. Dabei möchte ich besonders meinen Dank an Kristof Meixner für seine Geduld und Motivation richten. Weiters bedanke ich mich auch bei meiner Familie und meinen Freunden für die geleistete Unterstützung während meiner Diplomarbeit, vor allem möchte ich mich bei meiner Freundin Sirimah bedanken, welche diese Arbeit Korrektur gelesen hat.

Wir bedanken uns für die finanzielle Unterstützung durch das Bundesministerium für Digitalisierung und Wirtschaftsstandort und die Nationalstiftung für Forschung, Technologie und Entwicklung.

Acknowledgements

I would like to thank my advisor Dr. Stefan Biffl and his assistants Dr. Dietmar Winkler and Dipl.-Ing. Kristof Meixner for their guidance through this master thesis. Especially Kristof Meixner for his patience and motivation. I would also like to thank my family and friends for their support and most importantly my girlfriend Sirimah who was involved in proofreading process of this thesis.

The financial support by the Austrian Federal Ministry for Digital, Business and Enterprise and the National Foundation for Research, Technology and Development is gratefully acknowledged.

Kurzfassung

Im Bereich der multidisziplinären Produktionssystemplanung, werden Daten zwischen den verschiedenen beteiligten Ingenieuren ausgetauscht. Dieser Datenaustausch muss möglichst effektiv und effizient sein, um einen erfolgreichen Projektablauf zu gewährleisten. Gewöhnlich wird dieser Datenaustauschprozess von multidisziplinären Modelldaten, wie etwa Automation Markup Language (AML), mit Technologien wie E-Mail durchgeführt. AML ist ein maschinen-lesbares Datenaustauschformat das auf der Extensible Markup Language (XML) basiert und von verschiedensten Ingenieursdisziplinen verwendet werden kann. Historisch betrachtet, war AML als reines Austauschformat geplant, erhält aber immer mehr Aufmerksamkeit als Modell um ingenieurstechnische Information zu beschreiben. Dies führt dazu, dass die Darstellung und Speicherung von AML in XML in Hinsicht auf Zugreifbarkeit und Lesbarkeit oft nicht ausreicht. Deshalb ist es notwendig passende Speichertechnologien zu finden, um AML Modelle effizient persistieren zu können.

Diese Arbeit greift dieses Problem auf und schlägt eine Lösung vor die Einsicht in Datenspeicherungsmöglichkeiten für multidisziplinäre Modelldaten ermöglicht. Weiters werden zwei Datenbanklösungen für AML Modelldaten für das Speichern und Auslesen der Daten evaluiert. Um diese Performanzevaluierung durchzuführen, werden folgende Schritte ausgeführt. Zu Beginn werden zwei Datenspeicherungsparadigmen und -lösungen basierend auf Kriterien welche sich Multidisciplinary Engineering (MDE) Modelldaten ergeben ausgewählt. Danach wird eine Softwarearchitektur vorgeschlagen die es ermöglicht, Datenbanklösungen flexibel auszutauschen und AML Modelldaten zu speichern und auszulesen. Zusätzlich wird ein Prozess definiert welcher in der Durchführung der Evaluierung der zwei gewählten Datenspeicherungslösungen angewendet wird. Die gewählten Datenbanken sind BaseX als XML-basierte und Neo4J als graph-basierte Datenspeicherungslösung. Die Ergebnisse der Evaluierung werden analysiert und diskutiert um einen erweiterten Einblick zu bekommen welcher definiert welche Datenbank unter welchen Konditionen für AML Modelldaten geeignet ist. Im Detail ergeben sich daraus Erkenntnisse, welche für die Entscheidung welche der Datenbanken besser geeignet ist genutzt werden können. Die Analyse der Ergebnisse zeigt, dass BaseX für das Erzeugen, Aktualisieren und Löschen und Neo4J für lesende Operation besser geeignet ist. Es wird festgestellt, dass es keine klare Empfehlung gibt, welche der beiden Datenbanken in multidisziplinären Projekten verwendet werden soll da immer die situationsabhängige Nutzung im Projekt beachtet werden muss. Die Erkenntnisse der Arbeit können verwendet werden, um Datenbanken für das Speichern und Auslesen von AML Modelldaten zu ermitteln.

Abstract

In Multidisciplinary Engineering (MDE), projects rely on the effective and efficient coordination and collaboration of participating project members from various engineering disciplines such as electrical and mechanical engineering. Therefore, an effective and efficient exchange of engineering information and artifacts is vital to a successful project execution. Usually engineers exchange MDE data such as Automation Markup Language (AML) files by sharing them via common data sharing processes such as email. AML is a machine-readable data format which is based on the Extensible Markup Language (XML) allowing engineers from various disciplines to describe engineering data. Historically, AML was only meant as a data exchange format but nowadays receives more and more attention as a general model to describe engineering information. This however leads to the problem, that the representation and storage of AML as XML file is often insufficient in the sense of, e.g., querying or access control. Hence, it is important to find suitable storage technologies to persist the AML engineering model efficiently.

This thesis addresses this problem and proposes a solution to compare and evaluate storage solutions for MDE model data. Moreover, the thesis evaluates two selected data storage solutions for AML model data for their storing and querying performance. In order to carry out this performance evaluation, the following steps are executed. At first, two data storage paradigms and solutions are selected based on elicited criteria from MDE model data and related work. Then, a prototypical layered software architecture is designed and built which allows for a flexible exchange of the underlying data storage solution as well as storing and querying of AML model data. Furthermore, a performance evaluation process is established and used to execute a performance evaluation benchmark on the two selected data storage solutions, i.e. BaseX as an XML data store and Neo4J as a graph data store. The results of the evaluation are analyzed and discussed in order to provide deeper insight on the capabilities of the data storage paradigms to store and query AML model data. In particular, the results provide findings which can be used to determine whether one of the selected data storage solutions is better suited under certain conditions. The analysis of the performance results showed, that BaseX performs better for use cases which involve create, update and delete operations while Neo4J performs better for use cases involving read operations. In conclusion, a clear recommendation for MDE projects using AML as a data exchange format cannot be provided as it depends on the intended usage of the data storage solution. Yet, the gained insight can be used to determine a data storage solution for a MDE project.

Contents

CHAPTER 1

Introduction

This Chapter provides an introduction to the topic of database paradigms as persistence solutions for engineering data formats. Section 1.1 explains the motivation to study database paradigms that enable a better exchange of Multidisciplinary Engineering (MDE) models. Furthermore, Section 1.2 describes the problems which arise with exchanging MDE model data and Section 1.3 discusses the proposed results to these problems. Lastly, Section 1.4 provides a brief overview of the structure of this thesis.

1.1 Motivation

MDE projects such as the engineering needed for a production system and their according processes, such as planning a production system (e.g. a car manufacturing plant), typically involve heterogeneous tools and systems used independently by engineers of various engineering domains such as mechanical, electrical or logical engineering [10]. These domain-specific tools are all used to manipulate the same engineering product (e.g. a production plant); therefore, the underlying data also describes the same project. One of the main goals of the engineering process is the consistency of plan data among engineers in order to avoid duplicated or lost information. Typically, engineering data is shared via email or other basic forms of transferring data which can involve loss of information and usually only focuses on bulk data changes [18]. According to Drath [18], the data exchange format Automation Markup Language (AML) provides a solution to this problem. AML is an industry-standard data exchange format for production systems based on the Extensible Markup Language (XML) that aims at improving the data exchange process for MDE processes [25]. It provides the syntax and semantics to define automation objects which represent the entities of the modeled production system; this can range from a small screw of a robot to a complete construction hall. By utilizing AML for MDE, data can be exchanged via a standardized common data format that can be read by all involved engineers and is mappable to formats of other existing software tools. Nevertheless, the process of sharing information between engineers, their software tools and systems is far from being considered as trivial since each engineering tool used to modify data usually utilizes a proprietary data format that is tailored to the tool or system [51]. Basic import and export functions are typically provided, but in order to use an open data exchange format such as AML the engineering data has to be converted into the format used by the engineering software tool. In order to fully utilize AML as a supported data exchange format, the engineering tools would have to be adapted. AML itself is just a data exchange format stored in the form of an XML file and therefore the engineering process differs only by switching from several data sources to one data source in the form of a file. This AML file also has to be maintained and shared among engineers in order to avoid duplicated or lost information. Therefore, in order to achieve a central data source that is used by all stakeholders involved and would store data losslessly, AML itself has to be stored in a central data storage solution accessible by all engineers and their according tools and systems. This data storage solution has to be efficient in storing AML model data and provide an open access to the data so that it can be used by engineers working in various domains and their according software tools.

1.2 Problem Statement

In order to efficiently use a data exchange format such as AML for MDE projects, an efficient storage solution which improves the process of sharing data among different engineers of the MDE process is needed. In particular, a centralized storage solution for a common data exchange format such as AML omits the additional overhead of sharing data. Figure 1.1 shows the engineering process which uses a centralized data storage solution which is accessible for all involved engineers. Storing the AML model data in a central data store would keep the model up-to-date for every engineer involved. In addition, a concurrent data store allows for the tracking of changes which would avoid arising problems as a result of incompatible plan data. By utilizing such AML model data storage solutions, the overall work flow for engineering teams could be improved, leading to a more streamlined and less complex development. However, indicators need to be established for the evaluation of different data storage paradigms and comparison of their advantages and disadvantages as well as for measuring the performance characteristics of these paradigms.

The main stakeholders of this thesis can be categorized into four groups. The first stakeholder group consists of engineers of different disciplines such as electrical, mechanical or software engineering which face data exchange process problems directly. The second category–the project managers–are typically responsible for project planning, monitoring and controlling as well as team coordination. Project management itself deals with problems such as deadlines and time constraints and is therefore also directly influenced by a problematic data exchange process. The third category are tool developers which focus on the design and creation of software products that can be used in the processes of MDE. These developers face problems during the implementation such as choosing a

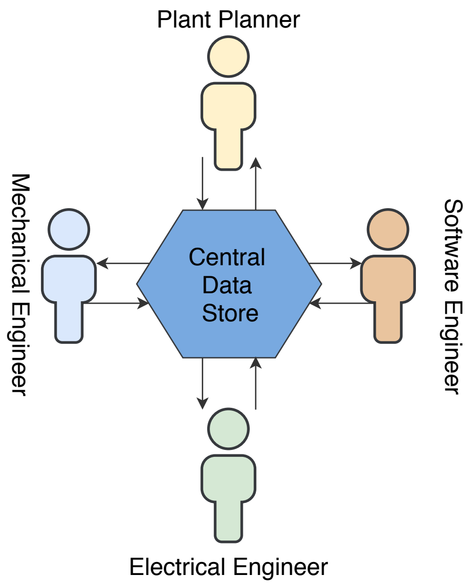


Figure 1.1: Engineering Process with Central Data Storage

data storage solution for MDE model data. The fourth group are researchers and domain experts which provide input and findings to the area of MDE.

1.3 Expected Results

The expected outcome of this thesis is a performance evaluation design. This design can be used to measure the performance of different data storage paradigms for storing and querying AML model data. For the performance evaluation, data storage solutions will be selected based on their applicability to support the specific data model structure of AML. The performance evaluation will focus on data storage CRUD operations (e.g. Create, Read, Update, and Delete operations) use cases. Furthermore, the example AML input data will be provided by research experts of the Otto-von-Guericke University Magdeburg, Germany that created this AML model as an academic demonstration and testing project with focus on a production system [37]. The comparison will provide a foundation for choosing a data storage paradigm for managing MDE model data, in particular AML model data.

The following steps will be executed in order to achieve the expected results of this thesis:

The first step is to identify suitable data storage paradigms for managing MDE model data. Therefore, common data storage paradigms that are suitable for analysis will be selected. Based on the result list, a subset of specific database implementations that look promising to be able to support the efficient storage of AML model data will be defined. Furthermore, performance indicators that are used to measure the efficiency and speed of the data storage paradigm have to be selected . These indicators will be derive from the defined use cases since the outcome of this thesis is an evaluation of the chosen paradigms for managing MDE model data.

Secondly, use cases have to be defined which can be used for the performance evaluation. In particular, a set of CRUD operations for AML model concepts will form the list of use cases.

Based on the chosen database types and use cases, a prototype will be designed and developed that can be utilized to store an example MDE model. In particular, this prototype design will use AML models. It will provide a suitable software architecture that can be transparently used from the underlying database technology and will be able to execute defined use cases on the database implementations. Furthermore, the prototype allows for a performance evaluation based on the defined performance indicators. The execution of this performance evaluation leads to different results which will be compared in the last part of this thesis.

A discussion of the performance of the proposed data storage paradigms for managing MDE model data based on the results of the performance evaluation will form the last part of this thesis The outcome of this part should be a detailed comparison between the performances of the chosen data storage implementations for the defined use cases on the MDE data exchange format AML.

1.4 Thesis Structure

The remainder of this thesis is structured as follows:

- Chapter 2 defines the concepts used in this thesis and describes the related work this thesis is based on.
- Chapter 3 defines the research questions which will be answered in the discussion.
- Chapter 4 explains the research approach and methodology used for this thesis.
- Chapter 5 defines the use cases for the performance evaluation.
- Chapter 6 gives an in-depth discussion about the prototype implementation which is used to execute a performance evaluation based on different storage paradigms.
- Chapter 7 describes the process of the performance evaluation which is executed in this thesis.
- Chapter 8 discusses the results of the performance evaluation of Chapter 7 based on defined performance indicators.
- Chapter 9 concludes the thesis by summarizing the limitations of the implemented prototype and provides an outlook on possible future work based on the prototype implementation executed in this thesis.

CHAPTER 2
Related Work

This chapter 2 provides an overview of the related work which serves as a source of basic knowledge relevant to this thesis. Furthermore, the context and area of this thesis are established. Section 2.1 introduces the concept of Multidisciplinary Engineering (MDE) by presenting a general overview and Section 2.2 provides an introduction to Automation Markup Language (AML) by describing the approach and its structural overview. Different data storage paradigms that can be used to manage MDE data will be explained in Section 2.3. Lastly, Section 2.4 defines the process of benchmarking data storage paradigms by giving a detailed introduction on how to measure the efficiency and suitability of different data storage paradigms for MDE data.

2.1 Multidisciplinary Engineering

According to Biffl et al. [10], MDE is an area in modern engineering where engineers from multiple engineering disciplines need to work together in order to create a Cyber-Physical Production System (CPPS) [10]. CPPS are systems where computational entities collaborate with the surrounding physical world and its processes as well as providing data or accessing it on the internet [41]. For example, as stated by Moser et al. in [43] industrial production automation systems consist of manufacturing systems which construct complex products and goods by combining smaller parts. These production lines are used in the car or furniture industry and allow for autonomous manufacturing processes. Typically, during MDE processes different engineering tools and systems are used [10]. MDE is used in engineering projects where systems, for example a production line or a power plant, are created [9]. The two main engineering disciplines in the 1950s were mechanical and electrical engineering [10]. With the introduction of logical engineering (e.g. software engineering) in the years following 1950, the engineering process has become much more complex. Additionally, all engineering disciplines have become more specialized and developed their own methods, models and terminologies. The current state of engineering processes is a result of all aforementioned factors combined: modern CPPS are complex and not trivial to manage. By providing general rules and applications that aim at streamlining the process of MDE the engineering itself becomes more usable for engineers and saves time during the execution of the engineering project.

As defined in [10], talking about MDE also implies talking about the following disciplines:

- Mechanical engineering, which is one of the oldest engineering disciplines and covers areas such as physics, mathematics and material sciences.
- Electrical engineering, which deals with the study, design and conception of systems that include electricity.
- Logical Engineering (Software Engineering), a discipline where engineering is applied to the conception of software systems.

Production Systems Engineering

Production Systems Engineering (PSE) can be described as a part of MDE. The main goal lies in providing quantitative and analytic methods for the analysis, continuous improvement and design of production systems [10]. By utilizing terms such as bottleneck, leanness and continuous improvement emerged with detailed production data, PSE offers design and managing metrics aiming at providing highest efficiency and performance. PSE follows a life cycle process [8].

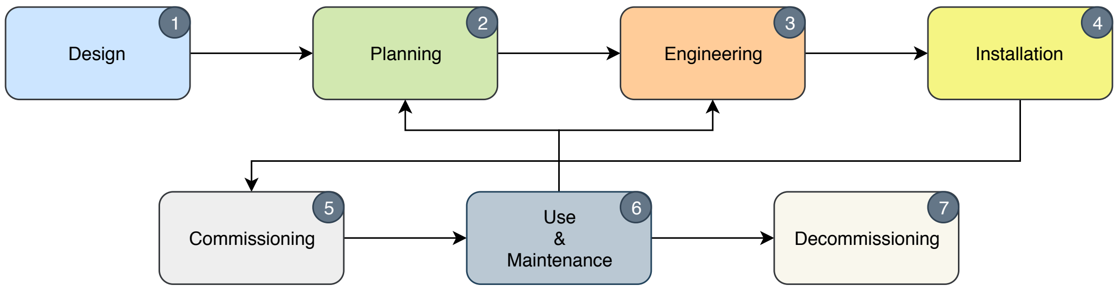


Figure 2.1: PSE Process

This process–shown in Figure 2.1–includes the system design phase (Pos. 1), the planning phase (Pos. 2), the engineering phase (Pos. 3), the installation phase (Pos. 4), the commissioning (Pos. 5), use and maintenance (Pos. 6) and decommissioning (Pos. 7). In addition, PSE projects can traverse back from the use and maintenance step to the planning and engineering phase due to problems and limitation in the execution. All these steps pass data to the subsequent step which involves sharing data between different engineers and their used tools. As different engineers have different views on the engineering project, they usually do not share the same plan. This results in several problems in terms of exchanging planning data and communicating on the project between two different engineering departments. Therefore, a data format which can be used to model the whole system by providing object-oriented concepts can be an improvement for that particular area of MDE [18]. By agreeing to use this data format as a central exchange format, engineers of all disciplines can communicate with one another by using it. It would eliminate the need for exchanging data in different data formats between different engineering departments. Makris and Alexopoulos [35] propose a prototype AML server which aims at providing a central storage solution for AML model data. The focus lies on providing a central server for collaborative computer-aided production engineering processes that enables the interaction of manufacturer and supplier companies. A data exchange format which is relevant to this thesis will be introduced in Section 2.2.

2.2 Automation Markup Language

AML is a data exchange format for MDE based on the Extensible Markup Language (XML), which was first introduced by Graeser et al. [25]. AML is aimed specifically for the purpose of providing a common data exchange format for engineering data and is therefore particularly useful for the area of MDE as it aims at solving the problem of having to deal with different plan data for different engineering departments [21]. According to Drath et al. [19], AML stores information about plants in terms of structure (topology and geometry) and behaviour (logical and kinematics). Furthermore, as stated in [40] by Miriam and Drath, AML is able to store vendor independent interlinked engineering data.

2.2.1 Historical Overview

The process of MDE involves engineering of different domains in a combined effort. Before the concept of MDE came up, the engineering steps used for engineering projects were strongly separated [10].

A strong separation provides the benefit of having distinct steps for each part of the process. However, a negative aspect is that the communication and exchange of information between those separated processes is more difficult in comparison to a homogeneous process. The problem occurs for the exchange of the data between all different engineering tools. According to Drath [18], this exchange process is a significant bottleneck of the PSE process.

In order to solve this problem, Daimler AG evaluated different data exchange formats [18]. Based on this evaluation, Daimler AG initialized the development and standardization of AML. Is is intended to be used as a standardized data exchange format for plant engineering [18]. This standardization was initialized in a industry consortium founded with the companies ABB[1], KUKA[2], Rockwell Automation[3], Siemens[4], netAllied[5] and Zühlke[6] and on the scientific side the University of Karlsruhe[7] and the University of Magdeburg[8] in 2006 [18]. This consortium founded an organization with the intent to further standardize and develop AML.

2.2.2 Approach

AML is used to describe the components of a production system of plants [25]. Based on the actual hierarchy of components, a described object can have sub-components and also ancestor-components (e.g. being part of something bigger). These components can range from a small screw, a production machine or a complete production line. It is possible to describe these components using different levels of detail. In terms of software engineering, AML is a data exchange format which follows the object-oriented and template-based approach.

AML itself utilizes different standardized aspects to describe the components as well as its structure and properties [18]:

Computer Aided Engineering Exchange (CAEX)

CAEX is a data format developed by RWTH Aachen in 2002 with industrial support of the ABB cooperation [18]. Its intended purpose is to provide a vendor- and system-neutral data format that allows to store hierarchy information of engineering objects or entities. CAEX can be used for a production system to represent the hierarchy in the plant (e.g. from production line to production machine and a particular part of the small machine). Basically, CAEX provides a way to represent the engineering objects and entities as the object-oriented paradigm is providing in the software development world [18]. Therefore, the concepts of inheritance, relations, instances, class libraries, instance hierarchies, attributes and interfaces are supported in CAEX. It is important to state that CAEX does not only support plant or production systems but can also be applied to all forms of hierarchical objects which correspond with each other in an object-oriented manner. CAEX is based on XML and can be verified using a XML Schema Definition (XSD) file. As of writing this thesis, the CAEX productive version used for AML is Version 2.15.

[1]Asea Brown Boveri AG `new.abb.com`
[2]Kuka AG `www.kuka.com`
[3]Rockwell Automation, Inc. `www.rockwellautomation.com`
[4]Siemens AG `www.siemens.com`
[5]NetAllied Systems `netallied.de`
[6]Zühlke Technology Group AG `www.zuehlke.com`
[7]Karlsruhe Institute of Technology `www.kit.edu`
[8]Otto von Guericke University Magdeburg `www.uni-magdeburg.de`

Collaborative Design Activity (COLLADA)

COLLADA is an open and interchangeable data format for 3D applications [5]. The data format was originally developed by Sony Computer Entertainment[9] but has become a shared copyright between Sony and the Khronos Group[10], which is an industry consortium dealing with the development of open industry standards and APIs. COLLADA is XML based and defines an XML schema for exchanging data. The data can be seen as digital assets of various graphical software applications. The COLLADA data format standard is compatible with many software tools and game engines including Adobe Photoshop, Blender, Cinema 4D, Unity, ArchiCAD, and Autodesk[11].

PLCopen XML

PLCopen is an industry organization which independently aims at developing efficiency and industrial standards in automation based industries. PLCopen's main area interest is the development of logic-focused industry standards as defined in IEC 61131-3 [18]. Additionally, they provide an XML schema–PLCopen XML–which can be utilized to describe logical processes in plant and production systems including sequences, internal behaviour of processes and input/output connections. Therefore, this schema is used to define attributes of plant and production systems in AML. PLCopen XML was first published in 2005 and has since been updated several times. As of writing this thesis, the current version of PLCopen XML–used in AML–is version 2.01.

2.2.3 Structure

As AML is XML data, it is by definition semi-structured data. Listing 1 shows a simple skeleton of an AML file.

As shown in Listing 1, the data is structured in an XML form where the XML tags correspond to the object hierarchy defined in CAEX and the attributes are used to define relationships. The usage of parent and child XML elements makes it easy to see–in this example file–how the data is structured. In comparison to the example shown in Listing 1, bigger files with several thousand lines of data are much more complex and therefore usually require more effort for humans to understand their overall structure. Nevertheless, as machines typically prefer structured data, complex files are perfectly processable by them and only require more computing power or longer execution times.

In addition to the example shown in Listing 1, Figure 2.2 shows the introduced modelling concepts in their relationship to each other. This hierarchy shows the parent elements and their according child elements. The modelling concepts themselves also provided inter-element links which combine several child elements via links (i.e. graph like structure).

[9]Sony Computer Entertainment `sony.at`
[10]Khronos Group, Inc. `khronos.org/`
[11]Collaborative Design Activity `khronos.org/collada`

```xml
<?xml version="1.0" encoding="utf-8"?>
<CAEXFile FileName="CAEX_simple.aml" SchemaVersion="2.15">
  <InstanceHierarchy Name="ManufacturingSystem">
    <InternalElement Name="firstScrewdriver"
    RefBaseSystemUnitPath="LibCT/ElectricScrewdriver">
    </InternalElement>
    <InternalElement Name="secondScrewdriver"
    RefBaseSystemUnitPath="LibCT/ElectricScrewdriver">
    </InternalElement>
  </InstanceHierarchy>
  <InterfaceClassLib Name="MyInterfaces">
    <Version>1.0</Version>
    <InterfaceClass Name="Energy"
    RefBaseClassPath="BaseInterfaceClassLib@
    AutomationMLInterfaceClassLib/
    AutomationMLBaseInterface"/>
  </InterfaceClassLib>
  <RoleClassLib Name="ManufacturingRoleClasses">
    <Version>1.0</Version>
    <RoleClass Name="Tool"
    RefBaseClassPath="BaseRoleClassLib@
    AutomationMLBaseRoleClassLib/
    AutomationMLBaseRole"/>
  </RoleClassLib>
  <SystemUnitClassLib Name="LibOfCommonTools">
    <Version>1.0</Version>
    <SystemUnitClass Name="ElectricScrewdriver">
      <ExternalInterface Name="EnergySupply"
      RefBaseClassPath="MyInterfaces/Energy"/>
      <SupportedRoleClass RefRoleClassPath=
      "ManufacturingRoleClasses/Tool"/>
    </SystemUnitClass>
  </SystemUnitClassLib>
</CAEXFile>
```

Listing 1: Example for AML File

The example shown in Listing 1 includes all modelling concepts that are relevant to this thesis. The following list introduces these concepts [25]:

CAEXFile / AMLFile: This concept is the parent entity which holds all properties and elements relevant to the engineering project. Its main purpose is to contain hierarchical information about objects of the plant model which should be represented. For example, a *CAEXFile* is the planning file of a complete plant engineering project.

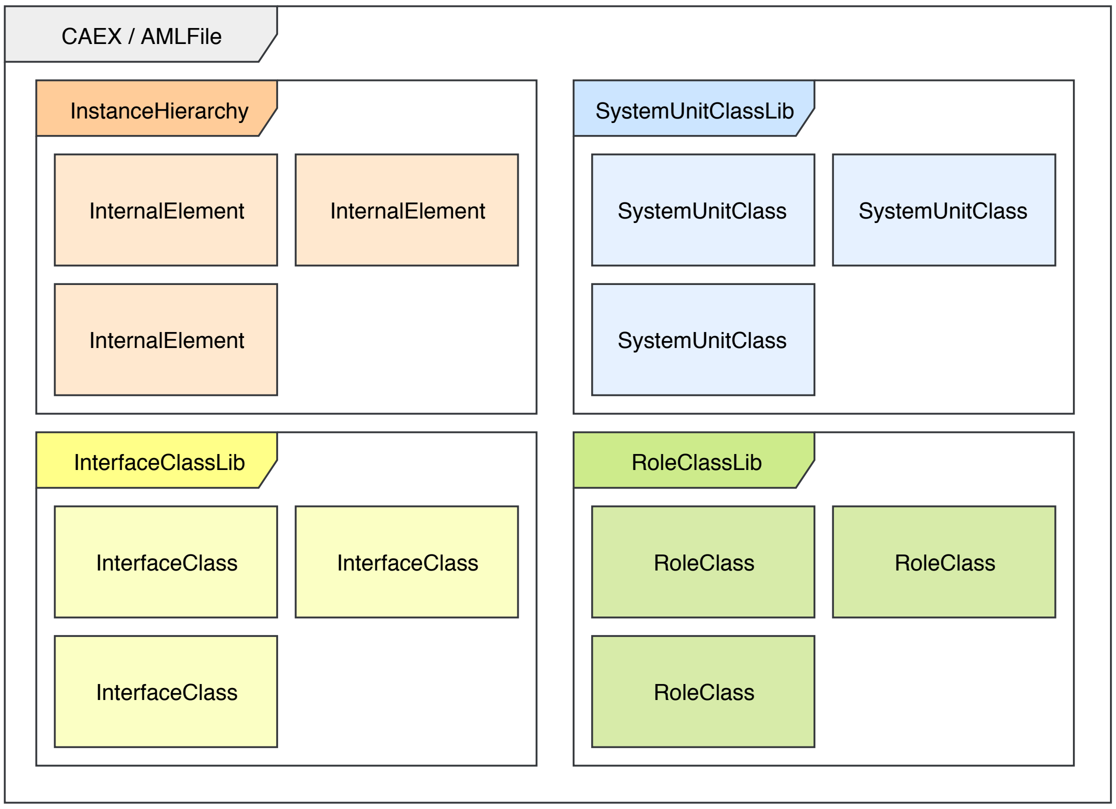


Figure 2.2: AML modelling concepts in its relationship structure

InstanceHierarchy: This concept is used to store the topological data of projects and is therefore the core of AML. As *InstanceHierarchies* are used to store all engineering-related information, they contain all data objects and their according properties as well as the relations to other objects and entities. For example, an *InstanceHierarchy* can be used to define a manufacturing system in a production line. *InstanceHierarchies* are located directly inside an *CAEXFile* as top-level concepts.

InternalElement: This concept can be seen as objects that represent real or logical entities e.g. robots or pumps. For example, an *InternalElement* can be a actual instance of a small tool in a manufacturing system. *InternalElements* are used inside *SystemUnitClasses* or another *InternalElement*.

SystemUnitClassLib: This concept contains a library of plant entities (*SystemUnit-Class*) where one *CAEXFile* can have as many plant entity libraries as it requires. For example, a *SystemUnitClassLib* can be used to describe a library of all tools used in the production system. As the *InstanceHierarchy*, this concepts is located directly inside an *CAEXFile* as a top-level concept.

SystemUnitClass: This concept describes a plant entity and captures all its relevant properties and relations. These entities are concrete objects and are typically reused. For example, *SystemUnitClasses* describe a specific type of tool in a manufacturing system. *SystemUnitClasses* are located inside a *SystemUnitClassLib*.

RoleClassLib: This concept is used to capture all roles (i.e. abstract characteristics) which are relevant to define the plant entities in the project. For example, all tools used for a manufacturing system have specific roles assigned and these roles are all contained in a *RoleClassLib*. As it is a library of *RoleClasses*, it is directly located under the *CAEXFile* and can be considered to be one of the four top-level concepts.

RoleClass: This concept describes a role of plant entity which does not include technical definitions but rather gives a general category in order to structure plant entities in their according *RoleClasses*. For example, all entities in a manufacturing system that are tools have the role *Tool* assigned which is represented by a *RoleClass*. A *RoleClass* is located directly inside a *RoleClassLib*.

InterfaceClassLib: This concept consists of all connections between plant entities. For example, all connections in a manufacturing system (e.g. energy, data) can be defined in an *InterfaceClassLib*. *InterfaceClassLibs* are top-level concepts as they are located directly under *CAEXFile*.

InterfaceClass: This concept defines a connection between plant entities and is located inside an *InterfaceClassLib*. An *InterfaceClass* can represent an energy connection between two components in a manufacturing system.

In order to be able to edit or manage AML model data, several software tools are available. The following list includes typical software tools used for managing AML data:

AutomationML Editor[12] is a software tool developed by AutomationML e.V. [13] which allows to create, visualize and edit AML files. It is mainly suited for evaluation purposes.

AML.hub[14] is a central software platform for exchanging engineering data in the area of industry 4.0 developed by logi.cals[15] and the CDL-Flex research laboratory[16] at the Technical University of Vienna.

COMAN[17] is a project management tool for MDE projects aimed at providing a synchronous view for all stakeholders of the project.

[12]AutomationML Editor `automationml.org/o.red.c/tools.html`
[13]AutomationML e.V. - open AutomationML society `automationml.org`
[14]AML.hub `amlhub.at`
[15]logi.cals GmbH `logicals.com`
[16]Christian Doppler Laboratory
Software Engineering Integration for Flexible Automation Systems
`cdl.ifs.tuwien.ac.at`
[17]Coman GmbH `coman-software.com`

RF::Suite[18] is a software suite for virtual plants and can be used for different project steps such as simulation, visualization, development and execution. AML is used as an import format for plant data.

Siemens TIA Portal V14[19] is part of the Siemens engineering framework and allows for exchanging hardware configuration data via AML.

2.3 Data Storage Paradigms

Data storage paradigms, or also called data store models, are the basic concepts on what a database is based on [3]. Data stores structure data differently based on the data storage paradigm they are following. These different data storage paradigms offer advantages and disadvantages for specific tasks and requirements. In order to select data storage paradigms which are relevant to the topic of storing and querying MDE models, a collection of common data storage paradigms has to be established. In this thesis, the focus lies on the data storage paradigms proposed in the following sections. These paradigms were elicited from interviews with MDE domain experts and selected as relevant data storage paradigms. The following lists provides an introduction to these selected data storage paradigms and introduces to implementation examples:

2.3.1 Semi-Structured Data Paradigm

In order to define the semi-structured data storage paradigm, the concept of semi-structured data has to be introduced. Buneman [15] defined semi-structured data as data that does not strictly follow a structure as it is typically contained in the data itself. The most prominent example for semi-structured data is XML [14]. XML is a markup language which can be used to define hierarchical data structures in a text document. Due to the generality of XML, it is widely used across the internet and in different areas of engineering. As AML is based on XML [25], semi-structured data is already used in the area of MDE. JavaScript Object Notation (JSON) is another example for semi-structured data.

Based on this introduction of semi-structured data, the semi-structured data storage paradigm utilizes semi-structured data as a basis to store data. The following subsection introduces to an example of a database implementation which is based on the semi-structured data storage paradigm utilizing XML.

[18]RF::Suite `eks-intec.de/rf.html`
[19]Siemens TIA Portal V14
`new.siemens.com/global/de/produkte/automatisierung/industrie-software/automatisierungs-software/tia-portal.html`

XML-Based Database

XML-based databases or XML databases are data persistence software systems that use the XML format for specifying and storing data [28]. XML databases are specific document-oriented databases or also called Non Structured Query Language (NoSQL) (see Section 2.3.3) databases due to the fact that Structured Query Language (SQL) (sse Section 2.3.2 is not used for querying and managing data [28].

As most other data storage solutions, data can be queried by using a specific data query language. Therefore, XML databases use a querying languages such as XML Query Language (XQuery), which is a World Wide Web Consortium (W3C) recommendation[20]. XQuery can be used to extract and edit parts of big XML-databases or documents. In general, XML-databases are not as widely used as other databases (such as relational databases) but they are relevant[21] in the industry of MDE [28].

Example: BaseX

BaseX is an open-source implementation of the XML-based database paradigm[22]. BaseX is implemented using the Java[23] programming language. For accessing data in BaseX XQuery is used. BaseX implements the XQuery 3.1 processor and fully supports the newest full text implementation defined by the W3C.

BaseX is structured in three parts:

BaseX Client/Server: The BaseX server implements the database functionality in a small package which does not depend on other external libraries. By implementing a client server architecture, it offers a central storage solution for XML documents. Additional features such as a REST server (to query and edit the data) and a WebDAV Server to access data using a web browser are also included in this part of the database.

XQuery Processor: In order to be able to provide efficient querying of XML, BaseX implements an XQuery processor that supports, at the time of writing this thesis, XQuery version 3.1. By providing different modules which are accessible by the database user, this processor offers the key functionality which makes BaseX an ideal solution for managing a huge number of XML files.

BaseX UI: The graphical user interface provides a way to visually inspect the main database feature and the data stored in the databases. Additionally, a powerful query editor is implemented that enables the user to execute queries on the connected database.

[20]XQuery 3.1: An XML Query Language `w3.org/TR/xquery-31`
[21]DB-Engines Ranking: `db-engines.com/en/ranking`
[22]BaseX - The XML Framework `basex.org`
[23]Java Programming Language: `https://www.java.com/`

2.3.2 Relational Data Model Paradigm

As stated by Elmasri and Navathe in [20], relational databases provide a way to store and manage data based on the relational model. This data model can be seen as the relational data model paradigm. The relational model utilizes the concept of mathematical relations by structuring the data in tables with different columns. Tables represent entities of the data model containing columns (attributes of the entity). One row of the table is a specific instance of the entity. Columns can contain keys which map to other columns in different tables in order to represent a connection between the data. Based on a survey among database market share[24], relational database systems are the most widely used data storage systems. The relational database paradigm offers a standardized query language–Structured Query Language (SQL)–which can be used to retrieve data from the data store. The following subsection provides an introduction to a relational database implementation.

Relational Database

Relational database follow the relational data model paradigm by utilizing the relational approach for storing and managing data. By structuring data into tables which represent entities and columns which represent parameters of those entities the relational model follows a straight forward approach on how to store data.

The general approach of the relational model is based on entities (tables) which have parameters (columns) and relationships (links between the data). In order to achieve that, the relational model introduces the concept of keys. Keys and relationships can be defined as follows [20]: (a) *Keys* are used to uniquely identify instances of entities (a tuple or row in a table) and links between those entities. (b) *Relationships* are interconnections between tables via keys.

Relational databases usually follow the *ACID* principle. According to Elmasri and Navathe [20], the *ACID* principle is defined as follows:

- The **Atomicity** property requires the database to guarantee that transactions of databases either succeed or fail completely and leave the database unchanged. This property must be guaranteed in every situation the database can be in.
- The **Consistency** property requires database transactions to transfer databases only from a valid state into another valid state.
- The **Isolation** property requires the database to ensure that concurrent transactions result in the same database state as if the same transaction were executed sequentially.
- The **Durability** property requires the database to ensure that committed transactions stay committed even in case of a system failure.

[24]DB-Engines Ranking: `db-engines.com/en/ranking`

To manage the data stored in relational databases a query language is needed [20]. Therefore, most of the implementations use SQL which allows the user to maintain and query the data stored in the system.

Relational Database Management System (RDBMS) are the actual implementations of the relational database model. The following subsection introduces an example implementation of a RDBMS:

Example: PostgreSQL

PostgreSQL is an open-source implementation of the relational database paradigm[25]. It is designed to be used from small applications to large data warehouses due to its simplicity and flexibility. The developer of PostgreSQL–the PostgreSQL Global Development Group–states that this database system needs minimal maintenance effort due to the provided stability. Furthermore, PostgreSQL is designed to be extensible providing, for example, the option to define user specific data types.

2.3.3 Document-oriented Paradigm

According to Han et al. [29], document-oriented databases, also known as document stores, are designed to support mass storage of data provided in a semi-structured data format. As document stores are part of the NoSQL database category, they do not utilize a relational model and SQL to store and retrieve information.

Document-oriented Databases

The main design goal of a document-oriented database is to manage information which is provided in a document-oriented way [29] e.g. utilizing a JSON document or an XML document. This document-orientation can also be seen as semi-structured data and therefore, document-oriented databases are a subset of the main data storage paradigm semi-structured data. Data is stored in documents which provide a structure for the data by utilizing different data formats such as XML or JSON. Furthermore, document stores are NoSQL databases, which do not use SQL to manage data stored in the database. Document-oriented databases also utilize a query language to retrieve data. This query language is not standardized and differs between implementations of this paradigm.

Example: MongoDB

MongoDB[26] is an implementation of the document-oriented database paradigm [6]. The format of the internal documents used by MongoDB is BSON[27], a binary representation of JSON documents. In contrast to JSON, more data types are provided by BSON.

[25]PostgreSQL `postgresql.org`
[26]MongoDB: mongodb.com
[27]Binary JSON: `http://bsonspec.org/`

MongoDB itself provides functionality to execute data management operations such as creating, accessing, updating and deleting of data.

2.3.4 Graph Data Paradigm

Angles and Gutierrez [3] defined that in graph databases a graph is used to model the data structure for schema and instances. According to Vicknair et al. [56], it is simple to store a graph in a relational database but the problem is the querying performance. As data structures may contain cyclic links between data entities, fitting this data into a relational data store approach could lead to performance disadvantages. Therefore, representing the data in graph-like structures can improve the (querying) speed of the database [56]. The graph data paradigm utilizes this graph-based concept to store and manage data in the data store.

Graph Databases

Graph databases are the implementation of the graph data paradigm which utilize a graph as their main data store model. In contrast to the relational model, the following concepts are used to store data [3]:

- A **node** represents an entity which can be seen as a row or a record in a relational database.
- **Edges** build direct relationships between different nodes by connecting them. This offers a better traversal speed in order to find connected entities.
- **Properties** can be seen as general attributes for nodes which describe the node in more detail.

In contrast to the traditional relational database model, graph databases require a specific query language that supports accessing and modifying data in the database. There are several implementations of these languages–the most common is Cypher used in Neo4J [32], SPARQL [44], GraphQL [30] and Gremlin [32]. A comparison of common graph databases is shown in [2]. For the purpose of this thesis, one graph database is selected (see Section 2.3.7).

The following subsection introduces an example graph database implementation.

Example: Neo4J

Neo4J[28] is an implementation of the graph database paradigm. It is aimed at providing a data storage solution that focuses on the connections between the stored data by storing everything in the form of an edge, node or attribute (property).

The used querying language is Cypher, which is also developed by Neo4J, Inc.[29]. Cypher

[28]Neo4J Database: `neo4j.com`

[29]Neo4J Cyper Query Language `neo4j.com/developer/cypher-query-language`

offers the capability to query and update the data in the stored graph [32]. The focus lies on simplicity and compatibility to express complex queries in a relatively simple way. This simplicity is achieved by providing the option to label node which can be used inside a database query.

2.3.5 Hybrid Data Storage Paradigms

Hybrid data storage paradigms are a combination of two or more data storage paradigms in a single data storage implementation [24]. By combining the properties of several data storage paradigms, the advantages of two or more data storage paradigms can be bundled into one.

Hybrid Databases

According to Goyal et al. [24], hybrid databases are abstraction layers which combine relational database models with NoSQL-like paradigms. Additionally to the above defined paradigms, there are some hybrid databases which combine features of two or more paradigms. The goal is to combine the best of the different data storage paradigms approaches in order to provide a more suitable data storage solution for specific cases of application.

The following subsection introduces an example hybrid data store.

Example: ArangoDB

ArangoDB[30] is an implementation of a hybrid data storage paradigm. It is a combination of the graph and document data storage paradigm with the addition of features from key/value stores. Furthermore, a unified (e.g. independent from the underlying data model approaches) query language (ArangoDB Query Language (AQL)) is used to enhance the process of accessing and managing data. AQL is an SQL-like data query language.

2.3.6 Linked Data Paradigm

As stated by Bizer et al. in [12], linked data is interlinked to other data sets and can be seen as structured data. Based on this, the linked data storage paradigm focuses on the connection between the data sets. Additionally, linked data is typically identifiable by some form of id. Due to the fact that the data is interlinked with other linked data it is queryable using semantic queries. Semantic queries not only provide a way to retrieve data stored in the linked data but also meta-data (e.g. data about the data) about the linked data which can be used to reason about the data itself. SPARQL Protocol And RDF Query Language (SPARQL)[31] is the W3C recommendation to formulate semantic queries on the web and it is based on the SQL syntax. Linked data is particularly useful

[30]ArangoDB: `arangodb.com`
[31]SPARQL Query Language for RDF `w3.org/TR/rdf-sparql-query`

Data Storage Paradigm	Implementation
Graph Data	Neo4J
Relational	PostgreSQL
Hybrid	ArangoDB
Semi-Structured	BaseX
Document-oriented	MongoDB
Linked Data	DBPedia

Table 2.1: Data Storage Paradigms Relevant for this Thesis

for the web because it is linked together which allows persons and machines to explore the data.

Example: DBPedia

DBPedia[32] is an example of linked data which extracts content from Wikipedia[33] and makes it available in a structured and interlinked form. This provides the benefit of finding answers to questions which require information from data spread across multiple Wikipedia articles. Another benefit is that DBPedia does not only provide access to the content of Wikipedia but also provides connections to other information provided on the internet (e.g. GeoNames[34]). These links are useful for applications by providing much more information than a standard non-interlinked data set would provide.

2.3.7 Relevance to this Thesis

Section 2.3 provided an overview of common data storage paradigms. Table 2.1 lists all data storage paradigms introduced in Section 2.3. For the performance evaluation of this thesis, these paradigms must be analyzed and a selection must be executed based on their relevance to store and query MDE model data. Therefore, this section deals with the relevance of the introduced data storage paradigms to the work of this thesis.

In order to decide whether a data storage paradigm is well suited to store and query MDE models efficiently, criteria have to be defined. As theMDE model data format addressed in this thesis is an AML model, the relevance must be based on attributes and properties of AML. The following listing defines criteria which are used to select the relevant data storage paradigms. In addition, the selected relevant data storage paradigms and their implementations are proposed.

- **Hierarchical Structure**

  As shown in [25], the data structure of AML provides internal links between the stored data. These internal links are part of the AML specification and therefore,

[32]DBPedia `wiki.dbpedia.org`
[33]Wikipedia `www.wikipedia.org`
[34]GeoNames Data Set: `geonames.org`

Data Storage Paradigm	Implementation
Graph Data	Neo4J
Semi-Structured Data	BaseX

Table 2.2: Chosen Data Storage Paradigms for Managing MDE Model Data

AML can be seen as a graph. Therefore, a data storage paradigm for AML should also natively support a graph structure in order to be able to query efficiently and fast. As stated in Section 2.3.4, graph databases provide exactly this kind of functionality by supporting graph-like data structures. The data storage paradigm **Graph Stores** is therefore relevant for managing MDE model data. In particular, the selected implementation of the paradigm will be **Neo4J** because of its high relevancy in the market.

- **XML based Format**

  AML files are typically stored by using the filetype .aml which is by definition XML [25]. As AML is the chosen MDE model which has to be managed, a data storage paradigm that supports XML data is needed in order to efficiently manage AML data. As shown in [28], BaseX provides an efficient solution to query and transform single or multiple XML documents stored in a database. Due to the fact that BaseX is a highly efficient XML storage solution with features, such as querying and managing of the stored data, it is clear that this implementation of the XML-based paradigm is an obvious choice. Furthermore, according to Grün shown in [26] and [27], BaseX is capable of storing an querying large XML files. Therefore, **BaseX** will be another relevant model data management paradigm.

Table 2.2 shows the data storage paradigms and their implementation chosen for the performance evaluation of this thesis.

At the time of writing this thesis, the relational data storage paradigm is the most commonly used paradigm in informatics[35]. Therefore, using this data storage paradigm in the performance evaluation of this thesis would be a rational choice. However, AML model data is used that is based on semi-structured data and provides a graph-like data structure. These two attributes lead to the conclusion that data storage paradigms which mainly focus on providing a data storage solution for semi-structured data or graphs offers the better suitability for this work. Therefore, no relational data storage solution will be used in the performance evaluation. Evaluating the performance for relational data storage paradigms is up for future work (see Chapter 9).

Other data storage paradigms such as document-oriented (Section 2.3.3) or hybrid databases (Section 2.3.5) would also be able to support model data in the format of AML. These data storage paradigms are left for future research as the focus lies on the graph database and the XML database paradigm.

[35]DB-Engines Ranking: db-engines.com/en/ranking

2.4 Data Storage Performance Benchmarking

As the main purpose of this thesis is to find out which of the two selected data storage paradigms is better suited for managing MDE model data, an evaluation process has to be established. This evaluation process has to determine whether one paradigm is better suited than the other. Therefore, this process can be seen as performance benchmark. In [4], database benchmarks are defined as important tools for researches to make comparisons between different database systems. Due to the fact that there is no general rule for the prototypical design used to execute a database benchmark, it is not possible to define a common architecture. Nevertheless, a general abstraction of the main parts a benchmark consists of is provided in the following listing.

- **Input Data**

  Data storage benchmarks usually utilize some form of input data which will be used for the execution of the benchmark. Based on the given input data, the benchmark will execute different steps to process the data and therefore provide different performance results. Therefore, input data can be seen as the basis for every performance evaluation.

- **Execution Platform**

  The data storage benchmark has to be performed on an execution platform which usually is a computing device (e.g. laptop, server). Differences in computation speed may influence the results of the performance evaluation. If eligible, executing the benchmark on different compute devices will enhance the results of the benchmark.

- **Executing Program**

  As data storage benchmarks operate on data storage implementation, an executing program (i.e. the database implementation) is needed. This executing program runs on the execution platform (or in some form of virtualization environment) and utilizes the computation resources of this platform.

- **Measurement Tool**

  In addition to the computation power provided by the executing platform and the data storage implementation another measurement tool is needed. This measurement tool has the following task: gathering the results and measuring the performance of the data storage paradigm under test.

- **Use Cases**

  Use cases provide a set of operations which are measured by the performance benchmark. These use cases can vary from basic CRUD (i.e. creating, reading, updating and deleting data) functions to more complex use cases which combine multiple CRUD operations. These more complex use cases are typically representations of real world business procedures.

Indicator	**Unit**	**Measurement**
Duration of Operation	Seconds	Measure Time
Size of Input Data	Bytes	Storage Calculation
Complexity of Input Data	Level	Using a Tool
Number of Operations	-	-
Compute Performance of Executing System	-	-
RAM Memory Usage	-	-
Storage Memory Usage	-	-

Table 2.3: General Performance Indicators of Data Storage Benchmarks

Indicator	**Unit**	**Measurement**
Duration of Operation	Seconds	Measure Time
Size of Input Data	Bytes	Storage Calculation
Complexity of Input Data	Level	Using a Tool

Table 2.4: Relevant Performance Indicators

Furthermore, performance indicators have to be defined in order to define how results can be measured. This will be done in Section 2.4.1

2.4.1 Data Storage Performance Indicators

In data storage performance evaluations (i.e. benchmarks) a set of performance indicators has to be defined in order to reason about the performance of a database. In order to properly evaluate the data storage performance of different database paradigms (i.e. benchmark execution), researcher define different data storage performance indicators to reason about performance characteristics, the particular context of the respective domain and data model under investigation [4][54][50]. Therefore, Table 2.3 lists general performance indicators used in database performance evaluations. This list is based on performance indicators used in database benchmarks and related work as conducted in [4], [54] and [50].

In addition to the performance indicators listed in Table 2.3, factors like the compute power of the executing system and the used program can alter the results of the performance evaluation. These factors will not be included in the results of the evaluation. As not all performance indicators seem relevant for the performance evaluation of this thesis, Table 2.4 shows relevant performance indicators which are possible candidates used during the performance benchmark. These indicators were selected based on their common usage in other performance benchmark.

CHAPTER 3

Research Questions

This Chapter discusses the research questions of this thesis. The answers to these questions will be provided in Chapter 8. The answers will be based on the results of the performance evaluation as defined in Chapter 7 and will utilize information provided in Chapter 2. The main focus of the research questions is on the evaluation of data storage solutions which are well suited to store and query Multidisciplinary Engineering (MDE) model data.

3.1 RQ1 - Data Storage Paradigms for Multidisciplinary Engineering

Which data storage paradigms are well suited to store and query models of Multidisciplinary Engineering (MDE), such as Automation Markup Language (AML)?

As stated in Section 1.2, a centralized data storage solution could improve the data exchange process in the area of MDE. In order to be able to store and query MDE model data in a central data storage solution, an underlying data storage paradigm must be selected. Therefore, an analysis of data storage paradigms for MDE model data is needed which should reveal promising candidates. This selection can be done by using the characteristics of MDE model data–in particular Automation Markup Language (AML) model data–and comparing them to the characteristics of the data storage paradigms. In addition to analyzing the characteristics of MDE model data, finding appropriate data storage solutions can be achieved by selecting common data storage paradigms which are typically used in the context of software engineering. The focus of this research question is on finding applicable data storage paradigms and establishing a set of criteria for selecting a suitable data storage solution. This results in a list of paradigms which are suitable for storing and querying MDE model data–in particular AML model data.

3.2 RQ2 - Software Architecture for Storing and Querying Multidisciplinary Engineering Models

Which software architecture is well suited for storing and querying MDE models, such as AML models, independently from the implementation of a data storage paradigm?

This research question deals with designing a suitable software architecture which is well suited for storing and querying AML model data. The software architecture has to be established based on the attributes and properties of the data exchange format AML as well as the used data storage solutions (BaseX and Neo4J). Additionally, this architecture should allow for a flexible exchange of the underlying data storage solution. Furthermore, it should be able to execute a performance evaluation which can be used to determine which data storage paradigm and implementation is better suited for storing and querying AML model data. Therefore, the performance evaluation infrastructure (i.e. the software architecture) will be a prototypical design that provides flexibility in the exchange of the underlying data storage solution. In addition to being easily exchangeable, it is important that the performance evaluation process executed using this infrastructure is reproducible and expandable for further additions to the performance evaluation process (e.g. adding new use cases or additional data storage technologies, see Section 9.2). To address this research question, a software architecture should be defined which can be used to execute a performance evaluation

3.3 RQ3 - Performance Evaluation Method

How can a standard database performance evaluation method be adapted to measure the performance of storing and querying MDE models, such as AML models?

In order to determine the performance of the selected data storage paradigms and their implementations, a performance evaluation has to be conducted. The performance evaluation of this thesis will consist of executing a database performance benchmark using example data provided by researchers of the Otto-von-Guericke Univeristy Magdeburg, Germany[1]. This performance benchmark execution process is derived from standard database performance evaluation methods (see [50] and [4]) and utilizes the prototypical design, which is the result of RQ2. Furthermore, an analysis of the performance benchmark results is needed. The results of this data analysis can be used as a basis for further discussion. This analyzed benchmark data is likely to provide deeper insights and allows for reasoning about which data storage paradigm is better suited for storing and querying MDE model data. Therefore, the result of this research questions is a case study of

[1]University Magdeburg: `uni-magdeburg.de`

how a performance evaluation method can look like. This case study is built-up by the performance benchmark process an all its according steps.

CHAPTER 4

Research Approach

This chapter describes the research approach used in this thesis. Section 4.1 discusses the fundamental approach which is applied and provides the scientific background of it. Afterwards, Section 4.2 explains the methodology used, which leads to the research process.

4.1 Design Science Research Approach

Hevner et al. in [31] propose a *Design Science* approach for information system research. Their approach is used as the fundamental research approach for this thesis. The *Design Science* approach is not behavioural driven and differs therefore from the behavioural research approach. It is not based on the creation and refinement of ideas coming from the observation of phenomena inside an environment. In contrast, the *Design Science* approach shows how IT artifacts and their evaluation can be utilized to gain deeper insight to a defined problem. This insight can then be used to find an answer to a defined problem which can then be used to solve this problem. In addition to Hevner et al., Wieringa also define a *Design Science* research approach for information system and software engineering research which follows the same principles.

The artifacts used during the execution of the approach are defined by Hevner et al. [31] as follows:

Constructs can be seen as vocabulary and symbols which provide the language in which problems and their according solutions can be defined and communicated.

Models are defined as abstractions and representations.

Methods are described as algorithms and practices.

Instantiations are implemented and prototype systems.

In order to illustrate the process which is used to execute this research approach, Figure 4.1 shows its concepts and tasks which need to be executed in order to achieve the proposed outcome of providing deeper insights to a defined problem of the research process.

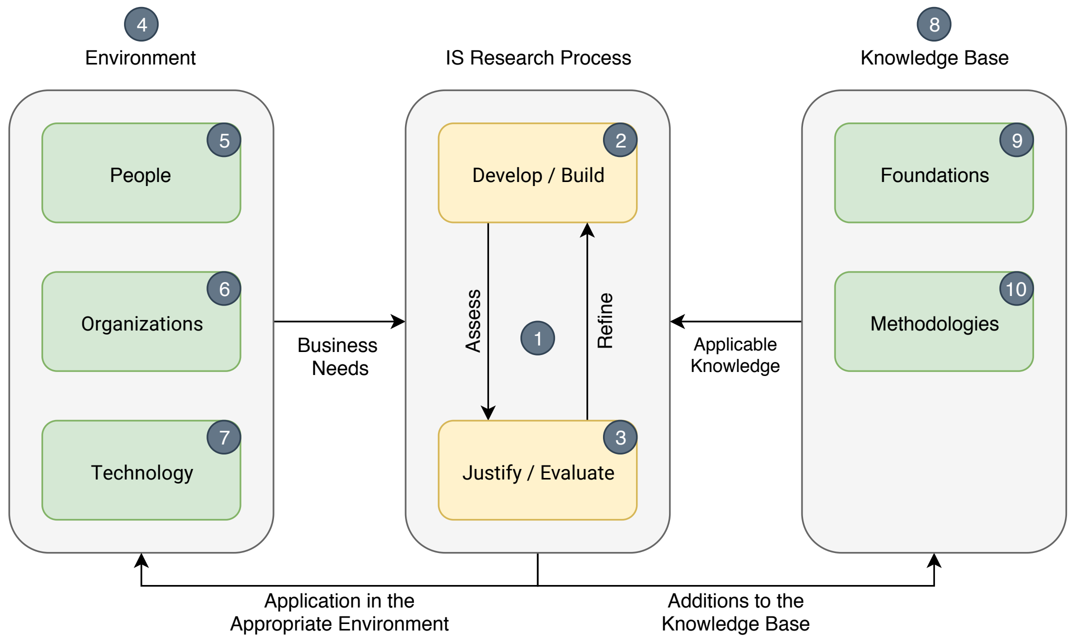


Figure 4.1: *Design Science* Approach Framework by Hevner et al. [31]

As shown in Figure 4.1, the research process is executed in cycles (1). Hevner et al. define these cycles as *Design Science* research cycles and as an iterating process with the goal of building (2) and evaluating (3) the previously defined IT artifacts. These artifacts are based on two input areas, namely *Environment* and *Knowledge Base*.

The *Environment* (4) defines three categories which serve as *business needs* inputs to the iterating IS process. The first category (5) describes the people and their respective needs towards the research process. The capabilities and characteristics of the people involved are relevant for the execution of the research process as they provide input in the form of *business needs*. The second category (6) describes the organizations involved, and their strategies, structures & cultures and processes. The third category describes (7) the technology as well as the according infrastructure, applications, communication's architecture and development capabilities. All these categories build the relevant *business needs* which form the first input to the research process.

The *Knowledge Base* (8) is split into two categories. The first category (9) describes the foundations of the research process i.e. theories, frameworks, instruments, constructs, models, methods and instantiations. The second category (10) describes the used methodologies i.e. data analysis techniques, formalism, measures and validation criteria. These two categories form the applicable knowledge which is the foundation for the research process and provides the second input source.

The information system research process itself is split into two parts:

Develop/Build (2): This part is the main task where the IT artifacts and their concepts are modeled and built. These artifacts serve as one part of the result of the research process.

Justify/Evaluate (3): This part executes an evaluation and justification process which is called in a cyclic manner. This is done to improve and justify the artifacts built in the other part of the process.

4.2 Adapted Design Science Research Approach

In order to achieve the expected result of finding a data storage solution which is well suited to store and manage Automation Markup Language (AML) model data (see Section 1.3), a methodological research approach has to be defined and followed. The research approach of this thesis is based on the *Design Science* approach [31] as described in Section 4.1. The main goal of the *Design Science* research approach is to develop and evaluate artifacts and their design and to improve their functional performance by iterating over them following the IS process cycle (see 1 in Figure 4.1). The resulting artifact of this thesis is the design of a prototypical benchmark which will be used for the performance evaluation of different data storage solutions. The research process as well as the resulting artifact are based on the *Environment* and the *Knowledge Base*. The *Knowledge Base* itself is defined by the current state of the art. Figure 4.2 shows the *Design Science* approach used in this thesis in detail. The environment part illustrates the *business needs* which are relevant to the research process, including the requirements of the stakeholders (i.e. find a well suited data storage solution for a central Multidisciplinary Engineering (MDE) model data store) as well as the technology and its capabilities currently used (i.e. current situation of exchanging AML files without a central data store) by the stakeholder (see Section 1.1). The *Knowledge Base* provides input to the research process (i.e. research questions and evaluation criteria) based on known scientific theories (i.e. related work)

These two parts are used in the research process in order to achieve the expected result of finding deeper insight which can be used to determine a well suited data storage solution for a central AML model data store (as defined Section 1.3).

The research process can be divided up into three parts that are described in the following listing:

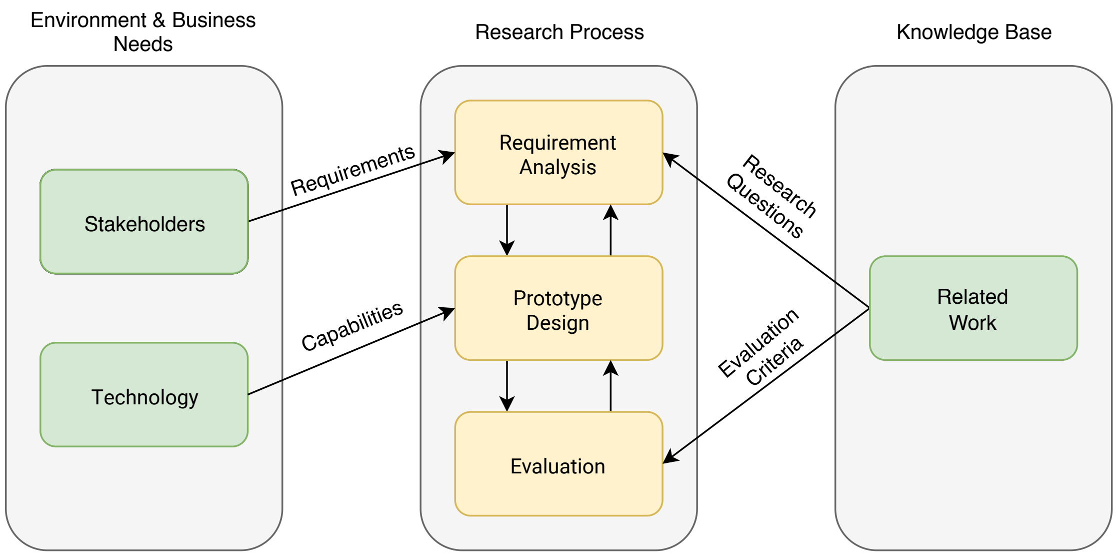


Figure 4.2: *Design Science* Approach of this Thesis based on [31]

Requirement Analysis Based on the *business needs* and the research questions (defined in Section 3) a requirement analysis to elicit the needs of the prototype design and the evaluation will be executed. This includes a description and analysis of the use cases, which form the basis for the performance evaluation. Furthermore, based on common data storage paradigms and criteria provided by the characteristics of the AML model data format, a selection of data storage paradigms will be executed. These data storage paradigms are then used for the prototype design and performance evaluation executed in this thesis.

Prototype Design for Evaluation In order to evaluate the selected data storage paradigms used for storing and querying AML model data, a prototype will be designed. This prototypical design includes a software architecture that allows for persisting data independently from the underlying database technology. Therefore, the underlying database solution can be flexibly exchanged. Additionally, a performance benchmark component will be designed which can be used to execute a performance evaluation.

Evaluation Furthermore, an evaluation on the suitability of the selected data storage paradigms for storing and querying AML model data will be executed. This performance evaluation will use the prototype design as well as the defined use cases in order to create performance results. In particular, these performance results of the selected data storage solutions will be compared and a detailed explanation will be given about which paradigms should be used for which use case. This leads to the conclusion of which data storage solution is better suited to store and query AML model data.

CHAPTER 5

Evaluation Use Cases

This chapter defines the required use cases, which frame the context and are relevant for the execution of the performance evaluation. The first part of this chapter, Section 5.1 provides an overview on the use cases used in this thesis which build a foundation for the performance evaluation. Furthermore, Section 5.2 illustrates and explains the execution process of several use cases.

5.1 Use Case Definitions

As stated in Section 2.4, a benchmark needs use cases as a basis for its execution. Therefore, relevant use cases have to be defined beforehand to afterwards structure the performance benchmark design accordingly. Use cases can derive from the needs of relevant stakeholder of the domain (e.g. Multidisciplinary Engineering (MDE) domain). Additionally, use cases can vary widely between more complex use cases, which represent real world processes, and more standard use cases, which represent technical approaches. Furthermore, use cases can derive from operations and processes which are common in the area of research e.g. steps required in the planning process of a plant. As use cases raised by domain experts usually focus on the business side of the particular problem, use cases from the area of research put their focus more on the theoretical side. Use cases relevant to this thesis are based on common data storage operations such as creating, accessing, updating and deleting data. Therefore, the use cases which provide a foundation for the performance evaluation focus on the so-called create, read, update and delete operations (CRUD) [34]. More complex use cases, such as filtering the data or checking data consistency, are subject to future work (Section 9.2) as specific requirements and examples of industry partners need to be elicited.

Therefore, the following list of uses cases was selected using a subset of Automation Markup Language (AML) model concepts [25]. For each of these model concepts, use cases focusing on either create, read, update or delete operations are defined.

- ***AMLFile***

  A user should be able to create, read, update or delete an *AMLFile* in the database, provided in an Extensible Markup Language (XML) format. This is, e.g., the case when an engineer creates the initial *AMLFile* for a Production Systems Engineering (PSE) project.

- ***InstanceHierarchy***

  A user should be able to create, read, update or delete an *InstanceHierarchy* of a specified *AMLFile* in the database. For example, adding new machines to a production line would involve the creation of an *InstanceHierarchy*.

- ***InterfaceClass***

  A user should be able to create, read, update or delete an *InterfaceClass* of an *InterfaceClassLib* of a specified *AMLFile* in the database. In case, e.g., an engineer wants to add a new connection between machines in a production line (e.g. new energy or data flow) a new *InterfaceClass* is created.

- ***InterfaceClassLib***

  A user should be able to create, read, update or delete an *InterfaceClassLib* of a specified *AMLFile* in the database. For example, adding new connections between machines in a production line requires an corresponding entry in an *InterfaceClassLib*.

- ***InternalElement***

  A user should be able to create, read, update or delete an *InternalElement* of an *InstanceHierarchy* of a specified *AMLFile* in the database. As *InternalElements* represent real world objects (e.g. robots or pumps), adding objects to a production line or a specific machine requires, e.g., the creation of an *InternalElement*.

- ***RoleClass***

  A user should be able to create, read, update or delete a *RoleClass* of a *RoleClassLib* of a specified *AMLFile* in the database. For example, engineers update *RoleClasses* when additional ports are added to a single devices role.

- ***RoleClassLib***

  A user should be able to create, read, update or delete a *RoleClassLib* of a specified *AMLFile* in the database. As, e.g., plant entities are described with specific roles, a *RoleClassLib* is updated in case a new device role is added.

- ***SystemUnitClass***

  A user should be able to create, read, update or delete a *SystemUnitClass* of a *SystemUnitClassLib* of a specified *AMLFile* in the database. *SystemUnitClasses* are used to describe plant entities including all its properties and relations and therefore, for example, adding adding a new plant entity requires the execution of this use case.

- ***SystemUnitClassLib***

  A user should be able to create, read, update or delete a *SystemUnitClassLib* of a specified *AMLFile* in the database. In case, e.g., a *SystemUnitClass* is added or updated, the according *SystemUnitClassLib* has to be changed as it is a library containing *SystemUnitClasses*.

These use cases are defined using natural language. The advantage of a natural language is that every stakeholder, no matter how high a person's technical knowledge is, can define and understand these use cases. The disadvantage is that they have to be transferred into a machine-readable format (e.g. source code) by a human. In order to solve this problem, a conceptual language, like Unified Modeling Language (UML) activity diagrams[1] or Petri-nets can be used that is interpreted by a corresponding tool or framework, like Foundational UML (fUML) [2] [36]. However, the focus of this thesis was rather on the benchmark than the use case descriptions. Therefore, the use case definitions are provided using natural language. However, selected use cases are introduced in more detail using flow diagrams (see Section 5.2).

For a better identification in later sections of this thesis, Table 5.1 numbers the use cases used in the performance benchmark.

	Create	Read	Update	Delete
InstanceHierarchy	C-IH	R-IH	U-IH	D-IH
InterfaceClass	C-IC	R-IC	U-IC	D-IC
InterfaceClassLib	C-ICL	R-ICL	U-ICL	D-ICL
InternalElement	C-IE	R-IE	U-IE	D-IE
RoleClass	C-RC	R-RC	U-RC	D-RC
RoleClassLib	C-RCL	R-RCL	U-RCL	D-RCL
SystemUnitClass	C-SUC	R-SUC	U-SUC	D-SUC
SystemUnitClassLib	C-SUCL	R-SUCL	U-SUCL	D-SUCL
AMLFile	C-AML	R-AML	U-AML	D-AML

Table 5.1: Performance Benchmark Use Cases

[1]Unified Modelling Language `uml.org`
[2]Foundational UML `omg.org/spec/FUML/About-FUML`

5.2 Use Case Execution Workflows

The identification of the use cases listed in the Table 5.1 is not sufficient to implement them using a programming language. Therefore, the following subsections provide a more detailed look at use case examples and state their input data. These use cases were selected as they representative for the main data operations (create, read, update and delete). Furthermore, these use cases cover AML model which are commonly used and are included in a representative manner in the performance evaluation input data (see Section 7.3.1).

5.2.1 Use Case C-RC: Create *RoleClass*

Figure 5.1 displays the execution steps–in the form of a flow diagram–needed to create a *RoleClass* element in the data store. The inputs to this use case are the name of the *AMLFile* and the name of the *RoleClassLib* as well as the *RoleClass* itself (see Section 2.2.3). These inputs are used during the execution of the use case and are mapped to the actual execution steps (marked orange). By using the *AMLFile* name, the first step (labelled with 1 in Figure 5.1) is to find the intended *AMLFile* in the data store. In case the database does not find an *AMLFile* with the provided name, an error is thrown stating that no *AMLFile* with the given *AMLFile* name is stored in the data store and nothing will be created (5). The *AMLFile* will be analyzed for the existence of an according *RoleClassLib* with the provided *RoleClassLib* name (2). In case the database does not find a *RoleClassLib* with the provided name, an error is thrown stating that no *RoleClassLib* with the given *RoleClassLib* name was found and nothing will be created (5). If a *RoleClassLib* is found, the database looks whether a *RoleClass* with the particular name already exists (3). If so, the database throws an error stating that a *RoleClass* with the particular name already exists and nothing will be created (5). Otherwise, the *RoleClass* will be added to the *RoleClassLib* and stored in the data store (4). In any case, the execution terminates.

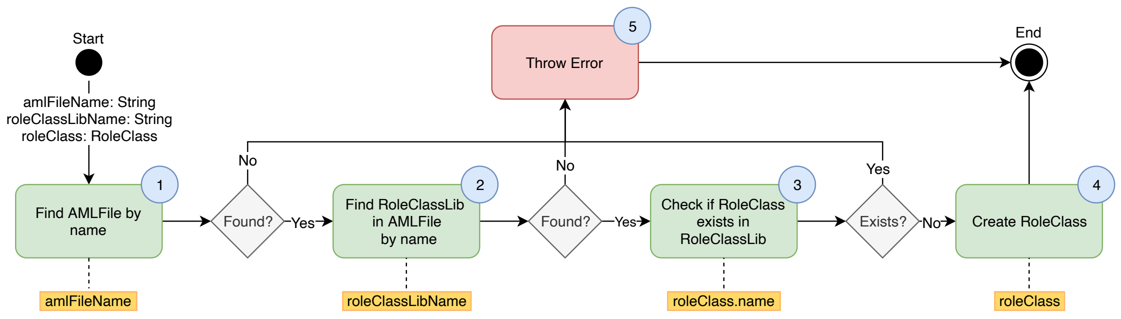


Figure 5.1: Creation of a RoleClass

5.2.2 Use Case R-IE: Read Internal Element

Figure 5.2 shows the execution sequence of the read operation for an *InternalElement* from a data store. The inputs to this use case are the name of the *AMLFile* and the *InstanceHierarchy* in which the *InternalElement* is located as well as the name of the *InternalElement* itself. As in the first use case flow diagram, the inputs used during the execution of the use case are mapped to the actual execution steps (marked orange). The first step is to locate the*AMLFile* with the given name in the data store, labelled with 1 in Figure 5.2. In case the *AMLFile* is not found, the database throws an error stating that there is no *AMLFile* with the given *AMLFile* name in the data store (5). If the *AMLFile* is found, the data store will be searched for an *InstanceHierarchy* using the given *InstanceHierarchy* name (2). If there is no suitable *InstanceHierarchy*, the database throws an error stating that there is no *InstanceHierarchy* stored in the data store under the *AMLFile* with the *AMLFile* name and the given *InstanceHierarchy* name (5). If the *InstanceHierarchy* possesses an *InternalElement* with the given *InternalElement* name, the *InternalElement* will be returned (4); otherwise, the database throws an error stating that there is no *InternalElement* with the given name (5). In any case, the execution terminates.

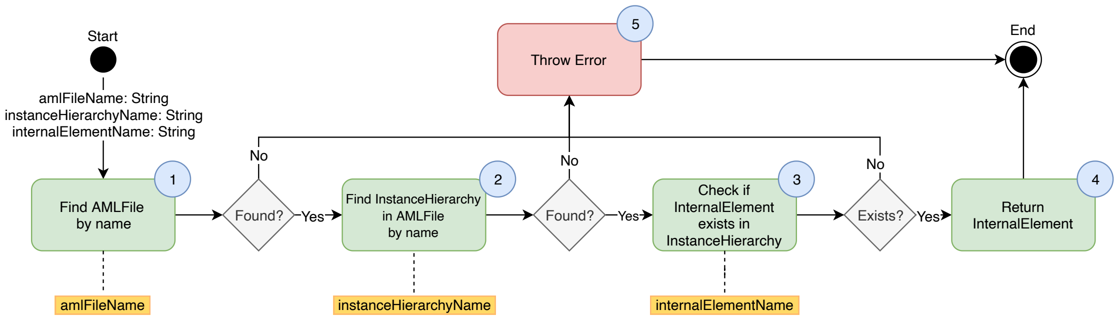


Figure 5.2: Read *InternalElement*

5.2.3 Use Case U-SUCL: Update *SystemUnitClassLib*

Figure 5.3 shows the execution sequence which is needed to update a SystemUnitClassLib. For this use case execution, an *AMLFile* name in which the *SystemUnitClassLib*, that needs to be updated is stored and the *SystemUnitClassLib* itself is needed. As before, the inputs are mapped to the actual use case execution steps and marked in orange. Based on this input, the *AMLFile* must be found in the data store based on the given *AMLFile* name (1). In case the *AMLFile* is not found, an error is thrown stating that there is no *AMLFile* stored with the given *AMLFile* name (4). In case the *AMLFile* is found, the data store will be checked if a *SystemUnitClassLib* already exists with the name of the *SystemUnitClassLib* (2). If a *SystemUnitClassLib* exists, the given *SystemUnitClassLib* in the data store will be updated based on the given *SystemUnitClassLib* (3). If there is no *SystemUnitClassLib*, an error will be thrown stating that there is no *SystemUnitClassLib* based on the name of the *SystemUnitClassLib* (4). In any case, the execution finishes.

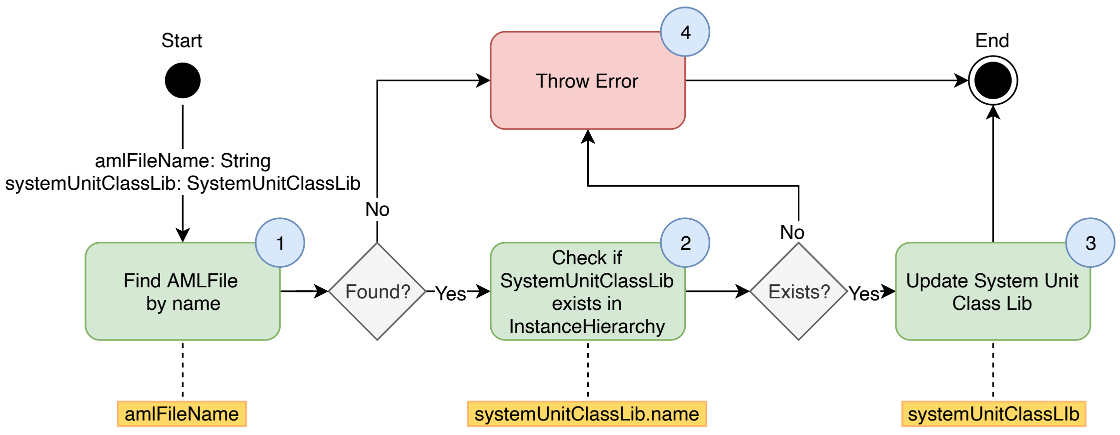


Figure 5.3: Update *SystemUnitClassLib*

5.2.4 Use Case D-AML: Delete *AMLFile*

Figure 5.4 shows the execution sequence of deleting an *AMLFile*. The input to the execution is a string variable containing the name of the *AMLFile*. Its usage is shown by the orange box connected to the consuming use case execution step. Based on this input, the data store must be searched for the *AMLFile* based on the given file name (1). In case a file is found, the complete file and all its connected sub elements are deleted (2) and the execution is finished. In case the file is not available in the data store, an error is thrown stating that no file with the given file name was found (3). Both of these steps result the execution to end.

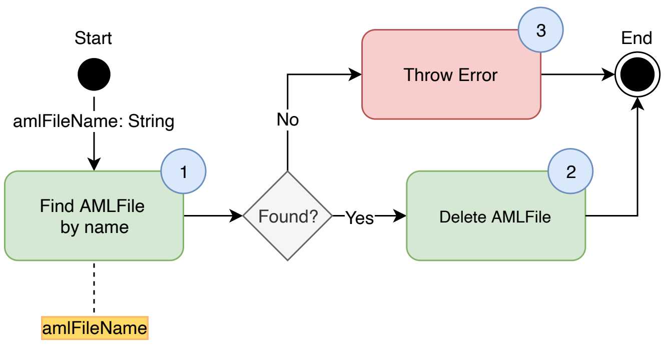


Figure 5.4: Delete *AMLFile*

Summary

This chapter defined the use cases which are relevant for the performance evaluation of the proposed solution design presented in this thesis. This was done by elaborating the scope of the uses cases (i.e. typical data operations such as create, read, update and delete) and by mapping this scope to relevant AML model concepts. The result of this process is a list of use cases (see Table 5.1). In addition, example use cases were further explained in detail using flow diagrams. The prototypical design–introduced in Chapter 7–utilizes these explanations as well as the created flow diagrams in the implementation process.

CHAPTER 6

Benchmark Evaluation Architecture and Design

This chapter introduces and describes the design of the evaluation infrastructure including the most relevant components, aiming at addressing the research questions in Chapter 3. Therefore, Section 6.1 explains the underlying software architecture of the performance evaluation infrastructure. In addition, Section 6.2 presents the benchmarking component of this prototype design and discusses the Java Microbenchmark Harness (JMH) framework which is used as a benchmarking framework for the performance evaluation.

6.1 Benchmark Evaluation Architecture

In order to follow the *Design Science* methodology and to address the research questions stated in Chapter 3, a design for a performance evaluation infrastructure has to be established. This design will be provided in the form of a prototypical infrastructure. This section provides an architectural overview of this infrastructure as well as information about the infrastructure's core components.

The architectural design of the performance evaluation infrastructure follows a layered software architecture pattern where different components build upon each other [45]. The layered software architecture pattern focuses on the separation of concerns, meaning that the layers provide a strict separation in terms of bundling common functionalities [52]. Therefore, layers only utilize concepts of their own or the layer below. Additionally, Richards proposes other software architecture patterns such as the event-driven architecture or microservices architecture pattern [45]. The event-driven architecture pattern is used for distributed systems which are event-driven and asynchronous. It consists of decoupled event processing components that receive and process events in an asynchronous manner. The software architecture for this performance evaluation does not need to be event-driven nor does the evaluation consist of events which can occur asynchronously. Therefore, the event-driven pattern is not a suitable architecture choice. The microservices architecture pattern is also not suitable for this performance evaluation infrastructure either as this pattern focuses on services which provide functionality to other services in the architecture. This would not be beneficial for the performance evaluation as the intended layers build on top of each other and are not decoupled services which communicate via defined endpoints with one and another. This leads to the fact that the layered software architecture pattern seems most relevant based on the needs of this thesis.

Figure 6.1 shows the architecture of the performance evaluation infrastructure with all its key components. The base of the performance evaluation architecture is built by a virtualization layer (Layer 1). A virtualization layer is an additional layer (on top of the operating system in which the virtualization layer runs) that manages access to resources such as processing power and storage for the virtual machines (e.g. Docker container), which are located inside this layer [55]. In this particular case, Docker[1] is used. It holds the different data storage solutions that are evaluated. This layer will be further discussed in Section 6.1.4. The next layer in the architecture, on top of the virtualization layer, is a *Managing Component* (Section 6.1.3) for each data store which implements all its specific functions (e.g. setup and shutdown methods) as well as the use cases defined in the *Common API (Application Program Interface)* (see Section 6.1.2) (Layer 2). The *Common API* defines all use cases (see Section 5.1) that were selected for the execution of the performance evaluation (Layer 3). APIs provide only declarations of the provided methods whereas implementations of the interface hold the actual functionality (e.g. steps) needed to execute the provided methods. Furthermore, a data model which represents the Automation Markup Language (AML) model in the form of Java classes was designed (Section 6.1.1) (Layer 4). The data model component is designed by defining the AML model interface which contains relevant AML elements (as shown in Section 2.2.3). In addition, an implementation of the AML model interface is required. The performance benchmark is implemented in a separate component (5), orthogonal to the layered architecture (see Section 6.2). It uses the *Common API* interface as well the AML model classes to transparently communicate to the databases for the benchmark execution.

The layered approach allows to exchange the underlying data storage solution and the corresponding *Managing Component* with minimal effort in order to be able to benchmark different data storage solutions. The following subsections describe the core components in detail.

[1]Docker Virtualization Software: https://www.docker.com/

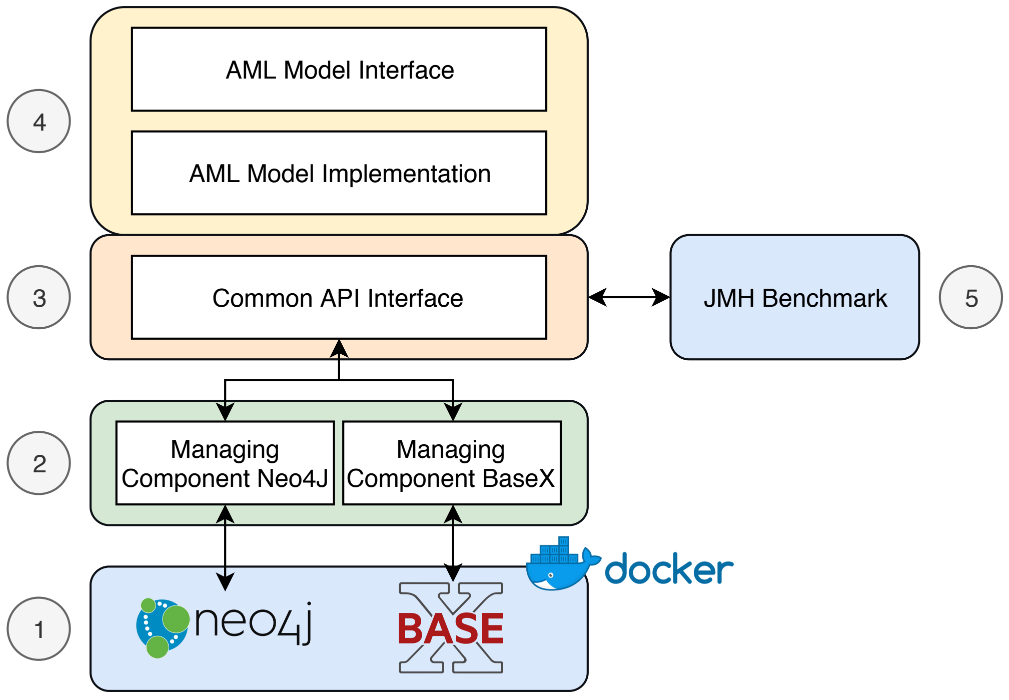


Figure 6.1: Architecture Overview of the Performance Evaluation Infrastructure

6.1.1 AutomationML Model

AML is a data exchange format used in the context of Multidisciplinary Engineering (MDE) [18] and Production Systems Engineering (PSE) [8]. In addition, AML can also be used as a model for engineering data as it is capable of representing engineering data of, for example, a car production plant. In order to handle such an AML model in the performance evaluation infrastructure, it requires a representation of the AML model in the particular programming language used for the architecture implementation. In this case, Java was used to implement the benchmark, therefore, requiring a representation of the AML model in Java. Java is an object-oriented programming language which, similarly to other languages of this type, allows to represent models of real-world entities (e.g. machines) as Java object instances. A Java class can be seen as a blueprint of Java instances containing all its variables, relationships as well as method definitions (e.g. functions). Variables can act as containers which hold values in the form of defined data types (such as String for literals) for the execution of a Java program. Relationships in the context of a Java class are connections to other Java classes in the form of class or object variables. For example, the connection between a car (as a Java class) and its wheels (another Java class) can be seen as a relationship. Methods or functions combine a collection of statements which perform a specific task (e.g. *startEnigne* of a car). Therefore, the AML model concepts can be transferred to Java classes. For the purpose of this thesis, an AML model interface–defined by Rosendahl et al. [47]–was used containing all relevant AML model concepts. In addition to the interface, the actual implementations of the interface were also provided. These implementations are used throughout the whole performance evaluation infrastructure.

The execution of the performance benchmark uses an AML file in the Extensible Markup Language (XML) format as input data. In order to bring this XML file into the format of a the AML model interface (i.e. Java objects) a transformation is needed. Java provides a solution for transforming XML files into Java objects: the Java Architecture for XML Binding (JAXB) [22]. This framework maps XML data like AML files to Java objects. In order to enable the transformation process, the Java classes are annotated which instructs JAXB how to map the information stored in an XML file into a Java object. Listing 2 shows an example of a *CAEXFile* Java model class annotated with JAXB annotations. The annotation *@XMLType* shown in Listing 2 line 1 states the type of the XML element e.g. *<CAEXFile></CAEXFile>* in the XML file and *@XMLRootElement* (line 2) states that this XML element can be a root element. XML documents have exactly one root element which encapsulates all other elements. Therefore, it is the parent element to every other element contained in the document. Furthermore, the *@XMLElement* annotation (line 8) is used to map methods (e.g. *getter* and *setter* methods) or class fields to according XML elements.

```
 1  @XmlType(name = "CAEXFile")
 2  @XmlRootElement(name = "CAEXFile")
 3  public class CAEXFile extends CAEXBasicObject implements ICAEXFile {
 4
 5      //class fields
 6      .....
 7
 8      @XmlElement(name = "References", type = ExternalReference.class)
 9      public List<IExternalReference> getExternalReferences() { ... }
10  }
```

Listing 2: Java XML Entity Annotation

The process of transforming XML files into Java objects is called unmarshalling or deserialization [22]. Listing 3 shows the example code used to transform an AML file provided with its file path into a *CAEXFile*. Due to the annotations in the Java classes and their JAXB interpretation, all relationships (e.g. object variables) are also transformed.

JAXB annotations can be used for XML elments and are therefore useful for XML based data stores. Nevertheless, other databases–such as Neo4J–need different annotations in order to be able to store and handle entities directly. The standard framework which implements persistence mappings (e.g. annotations) in Java is called Java Persistence API (JPA)[2] [11]. This framework allows to annotate model classes and makes them

[2] Java Persistence API
`oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html`

```java
    JAXBContext jc = JAXBContext.newInstance(CAEXFile.class);
    Unmarshaller unmarshaller = jc.createUnmarshaller();
    File xml = new File(fileName);
    CAEXFile file = (CAEXFile) unmarshaller.unmarshal(xml);

    Marshaller marshaller = jc.createMarshaller();
    marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
    marshaller.marshal(file, System.out);
    return file;
```

Listing 3: Java XML Marshalling

usable for databases for which an *Object Mapper* exists. This *Object Mapper* provides the main functionality for the conversion of either to and from data formats (such as XML) to Java objects. In addition, internal structures of Java objects can be mapped to database tables (Relational Database), documents (Non Structured Query Language (NoSQL) Database) and graphs (Graph Database). Listing 4 shows an example AML model class annotated using JPA annotations. *@Entity* states that this class is interpreted as an entity, which can be stored in the database. *@Id* defines that the database generates an (internal) ID for this entity and the *@OneToMany* annotation defines the relationships to other entities. *@OneToMany* provides a relationship mapping where one row in a table (represented by a Java object) is mapped to multiple rows in another Table (represented by other Java objects).

```java
@Entity
public class CAEXFile extends CAEXBasicObject implements ICAEXFile {
    @Id
    @GeneratedValue
    @OneToMany(targetEntity = InstanceHierarchy.class, cascade =
     ↪  CascadeType.ALL)
    protected List<IInstanceHierarchy> instanceHierarchies;

    ...
}
```

Listing 4: Java JPA Entity Annotation

For the purpose of this thesis, a combination of JAXB and JPA annotations can be used in order to support multiple database solutions. The main benefit for using two different types of annotation in one data model is that an additional data model (i.e. one data model for each specific data storage solution) is obsolete. This allows for an easier handling in the prototypical design as no further transformation or mapping, like Data Transfer Object (DTO), between two data model entities is required. Nevertheless, mapping two different annotations is not always a viable solution, e.g., when two different

Java libraries try to inject additional functionality into the bytecode of a single class.

6.1.2 Common API

The *Common API* provides the interface definitions (see Section 6.1) of all use cases as defined in 5.1. According to Meng et al. [38], an API (Application Program Interface) provides a set of functionalities usually exposed in the form of methods or objects. This interface is implemented by a *Managing Component* of an underlying data storage solution. Due to the fact that different data storage paradigms and solutions use different query languages and data base providers, it is necessary to implement the managing component layer for every storage solution.

The methods contained in the *Common API* interface are listed in Table 6.1. As these methods represent the use cases of this performance evaluation, the according use case ID is also listed.

Create	UC ID	Read	UC ID
createCAEXFile	C-AML	readCAEXFile	R-AML
addInstanceHierarchy	C-IH	readInstanceHierarchy	R-IH
addSystemUnitClassLib	C-SUCL	readSystemUnitClassLib	R-SUCL
addRoleClassLib	C-RCL	readRoleClassLib	R-RCL
addInterfaceClassLib	C-ICL	readInterfaceClassLib	R-ICL
addInternalElement	C-IE	readInternalElement	R-IE
addSystemUnitClass	C-SUC	readSystemUnitClass	R-SUC
addRoleClass	C-RC	readRoleClass	R-RC
addInterfaceClass	C-IC	readInterfaceClass	R-IC
Update		Delete	
updateCAEXFile	U-AML	deleteCAEXFile	D-AML
updateInstanceHierarchy	U-IH	deleteInstanceHierarchy	D-IH
updateSystemUnitClassLib	U-SUCL	deleteSystemUnitClassLib	D-SUCL
updateRoleClassLib	U-RCL	deleteRoleClassLib	D-RCL
updateInterfaceClassLib	U-ICL	deleteInterfaceClassLib	D-ICL
updateInternalElement	U-IE	deleteInternalElement	D-IE
updateSystemUnitClass	U-SUC	deleteSystemUnitClass	D-SUC
updateRoleClass	U-RC	deleteRoleClass	D-RC
updateInterfaceClass	U-IC	deleteInterfaceClass	D-IC

Table 6.1: Declared Methods of *Common API* Interface

In addition to the declared methods of the *Common API* interface (see Table 6.1), a base interface is defined which holds relevant database methods such as setup, shutdown and delete content of the database. These methods combine basic database operation and configuration methods and are therefore defined in their own interface in order to keep them separated from the *Common APIs* main interface definition which holds the use case methods.

6.1.3 Managing Component

Use cases serve as a basis for the performance benchmark (see Section 5.1). In order to use them in the performance benchmark, these use cases have to be defined in a machine-readable format to be translated to a software program. This is done by implementing the use cases stated in the *Common API* in a *Managing Component*. As all data storage solutions differ in their ways of establishing the connection and data modification and access, this common interface and the according *Managing Component* ensure that the use cases can be used for a performance benchmark execution. Figure 6.2 shows the implementations of the *Common API* used in the performance benchmark. The two data storage solutions used are: BaseX as a representative of the semi-structured data storage paradigm and Neo4J as a representative of the graph data store paradigm. Therefore, the two *Managing Components* are named *CommonAPIBaseX* and *CommonAPINeo4J*.

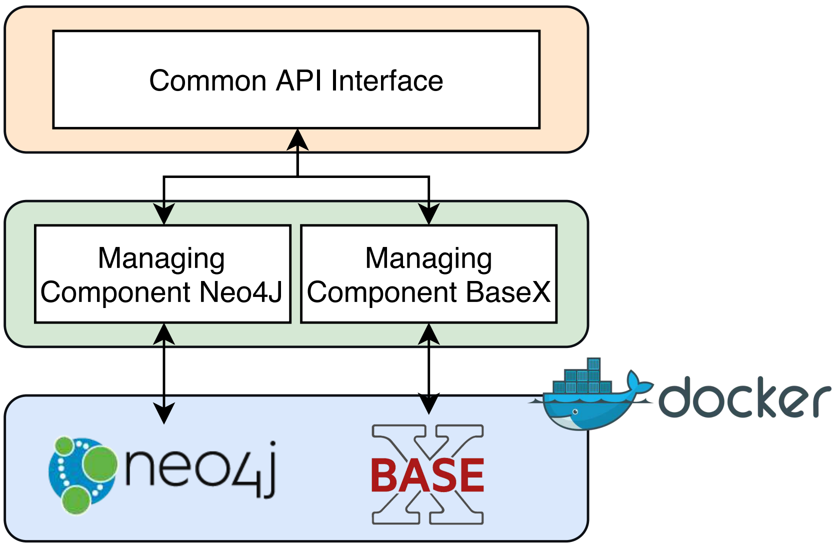


Figure 6.2: Implementations of *Common API*

The following subsections describe these two implementations in detail.

CommonAPIBaseX

The *CommonAPIBaseX* is an implementation of the *Common API* interface and serves as the *Managing Component* for BaseX. It includes BaseX specific source code such as database access methods and implementations of the defined use cases. As stated in Section 2.3.1, BaseX is a database with the main purpose of storing XML documents. In order to manipulate data stored in a BaseX database, one can execute XML Query Language (XQuery) queries which allow for an efficient manipulation of the data. In general, a connection (e.g. a client) to a BaseX data base is needed for the execution of the XQuery queries. As the programming language used for the implementation of the performance evaluation is Java, a Java-based data base client is required. For this purpose, BaseX provides a blueprint Java class (i.e. *BaseXClient*) to access a BaseX instance[3]. This *BaseXClient* offers methods to execute XQuery queries on the data store as well as default methods to add or remove complete XML files. Therefore, the execution of the use cases does not always require a XQuery statement (e.g. creating a *AMLFile*). The following subsections provide information about different data base operations which are used during the execution of the evaluation use cases. Additionally, XQuery examples for each proposed data base operation are shown.

Reading from a BaseX DB

The use case description of the Read use case R-IE from Section 5.2.2 shows that several steps are required. For example, during the creation process of a *RoleClass* (use case C-RC), the use case sequence requires two read operations from the data store (see Figure 5.1). In order to create a *RoleClass*, the according *AMLFile* and *RoleClassLib* have to be loaded from the BaseX database by executing the XQuery statement which is shown in Listing 5. The *collection()* call loads the BaseX database according to the given database name (*DBNAME*) and queries the database for a CAEX File (*AMLFile*) with a given file name (*AMLFILE_NAME*).

```
xquery collection('DBNAME')/CAEXFile[@FileName='AMLFILE_NAME']
```

Listing 5: XQuery: Read AMLFile

Similar to this XQuery method, other elements are also loaded from a BaseX database by querying through the hierarchy of an AML file.

Updating Elements in a BaseX DB

In addition to reading elements from the database, it is also necessary to update elements. The example use case in Section 5.1, use case U-SUCL, is the process of updating a *SystemUnitClassLib*. As seen in Figure 5.3, this execution consists of a step where the provided *SystemUnitClassLib* is updated in the data store. In order to do this in the BaseX database that uses Java, the XQuery query shown in Listing 6 is executed. The *$node* variable holds the System Unit Class Lib with the given name (*SYSTEMUNITCLASSLIBNAME*) stored in the database (*DBNAME*) under an AML file (*AMLFILE_NAME*). By calling the *replace node* function and replacing it with the provided System Unit Class Lib (*SYSTEMUNITCLASSLIB*)–which is serialized back from a Java object to an XML string–the *$node* is replaced with the new System Unit Class Lib.

The same method structure is used for updating other AML model concepts.

[3]Java Examples - BaseX Documentation `docs.basex.org/wiki/Java_Examples`

```
xquery let $node := db:open('DBNAME')/CAEXFile[@FileName =
'AMLFILE_NAME']/SystemUnitClassLib[@Name='SYSTEMUNITCLASSLIBNAME']
return replace node $node with " +
serializeOutputStream(SystemUnitClassLib.class, SYSTEMUNITCLASSLIB)
```

Listing 6: XQuery - Update System Unit Class Lib

Deleting Elements in a BaseX DB

In order to delete elements in a BaseX data store, it is also necessary to execute XQuery queries. For example, use case D-AML executes the deletion process of an AMLFile (see Section 5.1). Figure 5.4 shows the execution steps where the last step is to delete the AML file. The execution requires the XQuery query shown in Listing 7. It executes the same *open* command used in the updating process to load the corresponding AML file (AMLFILE_NAME) from the database (DBNAME). Deleting this element requires the execution of the XQuery command *delete node*.

```
xquery delete node
db:open('DBNAME')/CAEXFile[@FileName = 'AMLFILE_NAME']
```

Listing 7: XQuery - Delete AML File

The deletion of other AML model concepts follows the same principle and uses similar XQuery queries.

Adding Root Elements in a BaseX DB

The BaseX Java Client by BaseX[4] provides different methods which allow for the execution of XQuery queries as well as adding XML files directly to the database. The latter feature is used to create the root element–an AML file–in the database as it is the root element in the XML file and therefore it is possible to utilize this feature. Listing 8 shows the method call where a Java object named *CAEXFILE* is added to the data base. The *add* command requires a name of the XML file–in this case the *CAEXFILE* name is used–and a string representation of the *CAEXFILE*. Therefore, the provided AML File gets serialized before being passed on to the *add* method.

```
baseXClient.add(CAEXFILE.getFileName(),
serialize(CAEXFile.class, CAEXFILE));
```

Listing 8: Creation Process AML File

[4] Java Examples - BaseX Documentation `docs.basex.org/wiki/Java_Examples`

CommonAPINeo4J

The *CommonAPINeo4J* is an implementation of the *Common API* interface and represents the *Managing Component* for the Neo4J database. Similar to the *CommonAPIBaseX*, all Neo4J specific source code needed to execute the use cases is implemented in this component of the performance evaluation. Neo4J itself provides several access methods when using the Java programming language[5][39]. One of the main goals of this prototype design is to provide an extensible infrastructure which can easily support other data storage solutions in the future to evaluate whether they are better suited to store AML model data. Therefore, a common solution which can be used by several data storage solutions is preferred. The selected implementation uses the Hibernate Object Graph Mapping[6](Hibernate OGM) framework which utilizes JPA annotated Java entities (e.g. the implementations of the AML model concepts)[11]. As JPA annotated model entities can be used by several other data storage solutions, it–in conjunction with Hibernate as an object-relational mapping framework–is well suited for this implementation. The process of how JPA annotations are used in this implementation is shown in Section 6.1.1.

In order to connect to the Neo4J data store, JPA and Hibernate use so-called persistence units which hold important properties such as name and host of the database as well as access credentials. This information is then used to establish a connection to the database and instantiate relevant Hibernate components such as the *EntityManager* and the *TransactionManager*. According to Biswas and Ort [11], the *EntityManger* is used to manage states and life cycles of Java entities in the context of this persistence unit (e.g. the database). Furthermore, the *TransactionManager* is used to handle transactions (e.g. commits of data changes) issued to the data store.

Listing 9 shows a code snippet of the setup process that establishes a working connection with the Neo4J data store through JPA and Hibernate.

```java
public void setUp() {
    emf = Persistence.createEntityManagerFactory("ogm-neo4j");
    transactionManager = extractTransactionManager(emf);
    transactionManager.begin();
    entityManager = emf.createEntityManager();
}
```

Listing 9: Setup Process for JPA/Hibernate Neo4J

In order to be able to execute the use cases on the Neo4J data store, the two Hibernate components *EntityManager* and *TransactionManager* are used. Furthermore, querying the database is done by executing Cypher queries (see Section 2.3). The following

[5]Neo4J for Java `neo4j.com/developer/java`
[6]Hibernate Object Graph Mapping Framework: `https://hibernate.org/`

subsections provide examples of different operations on the database which are used in the execution process of the different use cases.

Reading from a Neo4J DB

As stated in Section 6.1.3, reading data from the data store is relevant for the execution of the use cases for the performance evaluation. In order to read data from the Neo4J database, executing a Cypher query is necessary. Reading an AML file from the data store is part of many use cases (e.g. use case C-IE) and therefore, Listing 10 shows the Cypher query needed to retrieve an AML file from the database. This query returns all *CAEXFiles* (e.g. AML files) which are a *CAEXFile* and have the provided file name (*AMLFILE_NAME*).

```
MATCH (c:CAEXFile) WHERE c.fileName = 'AMLFILE_NAME' RETURN n;
```

Listing 10: Cypher - Read AML File

Similar to the Cypher query in Listing 10, it is possible to retrieve other AML model elements stored in the database. Listing 11, for example, retrieves all SystemUnitClassLibs with the provided name (*SYSTEMUNITCLASSLIB_NAME*) inside an AML file with the appropriate name (*AMLFILE_NAME*). By mapping the *CAEXFiles* via their relationships *systemUnitClassLibs* to the SystemUnitClassLib entities, it is possible to solely filter for SystemUnitClassLibs that confirm to the provided *WHERE* statement.

```
MATCH (c:CAEXFile)-[r:systemUnitClassLibs]
->(i:SystemUnitClassLib)
WHERE i.name = 'SYSTEMUNITCLASSLIB_NAME' and
c.fileName = 'AMLFILE_NAME' RETURN i;
```

Listing 11: Cypher - Read System Unit Class Lib

Creating & Updating Elements in a Neo4J DB

In contrast to reading from the data store, creating and updating does not in itself require the execution of Cypher queries. Nevertheless, in order to create or update entities, a read operation is required before the creation or updating process can be executed. In case of a creation, the parent elements have to be loaded from the data store in order to modify and persist the entities back to the data store (except for creating an AML file which is a root element). In case of an update, after the to be updated entities are retrieved from the database, the Java objects representing an AML model concept are traversed until the part which will be updated is reached. The update is executed by modifying the Java object itself. After the modification, the Java object (for both the creation and the update) gets persisted by the *EntityManager* and in order to push the update to the database, the *TransactionManager*'s *commit* command will be executed. This command brings the modification of the Java object back into the data store.

Deleting Elements in a Neo4J DB

In addition to reading data from a Neo4J data store, the deletion of data also requires the execution of Cypher queries. Listing 12 shows the Cypher statement which is needed to delete an AML file including all its sub elements. By executing the matching command as shown in Listing 12, all *CAEXFiles* which match with the file name (*AMLFILE_NAME)* will be returned. Calling *DETACH DELETE* will traverse through all connected entities and delete the complete AML file including all sub elements.

```
MATCH (c:CAEXFile)
WHERE c.fileName = 'AMLFILE_NAME' DETACH DELETE c;
```

Listing 12: Cypher: Delete AML File

Similar to the process of deleting AML files using Cypher queries, other AML model concepts can be deleted. Using *detach delete* always deletes all sub elements of the main element.

6.1.4 Data Storage Solution

For the performance evaluation, the BaseX and Neo4J data stores have to be installed and configured. By using docker, it is possible to easily setup and shutdown the data stores on the local test machine and to separate the database engine from the respective system that the benchmark is executed on. Docker itself is a virtualization software for creating and running containers inside an operating system [13]. These containers are instances of Docker images, i.e. software bundles in which the required software is already installed. In order to avoid creating Docker images over and over again, Docker Hub [7] is used. Docker Hub is a remote repository containing multiple predefined images. Running a Neo4J instance and a BaseX instance in Docker requires the introduction of a Dockerfile which provide human readable specifications of the necessary software components [13]. Listing 13 shows the Dockerfile used to run a local Neo4J and BaseX instance. The images pulled from the Docker Hub repository run in a local Docker container. In the context of Docker, images are a set of layers, which can be defined by the user, and containers are instances of the defined images [13]. By defining the *ports* property, these network ports are exposed to the local machine and are subsequently accessible under *localhost*.

The following two subsections define the specifics of the two data stores.

[7]Docker Image Hub: https://hub.docker.com/

```
version: '3'
services:
  neo4j:
    image: neo4j:3.5.7
    restart: always
    ports:
    - "7474:7474"
    - "7687:7687"
  basex:
    image: basex/basexhttp:9.2.3
    restart: always
    ports:
      - "1984:1984"
```

Listing 13: Dockerfile for Neo4J and BaseX Data Store

BaseX

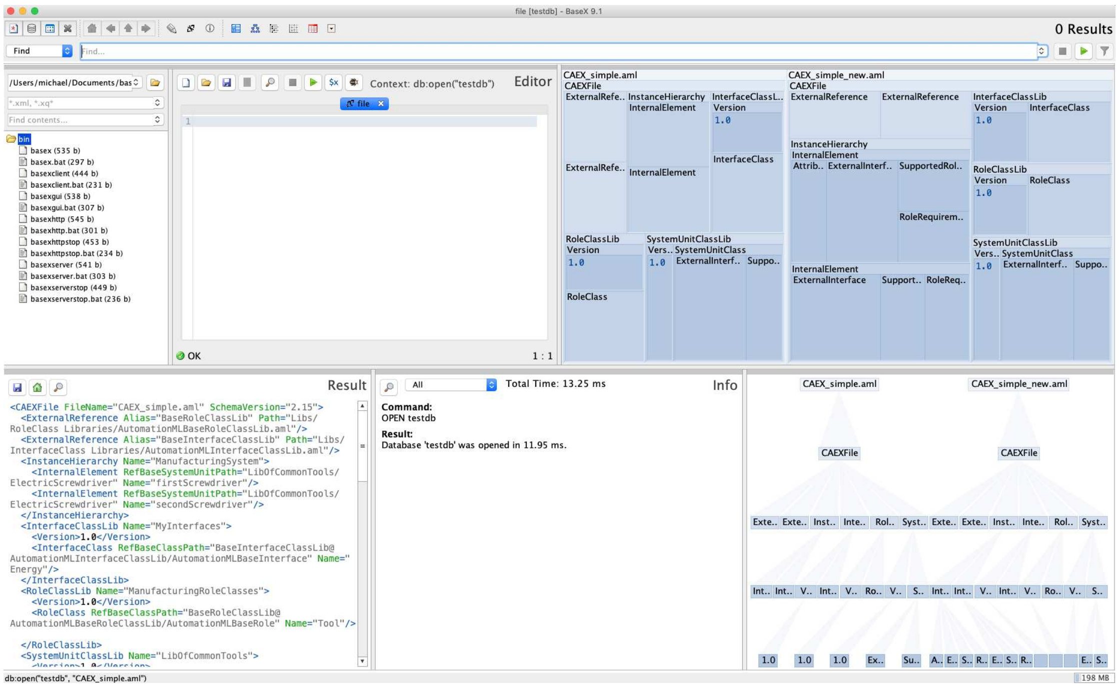
As the Dockerfile in Listing 13 shows, the *basexhttp* Docker image is pulled from the Docker Hub repository. By defining the version of the image, it is possible to request different versions of the BaseX data store. For the purpose of this thesis, Version *9.2.3* was used. Running this Dockerfile on the local machine exposes a BaseX database under `http://localhost:1984/`. This database can then be accessed in the *Managing Component* of BaseX (Section 6.1.3).

In addition to the data store server, accessing the database by using a graphical user interface enables easier access and development. Therefore, BaseX provides the *basexgui* which is a standalone user interface that can be used to visualize the data stored in the database. Furthermore, database queries can be executed. As this user interface is a standalone desktop program, it has to be started locally. This can be done by downloading the BaseX source package from BaseX [8]. This package includes several binaries (e.g. *basexhttp* server) and the *basexgui*. When running this *basexgui* binary, it opens a local desktop application which can be connected to the database that runs on *localhost*. Figure 6.3 shows a screenshot of the user interface with different data visualizations and the execution area for queries.

Neo4J

The Neo4J data store also runs in a Docker container. As shown in Listing 13, the Docker image *neo4j* is loaded from the Docker Hub repository, pulling Version *3.5.7*. By running the Dockerfile, a Neo4J database that can be reached under `http://localhost:7687/` is started. This running instance of a Neo4J database will be used by the *Managing Component* for Neo4J. In addition to starting the data base, the *neo4j* container also

[8]BaseX Source Package: http://basex.org/download/



```
<CAEXFile FileName="CAEX_simple.aml" SchemaVersion="2.15">
  <ExternalReference Alias="BaseRoleClassLib" Path="Libs/
RoleClass Libraries/AutomationMLBaseRoleClassLib.aml"/>
  <ExternalReference Alias="BaseInterfaceClassLib" Path="Libs/
InterfaceClass Libraries/AutomationMLInterfaceClassLib.aml"/>
  <InstanceHierarchy Name="ManufacturingSystem">
    <InternalElement RefBaseSystemUnitPath="LibOfCommonTools/
ElectricScrewdriver" Name="firstScrewdriver"/>
    <InternalElement RefBaseSystemUnitPath="LibOfCommonTools/
ElectricScrewdriver" Name="secondScrewdriver"/>
  </InstanceHierarchy>
  <InterfaceClassLib Name="MyInterfaces">
    <Version>1.0</Version>
    <InterfaceClass RefBaseClassPath="BaseInterfaceClassLib@
AutomationMLInterfaceClassLib/AutomationMLBaseInterface" Name="
Energy"/>
  </InterfaceClassLib>
  <RoleClassLib Name="ManufacturingRoleClasses">
    <Version>1.0</Version>
    <RoleClass RefBaseClassPath="BaseRoleClassLib@
AutomationMLBaseRoleClassLib/AutomationMLBaseRole" Name="Tool"/>

  </RoleClassLib>
  <SystemUnitClassLib Name="LibOfCommonTools">
```

```
All    Total Time: 13.25 ms

Command:
OPEN testdb

Result:
Database 'testdb' was opened in 11.95 ms.
```

```
db:open("testdb", "CAEX_simple.aml")
```

Figure 6.3: BaseX Graphical User Interface

starts the Neo4J Browser which is a user interface for the Neo4J database. Figure 6.4 shows the Neo4J browser which can be used to visualize data and execute queries.

Summary

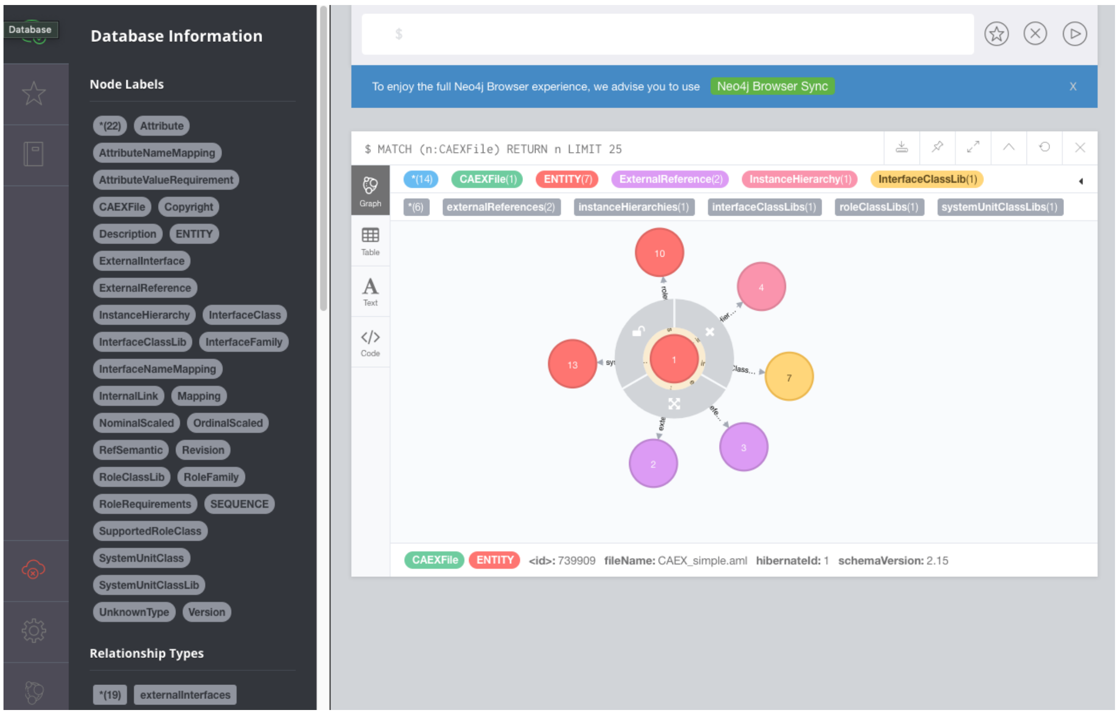
Section 6.1 explained the actual software architecture used in the prototypical design. Therefore, all core components such as the AML model, the *Common API*, the *Managing Components* as well as the data storage solutions BaseX and Neo4J where discussed.

6.2 Benchmark Evaluation Design

The benchmark design is an important part of the prototypical solution, as it holds the actual execution of the performance evaluation.

Figure 6.1 shows that the actual benchmarking part of the performance evaluation is separated from all other components. It utilizes the *Common API* interface in order to execute the performance evaluation (e.g. evaluating the use cases). In addition to all other components of the performance evaluation infrastructure, the benchmark itself was also implemented by using the Java programming language.

The aim of this section is to provide an overview on the design of this benchmark as well as the technologies and frameworks used. In particular, the architectural overview of the



```
$ MATCH (n:CAEXFile) RETURN n LIMIT 25
```

CAEXFile ENTITY <id>: 739909 fileName: CAEX_simple.aml hibernateId: 1 schemaVersion: 2.15

Figure 6.4: The Neo4J Browser

benchmark as well as the benchmarking framework and its problems will be explained. Furthermore, the process of how input data is handled inside the benchmark and the relevant performance indicators are described. Lastly, the format of the benchmark output (e.g. the results) will be explained.

6.2.1 Benchmark Architecture Overview

The benchmark is structured by using four Java classes for BaseX and four for Neo4J where each class contains benchmarks for one typical data storage operation such as Create, Update, Read, and Delete. Table 6.2 shows all benchmark classes and benchmark runners (benchmark executors) that form the benchmarking component. As shown in Table 6.2, each benchmark class is assigned a specific use case ID, which maps the benchmark to the use cases. In addition to the implementation classes, several relevant input files are stored inside a resource directory, which are accessed during the execution of the benchmark (see Section 6.2.3). The implementation itself is located inside the test directory of the performance evaluation infrastructure project.

Data Store	Data Storage Operation	Use Case ID
BaseX		
BaseXCreateBenchmark	Create	C-AML, C-IH, C-SUCL, C-RCL, C-ICL, C-IE, C-SUC, C-RC, C-IC
BaseXReadBenchmark	Read	R-AML, R-IH, R-SUCL, R-RCL, R-ICL, R-IE, R-SUC, R-RC, R-IC
BaseXUpdateBenchmark	Update	U-AML, U-IH, U-SUCL, U-RCL, U-ICL, U-IE, U-SUC, U-RC, U-IC
BaseXDeleteBenchmark	Delete	D-AML, D-IH, D-SUCL, D-RCL, D-ICL, D-IE, D-SUC, D-RC, D-IC
BenchmarkRunnerBaseX	-	
Neo4J		
Neo4JCreateBenchmark	Create	C-AML, C-IH, C-SUCL, C-RCL, C-ICL, C-IE, C-SUC, C-RC, C-IC
Neo4JReadBenchmark	Read	R-AML, R-IH, R-SUCL, R-RCL, R-ICL, R-IE, R-SUC, R-RC, R-IC
Neo4JUpdateBenchmark	Update	U-AML, U-IH, U-SUCL, U-RCL, U-ICL, U-IE, U-SUC, U-RC, U-IC
Neo4JDeleteBenchmark	Delete	D-AML, D-IH, D-SUCL, D-RCL, D-ICL, D-IE, D-SUC, D-RC, D-IC
BenchmarkRunnerNeo4J	-	

Table 6.2: Benchmark Structure Overview

6.2.2 Java Microbenchmark Harness

In order to measure the performance of the data storage solution, use cases are executed on the data stores using the performance evaluation infrastructure shown in Figure 6.1.

These use cases are each implemented in separate methods and therefore, measuring the performance of the data store can be achieved by measuring the execution time of one of these use case methods. The JMH framework [9] provides a solution for the execution of microbenchmarks on methods of Java components. According to Stefan

[9]Java Microbenchmark Harness: `www.openjdk.java.net/projects/code-tools/jmh/`

et al. [53], microbenchmarks are performance evaluations of small parts of a program (e.g. methods) which are often executed and therefore, they potentially influence the overall performance of a program. As stated in [53], JMH is the most commonly used Java microbenchmark framework. Nevertheless, as shown by Gil et al. and Horký et al., Java benchmarking itself is not trivial due to the Just In Time (JIT) compiler and the Garbage Collector (GC) [23][33]. The JIT compiler is aimed at improving the performance of Java programs because it compiles byte code into native machine code at run time [16]. The GC is used to execute automatic memory management by finding unused objects in the memory and deleting them which frees up memory space [17]. Therefore, the JIT compiler could change the code during the execution which could lead to unwanted performance execution results due to the time and resources the JIT compiler itself consumes. The GC on the other hand could asynchronously consume processor execution cycles which could lead to a performance reduction during the execution of the code. JMH itself aims at providing a solution to these problems [46]. The main goal of the JMH framework is to provide an efficient solution to execute benchmarks in the Java programming language. JMH itself offers the possibility to execute microbenchmarks, meaning that it is able to measure the performance of different components instead of measuring the performance of a complete system. The use cases which are used to measure the performance of different data stores are implemented inside the methods of the according *Managing Components* of the data stores meaning that JMH's microbenchmarking can be used for the performance evaluation. JMH is able to analyze the performance and calculate the execution time of methods in Java. These execution time calculations can be configured. JMH offers the following time methods:

Throughput
As the name suggests, this benchmark mode measures the number of operations per seconds e.g. how often the benchmark method can be executed per second.

Average Time
This benchmark mode measures the average execution time of the benchmark method (one single execution).

Sample Time
This benchmark mode measures the time it takes for the benchmark method to execute; additionally, the minimum and maximum execution time is measured.

Single Shot Time
This benchmark mode measures the time a single benchmark execution takes e.g. a cold start.

All
Includes all of the above mentioned benchmark modes.

For the execution of the benchmark methods, the average time method was selected as it provides the average execution time which can be used to compare different use cases. In addition to these configurations, JMH executes the benchmark as defined by the iterations property of the JMH runner class. Furthermore, warm up iterations are added which aim at removing the overhead from the warm up phase of the Java Virtual Machine (JVM). For the purpose of the benchmark execution, ten iterations and ten warm up iterations where configured. JMH is annotation based which can be seen in the code snippet of Listing 14. It shows an annotated Java class which can be executed as a JMH benchmark. By annotating the Java class with *@State(Scope.Benchmark)*, JMH knows that this class holds a benchmark and it is therefore possible to add this class to the JMH runner. In order to be able to provide initialized variables to the benchmark method–which should not be initialized during the benchmark itself–JMH provides the *@State(Scope.Thread)* annotation for inner classes where variables can be initialized. In addition to that, setup *@Setup(Level.Invocation)* and tear down *@Setup(Level.Invocation)* methods can be implemented which allow for executing code before and after each benchmark method. The benchmarks themselves are implemented as Java methods and are annotated with *@Benchmark*. Additionally, configuration annotations such as *@BenchmarkMode* and *@OutputTimeUnit* are possible.

```java
@State(Scope.Benchmark)
public class BaseXCreateBenchmark {

  @State(Scope.Thread)
  public static class MyState {
    private CAEXFile caexFileNew = new CAEXFile();
    private CommonAPIBaseX commonApi = new CommonAPIBaseX();

    @Setup(Level.Invocation)
    public void doSetup() { //init variables }

    @TearDown(Level.Invocation)
    public void doTearDown() { //close db connection etc. }
  }

  @Benchmark
  @BenchmarkMode(Mode.AverageTime)
  @OutputTimeUnit(TimeUnit.MILLISECONDS)
  public void benchmarkCreateCAEXFile(MyState state) {
    state.commonApi.createCAEXFile(state.caexFileNew);
  }
}
```

Listing 14: JMH Microbenchmark Code Example

In addition to these benchmark classes, a *BenchmarkRunner* is needed, which starts the execution and triggers the benchmark. This *BenchmarkRunner* provides the option to configure the benchmark by adding different arguments. Listing 15 shows the

BenchmarkRunner for BaseX stating all arguments used for the performance evaluation.

```java
public class BenchmarkRunnerBaseX {
  public static void main(String[] args) throws Exception {
    DateTimeFormatter formatter =
    DateTimeFormatter.ofPattern(Constants.PATTERN,
      Locale.GERMAN);

    Options opt =
      new OptionsBuilder()
        .include("BaseXUpdateBenchmarkHMI")
        .shouldDoGC(true)
        .resultFormat(ResultFormatType.CSV)
        .result(String.format(
        "%s%sBaseX-%s.csv", Constants.FOLDER,
        File.separator,
        LocalDateTime.now().format(formatter)))
        .jvmArgsAppend("-Djmh.stack.period=1")
        .warmupIterations(1)
        .measurementIterations(10)
        .forks(1)
        .build();
    new Runner(opt).run();
  }
}
```

Listing 15: JMH Microbenchmark *BenchmarkRunner* Options

The following list provides an explanation about each benchmark option used:

include()
The *include()* option offers the possibility to state which benchmarks should be included in the run of the performance evaluation. The input to this option method can be a regular expression which maps to one or several benchmark class names which are used in the execution run of the overall benchmark.

shouldDoGC()
This benchmark option can be used to configure whether Java's garbage collection should be executed in between benchmark iterations or not. Garbage collection is used for the execution of automatic memory management by finding unused objects in the memory and deleting them (see Section 6.2.2). This could potentially lead to unwanted performance execution results. Therefore, this option is useful in order to avoid problems with the Garbage Collector (GC) in Java.

resultFormat() & result()
The *resultFormat* and *result* options can be used to configure which data format and output file name should be used for the generation of the result file (e.g. Comma Separated Values (CSV) format with the name *basex.csv*).

jvmArgsAppend()
This benchmark option can be used to append JVM arguments for the execution of the benchmark. In particular, the arguments will be appended to the start command of the JMH benchmark Java program. The JVM itself is configurable (e.g. how much heap size it can allocate) via arguments and therefore changing them for the execution of a benchmark could potentially influence the performance results.

warmupIterations()
Horký et al. state that one of the most well-known issue in the area of performance evaluation is the warm up [33]. In particular, Java's JIT compiler and the class loading processes must be considered when executing Java benchmarks. Section 6.2.2 introduces to the problems of the JIT compiler in terms of performance benchmarks. In order to avoid the performance loss due to JIT compilation, warm up iteration can help to minimize the compilation effort during the actual benchmark run-time. Class loading in Java is executed class by class and on demand (i.e. first use). This results in the fact that classes under benchmark should be loaded beforehand in order to avoid unwanted performance results. Therefore, the *warmupIterations* option can be used to define how many warm-up iterations are executed before the actual benchmark execution.

measurementIterations()
The actual number of measurement iterations can be configured using the *measurementIterations* option.

forks()
The JVM is able to optimize the execution of Java programs by rearranging unrelated method calls. This could influence the benchmark execution. Therefore, the fork option can be used to reset the optimization process of the JVM.

6.2.3 Input to Benchmark

As stated in Section 2.4, every performance benchmark needs some form of input data. The performance evaluation of this work is based on different use cases which require different input types. The following list provides an overview on all used input types:

AML File
Due to the fact that this benchmark is implemented specifically for the AML model, one form of input data are AML files or sub-parts of AML files. As AML files are utilizing the XML structure as a data format, a way to transform this kind of data into a usable format for the performance evaluation is needed. Therefore, Java marshalling is used to map the content of an XML (e.g. AML) file to data store entities (see Section 6.1.1).

This input type is needed for use cases which deal with creating and updating data in the data store.

AML Element IDs
Not all use cases which are evaluated in the performance benchmark require a complete AML file or sub-parts as input, another input type is required. In order to access data from a data store, e.g. by reading an AML file, the identification of this AML file is sufficient in order to query the data from the data store. The identification of AML model concepts is the name of the concept. Therefore, this input type is a simple string input. This input type is typically needed for use cases which deal with reading or deleting data in the data store.

Depending on the input type, a JMH benchmark reads the required input data from a file or from a class variable. In order to provide a structured benchmark environment, the resource files are stored in a resource folder inside the test directory of the performance evaluation infrastructure. In addition, property files which hold the value of the file name and AML model concept identifications are used. This property file is accessed during the execution of the benchmark.

6.2.4 Performance Indicators

For the investigation executed in this thesis, time was selected as the most relevant performance indicator for the use cases. Further indicators might be relevant, however, in order to utilize them for the performance evaluation process a deeper investigation would be needed that goes beyond this thesis.

Therefore, the performance indicator for this performance evaluation is the duration of the operation. The duration of the operation indicator is mapped to the actual result of the performance execution. In this case, it is mapped to the average time a use case execution takes (calculated by dividing the sum of all execution times of a use case by the times of execution). The values for the indicator are derived from JMH's *Average Time* mode. The unit of this performance indicator is milliseconds per operation (ms/op). This performance indicator can, for example, show that more complex operations are more likely to take up more execution time than simpler ones.

6.2.5 Output of Benchmark

JMH benchmarks can have several options for the result format of the benchmark (see Section 6.2.2). In particular, the following result formats are possible:

CSV: Results in a Comma Separated Value File where values are separated via commas.

JSON: Results in a simple JavaScript Object Notation (JSON) file containing relevant result data.

TEXT: Results in a simple text file.

LATEX: Results in a basic latex source file which can be used to produce Portable Document Format (PDF) reports.

SCSV: Results in a Semicolon Separated Value file where values are separated via semicolons.

```
Benchmark;Mode;Cnt;Score;Error;Units
Neo4JCreateBenchmark.benchmarkAddInstanceHierarchy;avgt;12.414;ms/op
Neo4JCreateBenchmark.benchmarkAddInterfaceClass;avgt;4.565;ms/op
Neo4JCreateBenchmark.benchmarkAddInterfaceClassLib;avgt;8.218;ms/op
```

Listing 16: Neo4J Performance Benchmark Example Result File

The result type format of the benchmark execution process used in this work is a CSV file. This type was chosen as it allows for easy data import to spreadsheet based tools such as Microsoft Excel [10] which are later used to execute data analysis. Listing 16 shows an excerpt from a benchmark result containing the benchmark execution times of several use cases on the Neo4J data store. The benchmark mode used in this benchmark execution was *Average Time* as stated in the Mode column (*avgt* - *Average Time*) and the time unit used to list the results was *ms/op* (Milliseconds per Operation).

[10] Microsoft Excel `products.office.com/excel`

CHAPTER 7

Evaluation Process and Results

As stated in the research approach in Chapter 4, the third step of this thesis is to execute an evaluation, in particular to design and execute a controlled performance evaluation experiment. Therefore, this chapter starts with defining the design of the performance evaluation in Section 7.1. The process used in the performance evaluation will be explained in Section 7.2. Lastly, Section 7.3 describes the execution of the performance evaluation and analyzes the evaluation results.

7.1 Evaluation Experiment Design

The performance evaluation of the data storage solutions for storing and querying Automation Markup Language (AML) model data is based on a controlled experiment whose conditions have to be defined. In order to do this, an underlying process has to be utilized which provides a design process framework which can be used in the performance evaluation of this thesis. While Wohlin et al. [58] clearly differentiate between case studies and experiments, Runeson and Höst [48] propose a case study design process for software engineering research that provides valuable input for this work. The authors introduce guidelines that provide checklists which can be followed when designing software engineering related case studies. These guidelines can also be utilized as a foundation for the design of this performance evaluation experiment.

7.1.1 Evaluation Guidelines

The guidelines proposed by Runeson and Höst [48] contain a set of questions which have to be answered by the designer of the research case study. This set contains 50 different questions in the areas of *Case Study Design*, *Preparation for Data Collection*, *Analysis of Collected Data* and *Reporting*. As the process of this evaluation experiment should also contain a data collection and analysis as well as a reporting section (e.g. discussion of the results), these guidelines seem suitable for the evaluation experiment of this thesis.

The scope of these 50 questions covers a wide variety in order to support a broad field of software engineering related case studies. The performance evaluation of the two data storage solutions of this thesis is a controlled experiment and not a typical case study. However, nine questions were picked that seemed most relevant for this thesis, which will be answered in the following. Table 7.1 lists the questions selected for the design of this performance evaluation experiment. For the purpose of this thesis the categories are defined as follows: evaluation experiment design, preparation for data collection, analysis of collected data and reporting.

Evaluation Experiment Design
(1) What is the case and its units of analysis?
(2) Are clear objectives, preliminary research questions, hypotheses (if any) defined in advance?
(3) Is the theoretical basis—relation to existing literature or other cases—defined?
Preparation for data collection
(4) Does the collected data provide ability to address the research question?
(5) Are measurement instruments and procedures well defined (measurement definitions, interview questions)?
Analysis of collected data
(6) Are there clear conclusions from the analysis, including recommendations for practice/further research?
(7) Is the analysis methodology defined, including roles and review procedures?
Reporting
(8) Are the case and its units of analysis adequately presented?
(9) Are the objective, the research questions and corresponding answers reported?

Table 7.1: Evaluation Experiment Design Questions based on [48]

Nine questions, based on Runeson and Höst [48], were selected and applied to the context of this performance evaluation experiment (see Table 7.1). These questions are answered in the paragraphs below.

Answer Question 1 The object of this experiment is the evaluation of two data storage solutions–BaseX and Neo4J–in terms of their ability to store and query AML model data efficiently. The goal is to determine which of the two data storage solutions is better suited to store and query AML model data based on the proposed use cases (see Chapter 5). This is done by executing defined use cases such as the creation and deletion of *InternalElements* (see Chapter 5) and evaluating their performance results. The units of analysis for this experiment are the results of the performance evaluation. These results are average execution times of the use cases which are executed using the software architecture proposed in Chapter 6.

Answer Question 2 The objectives, like finding an efficient data storage solution for storing and querying AML model data which can be used as a central data store in the context of Multidisciplinary Engineering (MDE), were defined in advance in Section 1.2 and Section 1.3. Furthermore, preliminary research questions about a well suited software architecture for storing and querying AML models and and a performance evaluation method (i.e. evaluation experiment) were stated in advance (see Chapter 3). The findings of this evaluation experiment should provide deeper insight and will be used to answer the research questions. *RQ1* raises the question which data storage paradigms are well suited to store and query AML models efficiently (see Section 3.1). The results of this evaluation experiment are further used to for a data analysis process. The findings of this analysis can then be used to answer these proposed research questions.

Answer Question 3 The theoretical basis of this performance evaluation process is defined in Chapter 2, Related Work. In the said chapter, information and definitions are proposed which are necessary for the evaluation experiment of this thesis. Furthermore, an analysis is executed to determine relevant data storage solution candidates for the performance evaluation process (see Section 2.3.7).

Answer Question 4 The collected data of this thesis is provided by researchers of the Otto-von-Guericke University Magdeburg, Germany[1] and comes directly from a research project which examines a production system using AML [37]. Therefore, as it is relevant data from the MDE domain, the provided data can be used to answer the research questions in this chapter. This input data will be defined in Section 7.3.1.

Answer Question 5 The measurement instruments and procedures for this evaluation experiment include the prototypical software architecture design proposed in Chapter 6. This design includes a benchmarking component–utilizing the Java Microbenchmark Harness (JMH) framework–that is used for the execution of the performance evaluation. This component also provides the actual performance results which are the input of the data analysis process.

Answer Question 6 According to Runeson and Höst [48], the conclusions drawn from the analysis must be clear and provide recommendations for further research. The analysis is executed in Section 7.3.3 where the actual performance results are analyzed. Based on this information, a discussion is executed in Chapter 8. The obtained results form the analysis and discussion should provide deeper insight to form recommendations for future research (see Chapter 9).

[1]University Magdeburg: `uni-magdeburg.de`

Answer Question 7 The data analysis methodology is defined in Section 7.2. In this section, the approach is proposed stating that several graphical representations and tables are generated using a spreadsheet tool. These representations are then analyzed and used to propose deeper insight into the topic of this performance evaluation experiment.

Answer Question 8 An adequate representation of the analysis and its units (i.e. results) is needed. In order to provide such a representation, several tables and graphs are presented and explained, providing a better overview of the actual performance results (see Section 7.3.3). These representations make it easier for the reader to understand the actual results of this performance evaluation experiment.

Answer Question 9 The research questions are stated in Chapter 3 and answered in Chapter 8. Therefore, both the questions and the answers are reported.

The following sections explain the process of the performance evaluation experiment which is based on the questions and answers provided above. The definition of the benchmarking process and the execution of the performance evaluation are therefore based on the case study design guidelines proposed by Runeson and Höst in [48].

7.2 Benchmarking Process

The evaluation process used to determine the performance of two data storage paradigms for storing and querying AML models needs a defined execution path in order to be reproducible for future evaluation executions and extensions (see Section 7.1). Therefore, the performance evaluation experiment of this benchmarking process needs to be defined including each benchmarking process step needed to generate the performance results.

The benchmark process consists of several steps which are executed by different actors. There are two actors needed to perform the performance evaluation: the benchmark executor (e.g. a researcher or a software engineer) and a computational device (e.g. a laptop or a server). Figure 7.1 shows the steps needed to gather and analyze the performance evaluation results.

The first step is to prepare the benchmark (1). The execution of the benchmark relies on input data, as shown in Section 6.2.2 and Section 6.2.3. For this, the evaluation process of this thesis requires that the following two steps are executed in advance: a preparation of the input data and a connection of the input data to the benchmarks. Due to the fact that the actual implementation of the data stores is running inside a Docker container, starting the Docker containers on the computational device in advance is needed so that the benchmark can access the data stores. After the benchmark executor has finished these preparation steps, the benchmark is started on the computational device (e.g. a laptop or server) (2). Depending on the computational device, the execution of this benchmark, including evaluating all use cases against both data stores, requires up to 2.5 hours. The computational device used to execute this benchmark is described in

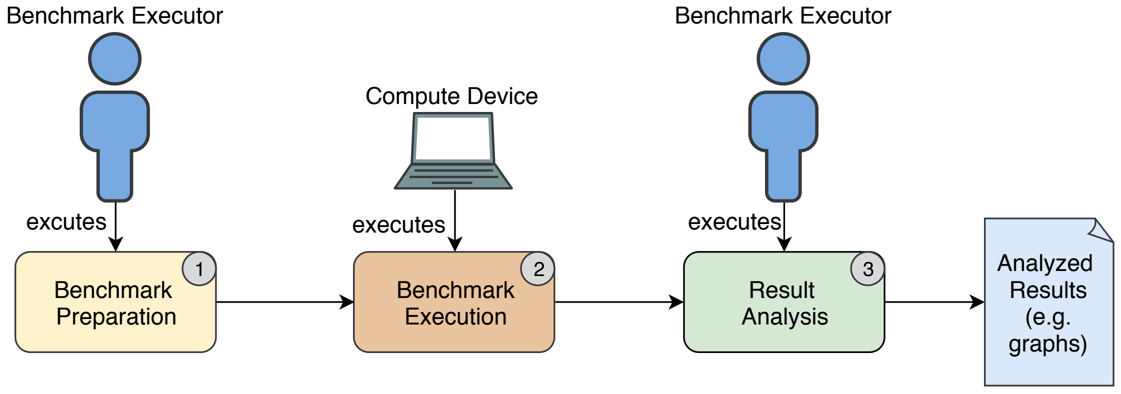


Figure 7.1: Benchmark Process of this Thesis

Appendix 9.2. After the benchmark execution is finished, the results are stored on the execution device waiting to be analyzed. Therefore, the next step is again executed by the benchmark executor and consists of an analysis of the provided result data (3). This can be done by using several data analysis tools and solutions. This thesis however, uses Microsoft Excel to generate result graphs which provide a graphical representation of the data. Furthermore, easier data transformation processes for a better graphical representation are executed using the spreadsheet tool. The result of this benchmarking process is, on the one hand, the raw result data, e.g. the average execution time of each use case, and on the other hand, the analyzed graphs. These artifacts can then be used to gain deeper insight into which data storage solution is better suited to store and query AML model data.

The benchmark execution itself is actually the evaluation execution running on the computational machine. Figure 6.2 shows this process in detail. The input data is accessed by the benchmarking class (see Section 6.2.2) and used to calculate a performance result (e.g. an average execution time) for each use case.

7.3 Execution of Performance Evaluation

The following sections define the actual execution of the performance evaluation following the defined evaluation experiment process of Section 7.1. The performance evaluation process as defined in Section 7.2 is used, containing the use of input data (see Section 7.3.1) and performance indicators (see Section 2.4). All results of the executed benchmark are provided in Section 7.3.2.

7.3.1 Evaluation Input Data

For the execution of the use cases–using the designed performance evaluation environment–an AML file provided by researchers of the Otto-von-Guericke University Magdeburg, Germany, was used. This AML file was created for the purpose of an academic demon-

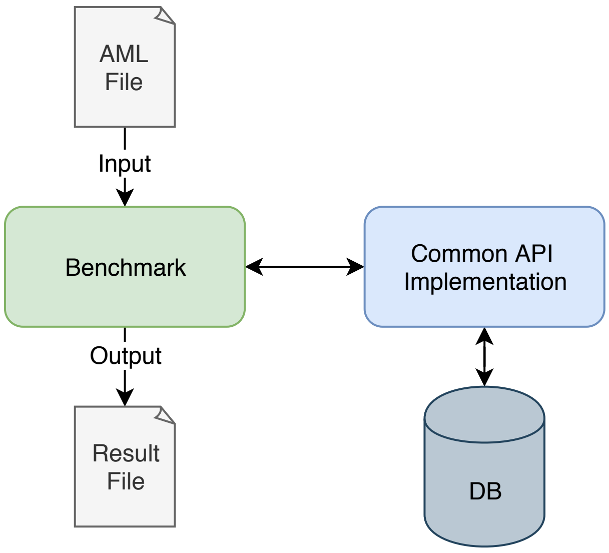


Figure 7.2: Benchmark Component Execution Process

stration and testing project with focus on a lab-sized production system [37]. According to Mazak et al., this production system was used to evaluate their Run-Time Data Collection approach–AML-RTDC–based on real world requirements. Figure 7.3 shows a graphical representation of the AML file which consists of one single Instance Hierarchy with 166 Internal Elements and seven Interface Class Libs with 27 Interface Classes, 14 Role Class Libs with 198 Role Classes and three System Unit Class Libs with 53 System Unit Classes. In addition, the file itself contains 21618 lines of code and uses about 1.4 Megabytes of data.

The benchmark itself uses several input types (see Section 6.2.3). Therefore, the complete example AML file and sub-parts of the file were used for this performance evaluation process. These sub-parts contain sub-elements of an AML file e.g. an *InstanceHierarchy*. For example, the use case 'C-ICL' (see Table 5.1) consists of creating/adding a new *InterfaceClassLib* to an *AMLFile*. It requires an *InterfaceClassLib* in the form of an Extensible Markup Language (XML) file as input. For the execution of this evaluation process, sub-element files are created by using copies of elements which are located inside the complete *AMLFile*. These copies are altered in order to fit the requirements of the use cases.

As the execution time of the performance evaluation on the used environment (see Appendix 9.2) with the input file from Mazak et al. [37] is around one hour, several smaller example files (see the example in Listing 1) were used to speed up the design and development process. Nevertheless, the results presented in this thesis are gathered using the complete example file provided by Mazak et al. in [37].

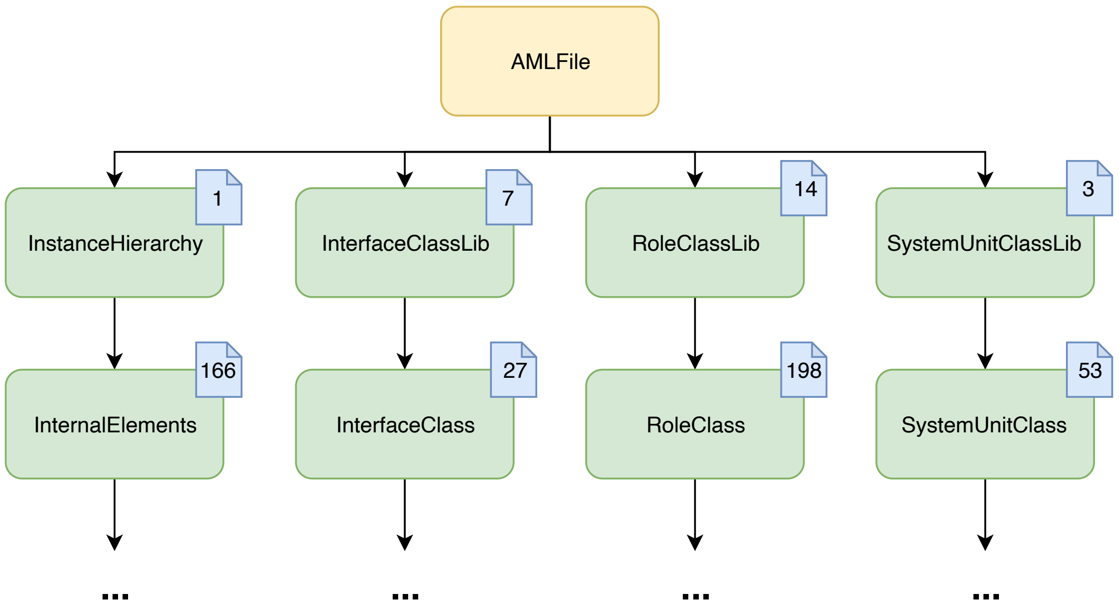


Figure 7.3: Example AML Input File

7.3.2 Performance Benchmarking Results

The execution of the performance evaluation benchmark creates two result files (see Section 6.2.5)–one for BaseX and one for Neo4J–containing the measured average execution times. These raw data files provide the basis for a more detailed analysis which will be used to draw further conclusions about the differences in performance between the two data stores. For the purpose of this work, the analysis of the data requires the generation of graphical representations like graphs and tables as well as the calculation of mathematical data transformations. The generated graphs can be used for the analysis task due to the fact that they provide a better overview as well as representation of the data. Furthermore, tables provide a structured overview of the actual raw data. In addition, mathematical data transformation can sometimes be useful for the representation of the data in graphs in order to provide a suitable design. For the analysis executed in this thesis, several different data graphs were generated including several different bar charts focusing on one or multiple use cases of the performance evaluation. In order to do this, a software tool has to be used which allows for generating graphs and tables as well as calculating mathematical transformations. Therefore, a Microsoft Excel spreadsheet was used to aggregate as well as transform the results into the required artifacts.

7.3.3 Result Analysis

In order to give a complete overview of all average execution times, the table shown in Figure 7.4 lists the results. The table consists of five columns where the first column lists each AML model concept. The other four columns are used to group the use cases into four groups: create, read, update and delete. These groups represent the main use case categories which are based on the fundamental data operations CRUD (Create, Read, Update, Delete). For each of these categories, two further sub-columns are shown, which structure the results into performance evaluation results of the two data stores, i.e. Neo4J and BaseX. These sub-columns contain the actual performance results which are generated by the JMH framework. The unit of the results is milliseconds per operation (ms/op). For each AML model concept, a corresponding average execution time is listed; mapped to the according data store as well as use case category. The results were mathematically rounded to two decimal places.

Figure 7.5 shows a graphical representation of the data. As the results vary quite widely in terms of execution times, providing a graphical representation requires some form of data transformation. For example, the creating an AML file (Use Case C-AML) using Neo4J as an underlying data store requires *73494,57* ms/op whereas BaseX requires only *102,60* ms/op. Therefore, Figure 7.5 is using a mathematical transformation in order to fit the results into one single diagram. The following mathematical transformation where x is the average execution time is used: $log10(10 * x)$. The diagram itself shows eight groups of bars which represent the use case categories for each data store. For example, the first group is called *CreateNeo4J*. It shows all transformed performance results of all use cases executed on the Neo4J data store which deal with creating data (e.g. Create an *AMLFile*). The colors of the bars represent one specific AML model concept. The actual mapping is shown in the legend of the diagram. The height of the bar itself represents the result of the use case transformed using the above mentioned mathematical data transformation. Therefore, the y-axis shows the performance of a use case. The bars allow for an easy comparison of the use cases, e.g. the creation use cases of Neo4J (first block) took longer in comparison to the creation use cases of BaseX (second block). This graphical representation can be used to provide a first overview of the performance evaluation results of the two data stores.

Nevertheless, the overview table (Table 7.4) and the graphical overview of all result data (Figure 7.5) does not provide further detailed information about the data store performance. Therefore, further analysis is needed. This analysis is conducted by selecting specific use cases and their according execution results which seem to provide interesting result data. The following subsections analyze these results of use cases executed directly on the *AMLFile* as well as other sub-elements of the *AMLFile*. The IDs of the use cases are mentioned in the heading of the subsections (see Table 5.1).

Creating (C-AML)& Updating (U-AML) *AMLFiles*

BaseX is a data store specifically intended to store XML-based data. This is done by directly storing XML files and altering them using XML Query Language (XQuery). Therefore, storing (e.g. creating or updating) operations using this type of data store are efficient for XML files. As AML is based on XML, storing *AMLFiles* in a BaseX data store is substantially faster than using Neo4J. Neo4J needs to transform the *AMLFile* into data store entities which are used during the storing process. Since the storing and

	Create		Read		Update		Delete	
	Neo4J	**BaseX**	**Neo4J**	**BaseX**	**Neo4J**	**BaseX**	**Neo4J**	**BaseX**
InstanceHierarchy	11258,73	220,15	8,61	25,61	10811,71	116,37	6,91	0,92
InterfaceClass	286,29	109,79	10,05	15,88	265,65	100,49	7,79	0,96
InterfaceClassLib	557,15	222,28	11,45	16,21	409,89	99,45	9,04	0,93
InternalElement	11960,65	153,91	3,92	16,11	10426,88	118,48	5,87	0,35
RoleClass	239,16	110,43	9,58	16,53	644,86	104,17	4,99	0,92
RoleClassLib	418,97	219,06	12,68	16,76	842,40	104,52	7,37	0,93
SystemUnitClass	1266,85	164,06	11,71	21,40	2662,13	104,41	7,09	0,99
SystemUnitClassLib	20955,14	231,56	13,09	29,09	37457,72	162,42	7,22	0,89
AMLFile	73494,57	102,60	27,76	52,95	90980,30	157,77	7,12	0,86

Figure 7.4: Performance Benchmark Results for BaseX and Neo4J in ms/op

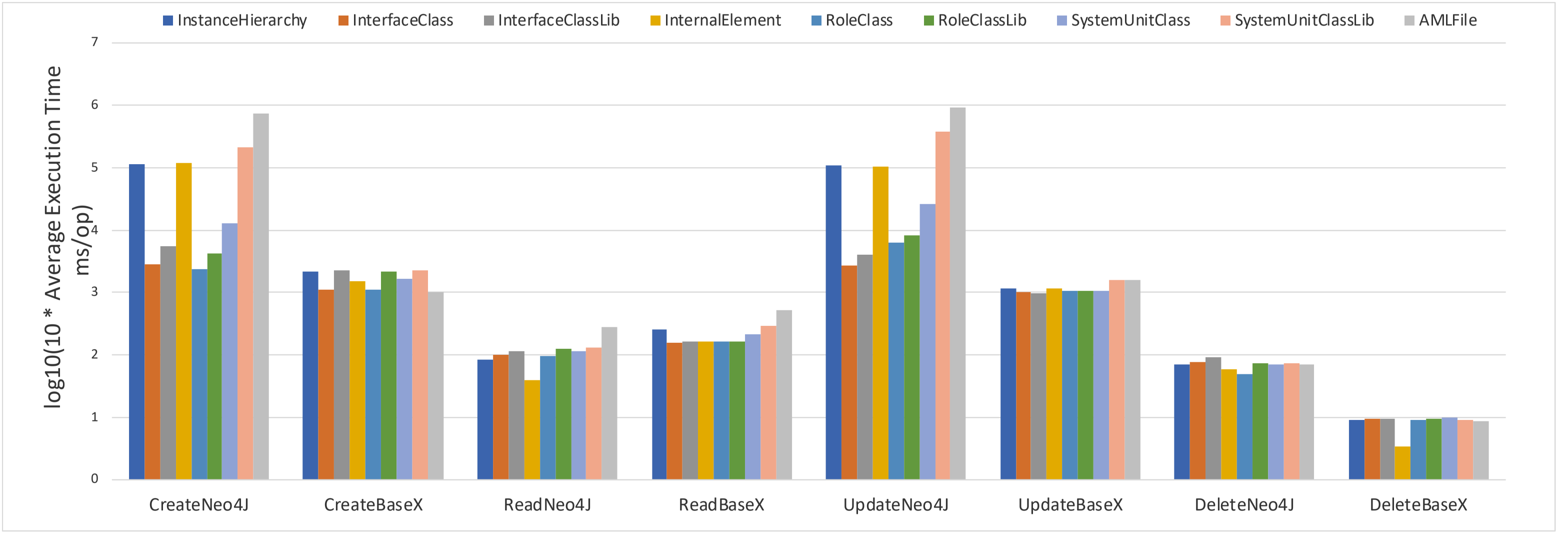


Figure 7.5: Performance Benchmark Diagram

handling of data store entities is not as optimized as directly storing an *AMLFile* (i.e. XML file), Neo4J takes longer to store or update the data.

Figure 7.6 shows a comparison between the average execution time of storing an *AMLFile* in a BaseX data store and a Neo4J data store (Use Case C-AML). Neo4J has an average execution time of *73494,57ms/op* and BaseX *102,60ms/op*. Furthermore, Figure 7.7 shows the same comparison of the updating process (Use Case U-AML). For this use case, Neo4J takes *90980,3ms/op* and BaseX *157,77ms/op*. Both figures show that for use cases which mainly include create and update operations, BaseX provides substantially better performance.

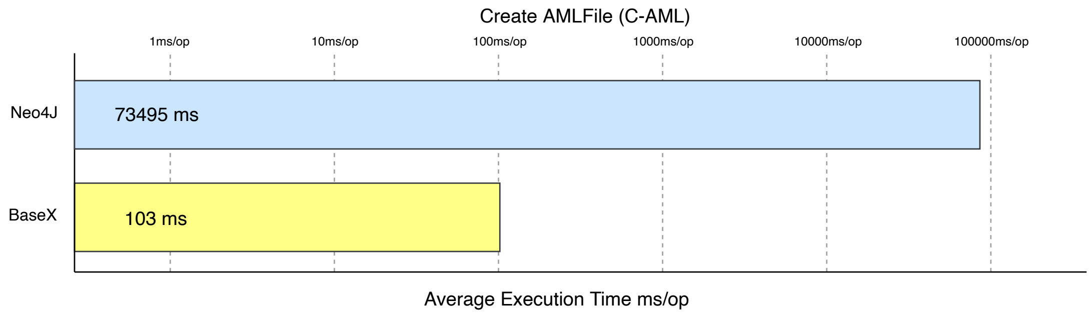


Figure 7.6: Result Use Case C-AML: Creating an *AMLFile*

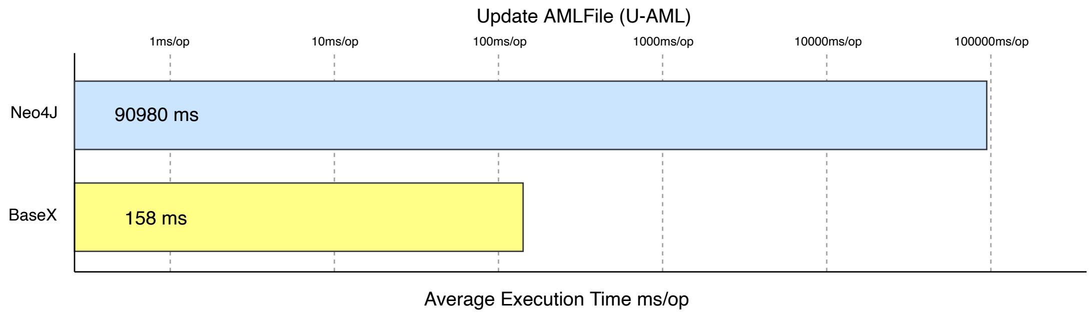


Figure 7.7: Result Use Case U-AML: Updating an *AMLFile*

Reading *AMLFiles* (R-AML)

Once entities are stored inside a data store, accessing them is a common task which is done quite frequently during the usage of a data store. Furthermore, update and create operations also include read operation used for existence checks of data entities. As Neo4J is optimized for efficient querying and accessing performance for linked data, it provides a better performance than BaseX for reading use cases.

Figure 7.8 shows the average execution time of use case R-AML which reads an *AML-File* from the data store. Neo4J takes on average *27,76ms/op*, whereas BaseX takes *52,95ms/op*. Therefore, Neo4J offers approximately twice the performance for read use cases (see Table 7.2).

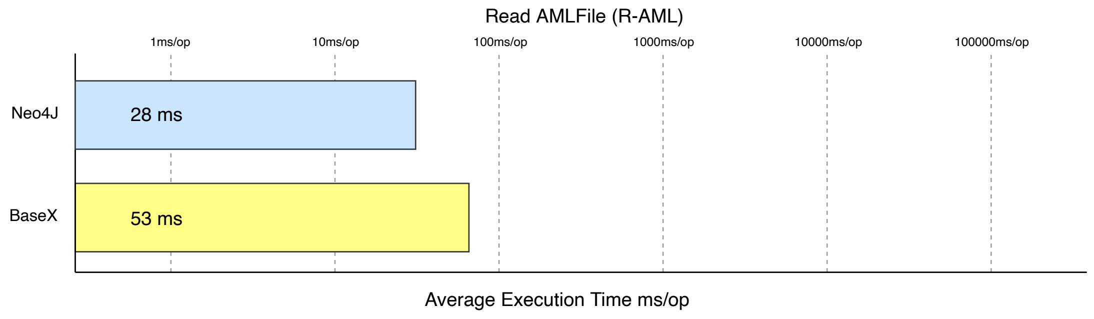


Figure 7.8: Result Use Case R-AML: Reading an *AMLFile*

Deleting *AMLFiles* (D-AML)

For the purpose of deleting *AMLFiles* from the data store, BaseX provides a better performance. This comes from the ability of simply deleting the XML file from the data store. In comparison, Neo4J has to delete all data store entities and the links between them. Figure 7.9 shows the resulting time difference of the example use case D-AML–Neo4J takes *7,12ms/op*, whereas BaseX takes less than *0,86ms/op*. Therefore, Neo4J takes approximately eight times longer than BaseX (see Table 7.2).

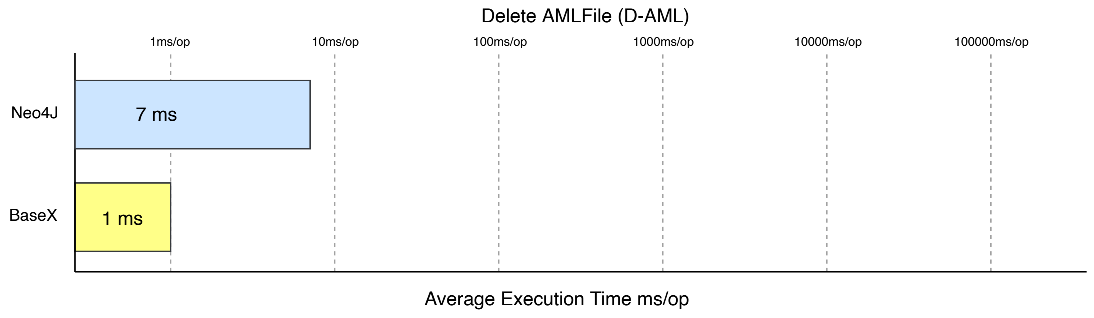


Figure 7.9: Result Use Case D-AML: Deleting an *AMLFile*

Creating (C-IE) & Updating (U-IE) *InternalElements*

In comparison to use cases which deal with the handling of *AMLFiles*, creating and updating *InternalElements* requires accessing a deeper hierarchical level in the data store. Therefore, more steps are needed to execute the use cases (see Section 5.2). As

InternalElements are sub-elements of an *AMLFile*, the input file for this benchmark is an *InternalElement* extracted (i.e. copied and edited) from the main input file (see Section 7.3.1). The actual average execution time of Use Case C-IE is shown in Figure 7.10 where Neo4J took *11960,65ms/op* and BaseX *153,91ms/op*.

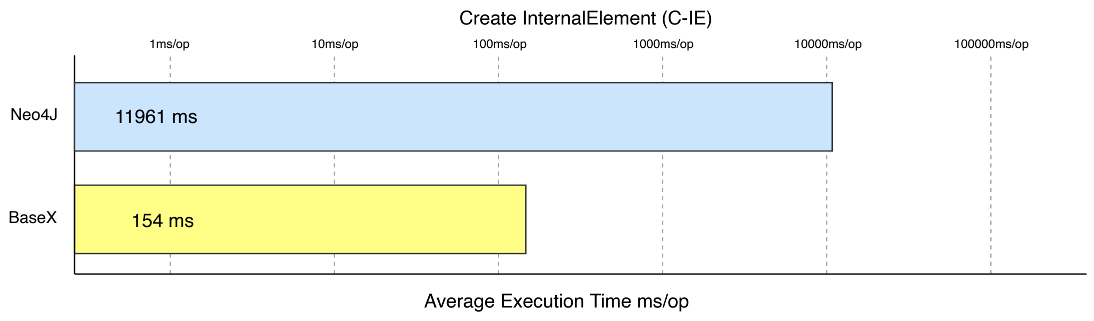


Figure 7.10: Result Use Case C-IE: Creating an *InternalElement*

Reading *SystemUnitClasses* (R-SUC)

Reading sub-elements of an *AMLFile*, e.g. *SystemUnitClasses*, resulted in average execution times, which are more or less the same as the average execution times of reading *AMLFiles*. The fact that Neo4J is optimized for accessing complex data sets which are linked in graph-like structures is fundamental in these use case executions. This leads to the fact that Neo4J takes half the time than BaseX does for accessing (i.e. reading) a *SystemUnitClass* from the data store. The actual execution times are shown in Figure 7.11 where Neo4J executed the use cases on average at *11,71ms/op* and BaseX at *21,4ms/op*.

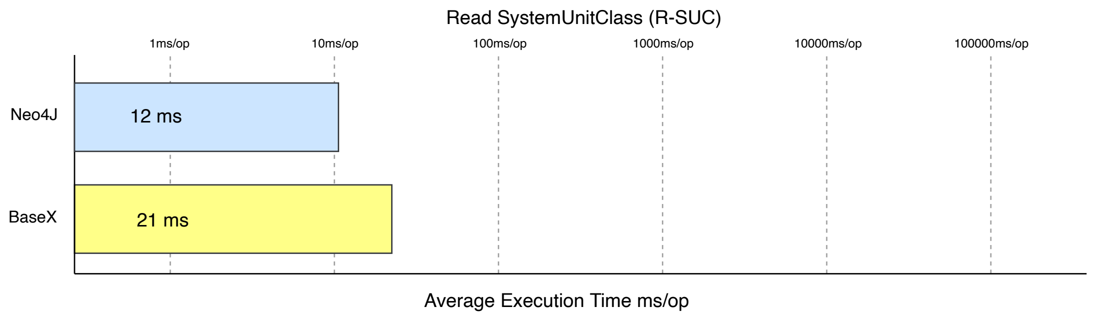


Figure 7.11: Result Use Case R-SUC: Reading *SystemUnitClasses*

Deleting *InstanceHierarchies* (D-IH)

BaseX offers performance gains for use cases which deal with deleting files as it can directly delete the underlying XML file (or structure in case of sub-elements). For the deletion process of *AMLFiles*, BaseX provides in-built functionality which allows for deleting complete XML documents. In order to delete sub-elements–such as *InstanceHierarchies*–XML Query Language (XQuery) has to be used. Nevertheless, the average execution times of the performance benchmark for deleting *InstanceHierarchies* does not widely differ from the deletion process of *AMLFiles*. This results in an average execution time of *0,92ms/op* for BaseX and *6,91ms/op* for Neo4J meaning that BaseX is approximately seven times faster (see Figure 7.12).

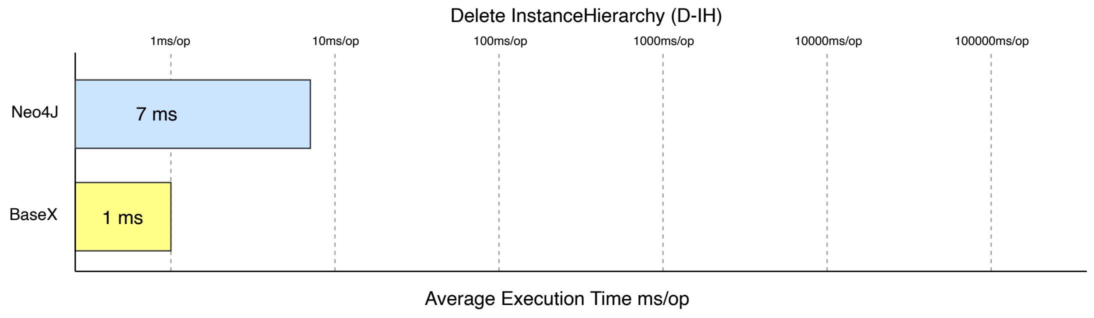


Figure 7.12: Result Use Case D-IH: Delete an *InstanceHierarchy*

Summary

Use Case	BaseX	Neo4J	Ratio
C-AML	102,60ms/op	73494,57ms/op	716,32
R-AML	52,95ms/op	27,76ms/op	1,91
U-AML	157,77ms/op	90980,30ms/op	576,66
D-AML	0,86ms/op	7,12ms/op	8,27
C-IE	153,91ms/op	11960,65ms/op	77,71
R-SUC	21,40ms/op	11,71ms/op	1,83
D-IH	0,92ms/op	6,91ms/op	7,51

Table 7.2: Result Ratio Overview: Maximum Value / Minimum Value

The actual difference between the average execution times of the performance evaluation results provides a good insight into the differences in terms of performance between the two data stores. However, providing an additional comparison, a ratio between the result values, would further improve the overall result analysis. Therefore, Table 7.2 lists the performance results of the uses cases from the result graphs shown in Section 7.3.3 and adds an additional ratio column. This ratio value is calculated using the following formula: $x = MinimumValue/MaximumValue$. The value of the ratio, i.e. x, states that the slower execution time is x-times slower in comparison to the faster execution time.

This chapter proposed the evaluation experiment on which the performance evaluation benchmark is based. The benchmark execution process as well as relevant attributes such as the input data and the performance indicators were discussed. Furthermore, the actual performance data results were shown and an analysis was executed providing several graphical representations of the performance results.

CHAPTER 8

Discussion & Limitations

This chapter discusses the results and findings of the performance evaluation as demonstrated in Chapter 7, the software architecture defined in Chapter 6 and the performance evaluation experiment process stated in Chapter 7. Therefore, Section 8.1 provides an overview of the work executed in this thesis. Section 8.2, 8.3 and 8.4 will each provide answers to the according research questions. Furthermore, the limitations of this work are discussed in Section 8.5.

8.1 Discussion Performance Evaluation

In the area of Multidisciplinary Engineering (MDE) projects, planning data needs to be exchanged between different engineers of the involved engineering domains (see Chapter 1). In order to provide a common data exchange format for this exchange process, Automation Markup Language (AML) was developed. AML is an open data exchange format for engineering planning data which can be used by different types of engineers–such as mechanical, electrical or logical engineers–involved in the project (see Section 2.2). Using this common data exchange format in MDE projects improves the project execution by eliminating the need of data transformations between different data formats. However, the exchange process itself is hardly improved, as it is still required to have some form of data exchange channel. In order to improve the data exchange process, a central data store which holds the plan data (i.e. AML) can be introduced. This data store would speed up the overall engineering process as all engineers from different domains could access the centralized data. For this, an efficient data storage solution for storing and querying AML models has to be found. Therefore, an evaluation based on the actual performance (i.e. average execution times) of the data stores needs to be executed.

This thesis compared two data storage solutions in order to find a well suited data store which can be utilized as a central data storage solution for AML model data. In order to execute this comparison, several steps were executed. The first step was a selection of data storage paradigms and solutions based on criteria provided by the AML data format which resulted in two candidates for the actual data storage paradigms evaluation process (see Section 6.1.4). In addition to that, a prototypical software architecture was designed which can be used for efficient storing and querying of AML model data (see Chapter 6). This software architecture is utilized during the execution of the performance evaluation experiment. In addition to this software architecture design, a benchmarking component was designed utilizing the Java Microbenchmark Harness (JMH) framework for Java code benchmarking (see Section 6.2.2). This component was used in the performance evaluation in order to determine the actual results. Furthermore, a performance evaluation process was established which can be used to execute a performance evaluation experiment as well as a performance result analysis. These performance results were then used to gain deeper insight into the suitability of the two data storage solutions for storing and querying AML model data efficiently (see Section 7.3.3). The results were presented using various graphical representations, created with Microsoft Excel spreadsheet software. These representations provided the foundation for the analysis process of this work. The analysis resulted in a recommendation for which data storage solution is better suited to store and query AML model data.

The results of the performance benchmark show that the performance for storing and querying AML model data varies widely between the two tested data storage solutions based on the underlying use case (see Chapter 7). The graphical representation of the selected benchmarks provide further evidence that the results, i.e. the average execution times, can be quite different. This leads to the fact that non of the two data storage solution can be selected as better suited to store and query AML model data for all use cases (see Chapter 5) evaluated in this work. Therefore, in order to provide a recommendation for choosing a data storage solution to store and query AML model data, the intended use case has to be considered. Depending on whether more accessing (e.g. reading an *InternalElement* (R-IE)) or writing (e.g. creating an *AMLFile* (C-AML)) data operations are involved, one data storage solutions is better suited than the other. This discussion can provide deeper insight into the topic of storing and querying AML model data which in turn can be used to choose a data storage solution for an according MDE project which is utilizing AML as a data exchange format. A recommendation for all MDE projects using AML model data can not be provided because different projects have different requirements.

The following two subsections provide a discussion about the results of the performance evaluation and propose which of the two evaluated data storage solutions are better suited for which specific use cases. Therefore, this discussion is based on the defined use cases (see Chapter 5).

BaseX

The first data store selected for the performance evaluation was BaseX (see Section 2.3.1). BaseX is an Extensible Markup Language (XML) data store based on the semi-structured data storage paradigm (see Section 2.3.1). This data storage solution was selected for the performance evaluation due to the criteria to store XML data because AML is based on XML. Generally speaking, BaseX offers a good performance for storing and querying XML data. However, there are some drawbacks in terms of accessing (i.e. reading) performance. BaseX is not optimized to query complex data and therefore, other data storage solutions which are aimed at querying complex (i.e. interlinked) data offer better performance. Nevertheless, creating, updating or deleting data using BaseX provides substantially better performance than the other evaluated data store (Neo4J). As a lot of use cases depend on accessing data beforehand (e.g. checking if something exists or querying something from the data store in order to alter it) using BaseX can be a drawback. This reading performance drawback can be seen in the benchmark result shown in Figure 7.8 which shows the performance difference between BaseX and Neo4J when accessing an *AMLFile* from the data store. The results show that BaseX needs approximately twice the time to access the data. However, if the use cases of the MDE project do not include accessing complex data and rather focuses on data manipulating operations, BaseX can be an efficient choice as a data storage solution for AML model data. Data manipulations, i.e. create, update or delete data, were executed in several use cases during the performance benchmark. For example, Figure 7.3.3 shows that BaseX is substantially faster than Neo4J in the creation process of an *InternalElement*.

Neo4J

The second data store selected for the performance evaluation was Neo4J (see Section 2.3.4). Neo4J is a data storage solution which follows the graph data paradigm (see Section 2.3.4) which utilizes a graph as the underlying data structure. As Neo4J is graph-based, it focuses on the connections between the stored data. This allows for providing good accessing performance for complex interlinked data (such as AML). AML is not only XML based but is also interlinked between the modelling concepts which makes it a graph-like data structure. Therefore, accessing (i.e. reading) AML model data stored in a Neo4J data store is more efficient than data stored in BaseX (see Figure 7.8). Use cases that require lots of data access operations are therefore more efficient if the underlying data storage solution is Neo4J in contrast to BaseX (such as Use Cases R-AML, R-SUC). Use cases that execute create, update or delete operations (e.g. Use Cases C-AML, U-AML or D-AML) are therefore not as efficient as BaseX and offer worse performance.

8.2 Discussion RQ1

Which data storage paradigms are well suited to store and query models of Multidisciplinary Engineering (MDE), such as Automation Markup Language (AML)?

By analyzing common data storage paradigms in the area of software engineering, a list which provides the foundation for the selection of a well suited data storage paradigm for MDE model data was created. Based on the MDE model AML and its criteria, such as interconnected data and XML as the data structure format, data storage paradigms had been selected. The selected data storage paradigms and its according implementation are well suited to store either XML data (e.g. BaseX) or cyclic graph-like interlinked structures (e.g. Neo4J). Therefore, the selected data storage paradigms were the graph data paradigm (see Section 2.3.4) and the semi-structured data paradigm (see Section 2.3.1). The according data storage implementations for these two paradigms were selected based on widespread usage in the domain of software engineering. BaseX was the selected implementation of the semi-structured data paradigm and Neo4J as a representative of the graph data paradigm. These two candidates were analyzed and evaluated using a performance benchmark based on the benchmarking process defined in Section 7.2. The results of this performance evaluation benchmark (see Table 7.4) were analyzed which results in deeper insight of the data store and query performance for both evaluation candidates.

As this question deals with the suitability of data storage paradigms to store and query models of AML, the answer is that it depends on the actual use case. Both BaseX and Neo4J are well suited to store and query AML data but they offer drawbacks in certain data store operations (see Section 8.1). Overall, BaseX offers better performance for use cases involving creating, updating and deleting data. Neo4J however is better suited for use cases which focus on accessing (i.e. reading) lots of data.

8.3 Discussion RQ2

Which software architecture is well suited for storing and querying MDE models, such as AML models, independently from the implementation of a data storage paradigm?

In order to answer this research question, a prototypical software architecture was designed and created. This software architecture is able to store MDE model data efficiently and independently from the underlying storage technology (see Chapter 6). A traditional layered software architecture approach was used in order to provide a software architecture which is independent from the implementation of the actual data storage paradigm. This layered approach includes several abstraction layers (through interfaces) including (from bottom to top) a data storage layer, a *Managing Component* for the data store, a *Common API* (which is accessed by the benchmark component) and AML model interfaces and according implementation. The data storage layer contains the actual implementation of the data storage paradigm, running inside a virtualization layer using Docker (see Section 6.1.4). For each data store in the prototypical software architecture, a managing component is needed which implements all data store specific functionalities (see Section 6.1.3). The *Common API* layer holds the declarations of the data store operations (in this case the use cases) which are implemented in the *Managing Component* of each data store (see Section 6.1.2). The AML model interface and implementation hold interface descriptions as well as implementations of the AML model concepts (see Section 6.1.1) which are used throughout the whole software architecture.

This question deals with finding a software architecture for storing and querying AML models which is independent from the implementation of the data storage paradigm. Therefore, the proposed software architecture utilizes a virtualization layer for the data storage solutions and requires a managing component for each data store. In order to exchange or add additional data store, the actual data store instance has to be added to the virtualization layer and an according data store *Managing Component* has to be designed. This allows for an independent software architecture for storing and querying AML models.

8.4 Discussion RQ3

How can a standard database performance evaluation method be adapted to measure the performance of storing and querying MDE models, such as AML models?

This question can be answered by providing a process description of a database performance evaluation method for storing and querying AML models. In this work, the performance evaluation process was defined by following the guidelines defined by Runeson and Höst [48]. Runeson and Höst propose guidelines for the definition and execution of case studies in the field of software engineering. As the performance evaluation process of this thesis can be seen as a case study, these proposed guidelines seem suitable. The guideline itself consists of a checklist with 50 different questions. Runeson and Höst state that by answering these questions a structured case study process can be defined. For this thesis, selected questions of the guideline's checklist were answered. The questions and answers provided the basis for defining a performance evaluation process (see Section 7.1 and 7.2). The process of the performance evaluation itself consists of a benchmark preparation step (input data analysis and preparation), a benchmark execution (i.e. running the performance evaluation) and a performance result analysis. The process of this performance evaluation is shown in Figure 7.1. For the benchmark execution, the software architecture–proposed in Chapter 6–and the JMH framework were used. The last step of the performance evaluation–the result analysis–is the part in which the performance results are discussed. This discussion leads to a better understanding of the suitability of the two selected data storage solutions for storing and querying AML model data (see Section 7.3.3). This insight can then be used to select an efficient data storage solution for future MDE projects which utilize AML as a data exchange format.

8.5 Limitations

The limitations of this work can be structured into two types: artifact and evaluation limitations. The following subsections provide information about the shortcomings of this work. These limitations cannot be dealt with in this work and are therefore subject to future work.

8.5.1 Artifact Limitations

Artifact limitations are limitations of the designed artifacts of this thesis. Therefore, the artifact limitations are the limitations of the design prototype which was used for the performance benchmark (see Chapter 6).

Benchmark Programming Language The programming language used for the performance evaluation architecture was Java. Java itself is not particularly suitable for the execution of a performance evaluation due to the lack of providing good benchmarking support (see Section 6.2.2). Due to the fact that Java runs inside a virtual machine–the Java Virtual Machine (JVM)–it is not a hardware focused programming language which limits the execution of the performance evaluation (e.g. the artifact). Programming languages which are more hardware focused, e.g. C[1], could provide more suitable performance results of data storage paradigms for storing and querying MDE model data due to the missing overhead of running inside a virtual machine.

Algorithm Performance The goal of this work was to design a prototypical artifact without the goal of looking at algorithm performance. In particular, the queries used for the execution of the use cases (e.g. Cypher or XML Query Language (XQuery)) could have been optimized. Therefore, one limitation to this artifact is that it could be better if less complex software algorithms or queries were implemented.

8.5.2 Evaluation Limitations

Limitations regarding the evaluation process of this work are listed below. This type of limitations deals with the shortcomings of the evaluation process executed in this thesis (see Chapter 7).

Limited Input Data The input data used for this thesis was provided by researchers of the Otto-von-Guericke University Magdeburg, Germany[2], and consists of AML data from one of their research projects [37]. As this input data represents a research project, limitations occur due to the limited informative value and scope in terms of real world usage. More complex and real-world-focused input data would have provided a more precise result of the performance.

[1]GCC Compiler `gcc.gnu.org`
[2]University Magdeburg: `uni-magdeburg.de`

Real-World Use Cases As the focus of the proposed use cases (see Chapter 5) in the performance evaluation of this thesis are common data storage operations, providing real-world use cases such as checking the consistency and filtering the data would improve the evaluation process. Therefore, using only data storage operation focused use cases is another limitation of this thesis.

Limited Performance Indicators The performance indicator used in the performance evaluation of this thesis is the average execution time of each use case. Utilizing additional performance indicators (see Table 2.3) in the evaluation process could improve the results and findings of the evaluation process. Therefore, relying solely on the average execution time as a performance indicator is an additional limitation of this performance evaluation.

CHAPTER 9

Conclusion and Future Work

This chapter concludes the work conducted in this thesis in Section 9.1 by considering the problems of finding an efficient solution for the process of sharing data among different engineers in Multidisciplinary Engineering (MDE) processes (see Section 1.2). Furthermore, issues for future work are raised in Section 9.2 considering the gained insight during the writing and execution of this thesis.

9.1 Conclusion

In the area of MDE, engineers from various domains such as mechanical, electrical or logical engineering work together in one common project (e.g. a car manufacturing plant). These projects typically involve the exchange of data. In order to avoid the overhead of having different data models in each engineering domain, using a standardized data model offers many benefits such as utilizing common concepts [42] and terminology as well as a reduction in engineering tool integration effort. Furthermore, in the domain of MDE, a standardized data model can bridge the gap between the various engineering disciplines by being exchangeable between engineers involved in the production process [7]. Nevertheless, due to the lack of integrated tool chains, even standardized MDE artifacts have their limitations such as the need for exchanging the artifacts via common communication channels, e.g. email. By providing a central data storage solution that acts as a mediator between the engineering disciplines and their entities, the overhead of exchanging data manually is eliminated. In order to find a sufficient data storage solution, several steps were executed. Data storage paradigms were analyzed (Section 2.3) and two data storage solutions were selected as candidates for further investigation (Section 6.1.4). In order to efficiently store and query MDE model data (e.g. Automation Markup Language (AML)) a prototypical software architecture was created based on a layered architecture pattern approach (Chapter 6). Furthermore, a performance evaluation process was established which proposes a suitable approach to evaluate the performance of a data storage solution in terms of storing and querying performance of MDE model data (Section 7.2). The produced results of this performance evaluation experiment can be used to evaluate which data storage solution is better suited to store and query MDE models (Section 7.3.3).

For this work, the used model data is AML as it is a data exchange format developed by different companies and researchers in order to improve the data exchange process for MDE projects (see Section 2.2). The data storage solutions evaluated in this work are Neo4J and BaseX. They were selected based on criteria coming from the model data format AML (see Section 2.3.7). In conclusion, this work shows that BaseX performs better for use cases which involve create, update and delete operations and Neo4J performs better for use cases involving read operations.

9.1.1 Improvements for Stakeholders

As the findings of this thesis are aimed at providing a benefit to MDE projects and the engineers involved in these projects, the following list provides an overview of the insights gained for the different stakeholders.

Engineers Using a central data storage solution can reduce the effort of exchanging MDE plan data manually via common data exchange channels such as email. This results into less errors during data exchange and provides less overhead in terms of data synchronization. Therefore, the findings of this thesis can be used to introduce a data storage solution for MDE, e.g. AML, model data in MDE projects based on its requirements.

Tool Developers The findings of this thesis provide insight whether BaseX (i.e. the semi-structured data paradigm) or Neo4J (i.e. the graph data paradigm) is better suited to store and query AML. Based on the performance evaluation executed using the proposed use cases, a tool developer could use these findings to determine which data storage solution can be suitable for their next software tool project in the area of data exchange solution for AML model data.

Project Managers The findings of this thesis can be used for project managers to more effectively analyze and evaluate the performance of data storage solutions which could be used in MDE project as a central data store for, e.g., AML model data. The project managers gain deeper insight into the performance of the evaluated data storage solution, based on the performance evaluation and the proposed use cases, and can use the performance results to determine an efficient solution for the needs of their project.

Researchers & Domain Experts As this thesis includes the design of a prototypical performance evaluation infrastructure, which allows for efficient storing and querying of AML model data, as well as the design of a performance evaluation process, researchers and domain experts can utilizes these artifacts in order to reason about additional further data storage solutions. These artifacts can provide a basis for other evaluation processes in the domain of MDE model data. Therefore, the artifacts of this thesis can be used to execute further research.

9.2 Future Work

During the writing of this thesis, the following ideas were gathered which could potentially be interesting for future work.

Additional Data Storage Solution This work focused on the performance evaluation of two data storage solutions which were chosen based on their relevancy to the format of the AML data model. Adding further data storage solutions such as relational data bases (e.g. PostgresSQL), hybrid databases (e.g. ArrangoDB) or document stores (e.g. MongoDB) to the performance evaluation infrastructure would provide more results and lead to a more complete basis for arguing which data storage solution is best suited for storing AML model data.

Extend Model Data Concepts As the performance evaluation is focused on basic AML model data concepts, extending to more advanced model data concepts would lead to a more refined result. This improved result could be used to reason in a more detailed manner about the performance of a particular data storage paradigm.

Additional Input Data By using more complex or bigger example input data, the result could change as the performance results are based on one example AML project (provided by Otto-von-Guericke Univeristy Magdeburg, Germany).

Additional MDE Model Data Formats As with the data storage solutions, this work was solely focused on evaluating the performance of storing AML model data. Using another MDE model data format such as the *Formalised Process Description* [1] or *Product-Process-Resource* [49] models would broaden the test results.

Additional Use Cases By using additional use cases in the performance evaluation execution, a more refined result could be established. As the use cases in this work are focused on common data storage operations such as create, read, update or delete, adding more complex use cases such as checking the consistency of the stored data would provide a better basis for analyzing whether one data storage solution is better suited for managing AML model data.

Storage Optimization The performance of the selected data storage solution could be improved by optimizing the process of storing elements in the data stores. Currently, elements can be stored in more than one place in the data store. For example, for two *InstanceHierachies* with the same sub-element, the sub-element would be stored twice in the data store.

Execution on Server The performance evaluation of this work was executed on a local machine (see Appendix 9.2). By bringing the execution of the performance evaluation as well as the data store instances to a server environment which provides more computing power, the results of the performance evaluation could improve.

Distributed Solution As the performance evaluation infrastructure is focused on one single data store with one single client (e.g. the benchmarking component), designing a distributed solution could alter the results. Therefore, bringing the benchmark and data stores into a distributed environment would be worth looking into.

List of Figures

Acronyms

AML Automation Markup Language. xi, xiii, 1–5, 7–13, 19, 20, 23, 24, 29–31, 34, 37, 40–43, 46–50, 52, 58, 59, 61–68, 75–80, 83–85, 87

AQL ArangoDB Query Language. 18

CAEX Computer Aided Engineering Exchange. 8, 9

COLLADA Collaborative Design Activity. 9

CPPS Cyber-Physical Production System. 5, 6

CSV Comma Separated Values. 58

DTO Data Transfer Object. 43

fUML Foundational UML. 33

GC Garbage Collector. 55, 57

JAXB Java Architecture for XML Binding. 42, 43

JIT Just In Time. 55, 58

JMH Java Microbenchmark Harness. 39, 54–56, 58, 59, 63, 68, 76, 79

JPA Java Persistence API. 42, 43, 48

JSON JavaScript Object Notation. 13, 16, 59

JVM Java Virtual Machine. 56, 58, 80

MDE Multidisciplinary Engineering. xi, xiii, 1–7, 12–14, 19–21, 23, 24, 29, 31, 41, 63, 75–80, 83–85

NoSQL Non Structured Query Language. 14, 16, 18, 43

PDF Portable Document Format. 60

PSE Production Systems Engineering. 6, 7, 32, 41, 87

RDBMS Relational Database Management System. 16

SPARQL SPARQL Protocol And RDF Query Language. 18

SQL Structured Query Language. 14–16, 18

UML Unified Modeling Language. 33

W3C World Wide Web Consortium. 14, 18

XML Extensible Markup Language. xi, xiii, 1, 2, 7–9, 13, 14, 16, 20, 32, 42, 43, 46, 47, 58, 66, 68, 70, 71, 73, 76–78

XQuery XML Query Language. 68

XQuery XML Query Language. 14, 45–47, 73, 80

XSD XML Schema Definition. 8

Bibliography

[1] VDI/VDE 3682: Formalised Process Descriptions. Beuth Verlag, 2005.

[2] R. Angles. A Comparison of Current Graph Database Models. In *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 171–177, 2012.

[3] Renzo Angles and Claudio Gutierrez. Survey of Graph Database Models. *ACM Comput. Surv.*, 40(1):1:1–1:39, February 2008.

[4] Timothy G. Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. LinkBench: A Database Benchmark Based on the Facebook Social Graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1185–1196. ACM, 2013.

[5] Remi Arnaud and Mark C Barnes. *COLLADA: sailing the gulf of 3D digital content creation*. AK Peters/CRC Press, 2006.

[6] Kyle Banker. *MongoDB in action*. Manning Publications Co., 2011.

[7] Luca Berardinelli, Alexandra Mazak, Oliver Alt, Manuel Wimmer, and Gerti Kappel. Model-driven systems engineering: Principles and application in the CPPS domain. In *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*, pages 261–299. Springer, 2017.

[8] Stefan Biffl and Marta Sabou. *Semantic Web Technologies for Intelligent Engineering Applications*. 2016.

[9] Stefan Biffl and Alexander Schatten. A platform for service-oriented integration of software engineering environments. *Frontiers in Artificial Intelligence and Applications*, 199(1):75–92, 2009.

[10] Stefan Biffl, Arndt Lüder, and Detlef Gerhard. *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*. 2017.

[11] Rahul Biswas and Ed Ort. The java persistence api-a simpler programming model for entity persistence. *Sun, May*, 2006.

[12] Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. Linked Data on the Web (LDOW2008). In *Proceedings of the 17th International Conference on World Wide Web*, pages 1265–1266. ACM, 2008.

[13] Carl Boettiger. An introduction to Docker for reproducible research. *SIGOPS Oper. Syst. Rev.*, 49:71–79, 2015.

[14] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (XML). *World Wide Web Journal*, 2(4):27–66, 1997.

[15] Peter Buneman. Semistructured Data. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '97, pages 117–121. ACM, 1997.

[16] Timothy Cramer, Richard Friedman, Terrence Miller, David Seberger, Robert Wilson, and Mario Wolczko. Compiling Java just in time. *IEEE Micro*, 17(3):36–43, 1997.

[17] Tamar Domani, Elliot K. Kolodner, Ethan Lewis, Eliot E. Salant, Katherine Barabash, Itai Lahan, Yossi Levanoni, Erez Petrank, and Igor Yanorer. Implementing an On-the-fly Garbage Collector for Java. *SIGPLAN Not.*, 36(1):155–166, 2000.

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

[19] Rainer Drath, Lüder Arndt, Jörn 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, 2008.

[20] Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. Addison-Wesley Publishing Company, 6th edition, 2010.

[21] Sebastian Faltinski, Oliver Niggemann, Natalia Moriz, and André Mankowski. AutomationML: From data exchange to system planning and simulation. In *Proc. IEEE Int. Conf. Ind. Technol.*, pages 378–383, 2012.

[22] Joseph Fialli and Sekhar Vajjhala. The Java architecture for XML binding (JAXB). *JSR Specification, January*, 2003.

[23] Joseph Yossi Gil, Keren Lenz, and Yuval Shimron. A microbenchmark case study and lessons learned. 2012.

[24] Saumya Goyal, Pragati Prakash Srivastava, and Anil Kumar. An overview of hybrid databases. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 285–288. IEEE, 2015.

[25] Olaf Graeser, Lorenz Hundt, Michael John, Gerald Lobermeier, Arndt Lüder, Stefan Mülhens, Nikolaus Ondracek, Mario Thron, and Josef Schmelter. Whitepaper: AutomationML and eCl@ss Integration. 2017.

[26] Christian Grün. Pushing XML Main Memory Databases to their Limits. In *Grundlagen von Datenbanken*, pages 60–64, 2006.

[27] Christian Grün. *Storing and querying large XML instances*. PhD thesis, 2010.

[28] Christian Grün, Alexander Holupirek, and Marc H. Scholl. Visually Exploring and Querying XML with BaseX. In Alfons Kemper, editor, *Datenbanksysteme in Business, Technologie und Web (BTW) : 7. - 9.3.2007 in Aachen*, number 12 in Gesellschaft für Informatik...Fachtagung des GI-Fachbereichs Datenbanken und Informationssysteme (DBIS), pages 629–632. GI, 2007.

[29] Jing Han, E Haihong, Guan Le, and Jian Du. Survey on NoSQL database. In *Pervasive computing and applications (ICPCA)*, pages 363–366. IEEE, 2011.

[30] Olaf Hartig and Jorge Pérez. An initial analysis of Facebook's GraphQL language. In *AMW 2017 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7-9, 2017.*, volume 1912. Juan Reutter, Divesh Srivastava, 2017.

[31] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Q.*, 28(1):75–105, 2004.

[32] Florian Holzschuher and René Peinl. Performance of Graph Query Languages: Comparison of Cypher, Gremlin and Native Access in Neo4J. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pages 195–204. ACM, 2013.

[33] Vojtěch Horký, Peter Libič, Antonin Steinhauser, and Petr Tůma. DOs and DON'Ts of Conducting Performance Measurements in Java. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE '15, pages 337–340. ACM, 2015.

[34] Yishan Li and Sathiamoorthy Manoharan. A performance comparison of SQL and NoSQL databases. *IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing - Proceedings*, pages 15–19, 2013.

[35] S. Makris and K. Alexopoulos. AutomationML server - A prototype data management system for multi disciplinary production engineering. *Procedia CIRP*, 2012.

[36] Tanja Mayerhofer, Philip Langer, and Gerti Kappel. A Runtime Model for fUML. In *Proceedings of the 7th Workshop on Models@Run.Time*, MRT '12, pages 53–58. ACM, 2012.

[37] Alexandra Mazak, Arndt Lüder, Sabine Wolny, Manuel Wimmer, Dietmar Winkler, Konstantin Kirchheim, Ronald Rosendahl, Hessamedin Bayanifar, and Stefan Biffl. Model-based generation of run-time data collection systems exploiting AutomationML. *at-Automatisierungstechnik*, 66(10):819–833, 2018.

[38] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. Application Programming Interface Documentation: What Do Software Developers Want? *Journal of Technical Writing and Communication*, 48:295–330, 07 2018.

[39] Justin J Miller. Graph database applications and concepts with Neo4J. In *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, volume 2324, page 36, 2013.

[40] Schleipen Miriam and Rainer Drath. Three-view-concept for modeling process or manufacturing plants with AutomationML. In *2009 IEEE Conference on Emerging Technologies Factory Automation*, pages 1–4, Sept 2009.

[41] László Monostori. Cyber-physical production systems: Roots, expectations and R&D challenges. *Procedia CIRP*, 17(October):9–13, 2014.

[42] Thomas Moser and Stefan Biffl. Semantic Integration of Software and Systems Engineering Environments. *Trans. Sys. Man Cyber Part C*, 42(1):38–50, January 2012.

[43] Thomas Moser, Stefan Biffl, Wikan Danar Sunindyo, and Dietmar Winkler. Integrating Production Automation Expert Knowledge Across Engineering Stakeholder Domains, 2010.

[44] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and Complexity of SPARQL. In *International semantic web conference*, pages 30–43. Springer, 2006.

[45] Mark Richards. *Software architecture patterns.* O'Reilly Media, Inc., 2015.

[46] Marcelino Rodriguez-Cancio, Benoit Combemale, and Benoit Baudry. Automatic microbenchmark generation to prevent dead code elimination and constant folding. *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 132–143, 2016.

[47] Ronald Rosendahl, Konstantin Kirchheim, and Josef Prinz. Implementing reference APIs for AutomationML - A Java based walkthrough. In *Conf. Proc. of the 5th AutomationML user Conf. in Gothenburg/Sweden*, 2018.

[48] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131, Dec 2008.

[49] Miriam Schleipen, Arndt Lüder, Olaf Sauer, Holger Flatt, and Jürgen Jasperneite. Requirements and concept for Plug-and-Work. *at-Automatisierungstechnik*, 63(10): 801–820, 2015.

[50] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J Carey, Ioana Manolescu, and Ralph Busse. XMark: A benchmark for XML data management. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 974–985. Elsevier, 2002.

[51] Estefania Serral, Richard Mordinyi, Olga Kovalenko, Dietmar Winkler, and Stefan Biffl. Evaluation of semantic data storages for integrating heterogeneous disciplines in automation systems engineering. In *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, pages 6858–6865, Nov 2013.

[52] Monique Snoeck, Stephan Poelmans, and Guido Dedene. A Layered Software Specification Architecture. volume 1920, pages 454–469, 10 2000.

[53] Petr Stefan, Vojtech Horky, Lubomir Bulej, and Petr Tuma. Unit Testing Performance in Java Projects. pages 401–412, 2017.

[54] Igor Tatarinov, Stratis D Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita, and Chun Zhang. Storing and querying ordered XML using a relational database system. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 204–215. ACM, 2002.

[55] James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[56] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, pages 42:1–42:6. ACM, 2010.

[57] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.

[58] Claes Wohlin, Martin Höst, and Kennet Henningsson. Empirical research methods in software engineering. In *Empirical methods and studies in software engineering*. Springer, 2003.

Used Technologies

This chapter provides an overview of all technologies and their according versions used during the creation of this thesis allowing for easier reproduction of the presented results in the future.

Tool	Version	Info
AutomationML	2.15	Data Exchange Format
BaseX	9.2.3	XML Database
draw.io	12.1.0	Diagram Software
Git	2.21.0	Version Control System
Hibernate	5.3.1.Final	Object Relational Mapping Framework
Java	1.8.0 191	Programming Language
Java Microbenchmark Harness	1.21	Benchmark Framework
JUnit	4.12	Unit Testing Framework
IntelliJ IDEA (Ultimate Edition)	2019.1.4	Integrated Development Environment
Maven	3.6.0	Build Management Tool
Microsoft Excel	16.30	Spreadsheet Software
Neo4J	3.5.7	Graph Database

Table 1: Used Software & Tools

Test Environment

This Chapter introduces to the environment used to execute and create the artifacts of this thesis. All implementations and performance evaluation were executed on the following compute machine:

- MacBook Pro 15 2018
- OS: MacOS Mojave 10.14.6
- CPU: Intel Core i7 8th Gen 2.6 GHz
- GPU: Radeon Pro 560X 4GB
- RAM: 16 GB