Enabling Data Analytics and Machine Learning in Model-Driven Software Engineering of Smart IoT Services

Armin Moin

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigten Dissertation.

Vorsitz: Prof. Dr. Jörg Ott

Prüfer*innen der Dissertation:

1. Prof. Dr. Stephan Günnemann
2. Prof. Atta Badii

Die Dissertation wurde am 07.06.2022 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 09.07.2022 angenommen.
Abstract

Engineering of modern, Artificial Intelligence (AI) enabled, distributed software systems that are inter-connected through the Internet - which is being transformed into the Internet of Things (IoT) – is highly complex, thus requiring the collaboration of various stakeholders with different concerns, viewpoints and vocabularies. For instance, while a data scientist is concerned with analytics modeling, a data engineer is focused on analytics operations, and a software engineer may have a range of other generic and application-specific concerns, such as modularity, reliability, maintainability, and (re-)usability. Hence, the challenge of designing, implementing and maintaining AI-intensive software systems extends beyond software engineering, and additionally requires AI Engineering. This doctoral dissertation proposes a novel approach, called ML-Quadrat, to support enhanced development of smart services for the IoT and smart Cyber-Physical Systems (CPS). ML-Quadrat offers abstraction and automation to the software engineering processes, as well as supporting Data Analytics and Machine Learning (DAML) practices. This is realized in an integrated and seamless manner by implementing an open-source software modeling tool to validate the proposed approach. This tool supports automated generation of the full life-cycle implementation of smart IoT services. Further, ML-Quadrat supports the execution of automated DAML tasks by providing automated ML in the background, and, through model-checking constraints and hints, at design-time. ML-Quadrat has been validated empirically, through experiments using two case studies in the vertical IoT use case domains of smart energy systems and predictive maintenance, as well as, through expert interviews. Finally, we combine AI models (specifically ML models) with software models through providing integrative support for both Model-Driven Software Engineering (MDSE) and Model-Driven AI Engineering (MDAE). Our model transformations not only generate the software source code and scripts out of the software models, but also generate, train, deploy and re-train ML models as required.
German Abstract (‘Zusammenfassung’)

# Contents

Abstract v

German Abstract (‘Zusammenfassung’) vii

Contents ix

List of Figures xv

List of Tables xxi

1 Introduction 1
   1.1 Motivation ................................................. 1
   1.2 Problem statement ........................................... 2
   1.3 Contributions ............................................... 4
   1.4 Publications ............................................... 5
   1.5 Structure ................................................... 7

2 Background 9
   2.1 The Internet of Things (IoT) and Cyber-Physical Systems (CPS) . . . . 9
   2.2 Machine Learning (ML), data science, and automated ML ................... 11
   2.3 Software Engineering (SE) ........................................ 13

3 State of the Art 17

4 Research Methodology 21
   4.1 Research questions ........................................... 22
   4.2 Research methods ............................................ 23

5 Proposed Approach 27
   5.1 Framework architecture ....................................... 27
   5.2 Formalization ............................................... 41
CONTENTS

5.3 ML-enabled MDSE models for smart IoT services ......................... 46
5.4 AI-enabled architecture frameworks ..................................... 47

6 Implementation, Validation, and Evaluation ............................... 57
  6.1 Validation method .................................................. 57
  6.2 Open-source prototype .............................................. 60
  6.3 Validating RQ 1 ..................................................... 69
  6.4 Validating RQ 2 ..................................................... 76
  6.5 Validating RQ 3 ..................................................... 84
  6.6 Validating RQ 4 ..................................................... 87
  6.7 Discussion and threats to validity .................................. 103

7 Conclusion and Outlook .................................................. 143
  7.1 Summary of contributions ........................................... 143
  7.2 Limitations .......................................................... 144
  7.3 Future work .......................................................... 146

Bibliography .............................................................. 149
Acknowledgments

I am sincerely grateful to my parents, who enabled me to thrive. Also, I am proud of my two younger brothers, who achieved high academic and professional ranks. Moreover, I am grateful to my wonderful wife, who helped me in establishing a fantastic family, and gave birth to our daughter, who has increased our motivation and joy immensely.

Further, I would like to deeply thank my doctoral supervisor Prof. Dr. Stephan Günnemann, who leads the chair for Data Analytics and Machine Learning (DAML), as well as the Munich Data Science Institute (MDSI) at Technical University of Munich (TUM), Germany, for his kind and continuous support of my PhD. Moreover, I am grateful to my co-supervisor Prof. Atta Badii from University of Reading, United Kingdom, for his excellent co-advisory role. They both supported me scientifically and intellectually at the critical times of my PhD. I also appreciate Prof. Dr. Jörg Ott joining my dissertation examination committee as the chair.

I would like to also acknowledge the TUM Graduate School scholarship for my Executive MBA in Innovation and Business creation at TUM, in collaboration with University of California, Berkeley, USA, during my PhD, as well as the grants of the German Federal Ministry for Education and Research, in collaboration with Software AG and EIT ICT Labs (Software Campus), as well as the German Federal Ministry for Economic Affairs and Climate Action, and the European Social Fund (i.e., EXIST, which co-funded our spin-off DriotData). Further, I would like to thank the evaluators, interviewees, and survey participants for supporting my PhD study.

Additionally, I had the opportunity of being inspired, mentored, or advised by the following individuals during and before my doctoral studies, and I am thankful to them: Prof. M. Challenger, Prof. Dr. M. Broy, Prof. Dr. H. Krcmar, Prof. Dr. G. Neumann, Prof. M. Khansari, PD Dr. B. Schätz, Dr. M. Sayih, and Mr. S. Rössler. Last but not least, I am truly grateful of the chance I had to work with my great colleagues at TUM, DriotData, and fortiss in Munich, Germany, and elsewhere, as well as my amazing students, whom I advised at TUM: U. Wattanavaekin, A. Lungu, A. Mituca, C. Pröschel, D. Avdiu, and G. Wedler.
To my late supervisor Bernhard Schätz, who passed away five years ago [1],

and my academic ancestor, Friedrich Leibniz, who graduated 400 years ago [2].
List of Figures

1.1 Synthesizing all artifacts, including ML models, out of software models. The ML models are seamlessly integrated with the generated source code. 5

5.1 The UML Use case diagram of the proposed approach. ................. 33

5.2 The UML Activity diagram illustrating the usual workflow of using the proposed approach. Figure from [3]. ................................. 34

5.3 The UML Class diagram showing part of the meta-model of the proposed DSML. Figure from [3]. ................................. 36

5.4 Part of the Ecore meta-model of the proposed DSML. The red stars mark some of our extensions to the Ecore meta-model of the prior work ThingML [4]. Figure from [5]. ................................. 37

5.5 A sample ML model, namely a Probabilistic Graphical Model (PGM). Inspired by [6]. ................................. 40

5.6 A sample behavioral software model in MDSE using a Finite-State Machine (FSM). ................................. 40

5.7 The proposed reference model for DEA using the UML class diagram. Disciplines and individuals are marked in red and yellow, respectively. PMML [7, 8] and PFA [9, 10] are standards of the Data Mining Group (DMG). ................................. 54

5.8 The proposed reference model for ML using the UML class diagram. ................................. 55

6.1 The use case scenario concerning load monitoring using smart meters (i.e., the intrusive way) for validating RQ 1. ................................. 59

6.2 The use case scenario concerning the predictive maintenance of a hydraulic system for validating RQ 2. Figure inspired by [11]. ................................. 60

6.3 The UML Component diagram illustrating part of the logical architecture view of our prototype. ................................. 61

6.15 Part of the PIM for training the ML model. ................................. 81

6.16 The behavioral model of the PIM for training the ML model. ................................. 82
LIST OF FIGURES

6.17 Part of the PSM for training the ML model on an x86 Linux platform. The PIM of Figure 6.15 is imported here. ........................................... 82
6.18 Part of the PIM for predictions using the ML model. .......................... 83
6.19 The behavioral model of the PIM for predictions using the ML model. .................................................................................................................. 83
6.20 Part of the PSM for predictions using the ML model on an x86 Linux platform. The PIM of Figure 6.18 is imported here. ........................... 84
6.4 The textual model editor showing part of a sample model for the PingPong example. ........................................................................................................ 106
6.5 The graphical, EMF tree-/form-based model editor showing part of a sample model for the PingPong example. .................................................. 107
6.6 The web-based prototype showing part of a sample model for the PingPong example. ...................................................................................................... 108
6.7 The state machines modeling the behaviors of the three things of the smart ping-pong example. ................................................................. 108
6.8 Part of the model instance of the smart ping-pong example. .................. 109
6.9 Part of the model instance of the smart ping-pong example (continued). . 110
6.10 The data analytics part of the software model instance in textual form. .... 110
6.11 The data analytics part of the software model instance in the EMF tree-based editor. ......................................................................................... 111
6.12 The data analytics part of the software model instance that shows the black box ML mode in textual form. ...................................................... 112
6.13 The values of the loss function and the accuracy during the training of the ML model for validating RQ 2. .......................................................... 112
6.14 From PIM to PSMs, and the full generation of the IoT services for the target platforms. The images of Arduino and Raspberry Pi are from [12] and [13], respectively. ................................................................. 113
6.21 The tree-structured search space for the AutoML engine. ..................... 113
6.22 The overall architecture of the IoT service for the case study validating RQ 3. ................................................................................................. 114
6.23 The behavioral model of the energy disaggregation server of the smart grid in the IoT service for the case study validating RQ 3. ............... 115
6.24 The disaggregation accuracy using different ML methods. Color codes: dark blue, orange, gray, yellow, and light blue represent the results for the fridge, lights, sockets, washer/dryer, and average for all appliances, respectively. ........................................................................ 116
LIST OF FIGURES

6.25 The gender distribution of the online survey participants. .......................... 116
6.26 The age group distribution of the online survey participants. ...................... 116
6.27 The highest academic degree distribution of the online survey participants. 2.38% preferred not to disclose. ................................................................. 117
6.28 The education background distribution of the online survey participants. ...... 117
6.29 The current job or occupation distribution of the online survey participants. 118
6.30 The current role or position distribution of the online survey participants. .... 119
6.31 The familiarity of the online survey participants with the related fields/topics. ................................................................. 120
6.32 The expertise level distribution of the online survey participants in the DEA/ML fields. ................................................................. 120
6.33 The expertise level distribution of the online survey participants in SE. ...... 121
6.34 Answers of the online survey participants to question 1 in section 1 of the questionnaire. The best model kind and notation for analytics modeling, and the communication of data scientists with their peers. .................. 121
6.35 Answers of the online survey participants to question 2 in section 1 of the questionnaire. The best model kind and notation for analytics operations, and the communication of data engineers with their peers. ................. 122
6.36 Answers of the online survey participants to question 3 in section 1 of the questionnaire. The best model kind and notation for describing the architecture from the viewpoint of a data scientist. .................. 123
6.37 Answers of the online survey participants to question 4 in section 1 of the questionnaire. The best model kind and notation for describing the architecture from the viewpoint of a data engineer. .................. 124
6.38 Answers of the online survey participants to question 1 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with the end-users of software and information systems. ................................................................. 125
6.39 Answers of the online survey participants to question 2 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with the business stakeholders of software, information systems, and enterprises. .................. 126
6.40 Answers of the online survey participants to question 3 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with safety and regulatory compliance engineers, data protection (privacy) officers and ethics committees/boards. .......................................................... 127

6.41 Answers of the online survey participants to question 4 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with database engineers/designers. . 128

6.42 Answers of the online survey participants to question 5 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with software engineers and software architects. .......................................................... 129

6.43 Answers of the online survey participants to question 6 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with system engineers and network experts. .......................................................... 130

6.44 Answers of the online survey participants to question 7 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with security experts. . . . . . . . . 131

6.45 Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which end-users will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level. . . . 132

6.46 Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which ethics committees/boards will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level. .......................................................... 133

6.47 Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which data protection (privacy) officers will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level. .......................................................... 134

6.48 Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which data engineers will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level. 135
6.49 Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which database engineers/designers will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level. ......................................................... 136

6.50 Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which business stakeholders will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level. ......................................................... 137

6.51 Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which safety and regulatory compliance engineers will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level. ......................................................... 138

6.52 Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which data scientists (including Machine Learning engineers) will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level. ......................................................... 139

6.53 Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which software engineers and software architects will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level. ......................................................... 140

6.54 Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which domain/knowledge engineers will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level. ......................................................... 141

6.55 Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which security experts will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level. 141
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Related work compared to the proposed approach (ML-Quadrat [3, 5, 14, 15, 16, 17, 18])</td>
<td>19</td>
</tr>
<tr>
<td>5.1</td>
<td>The proposed architectural viewpoints concerning DEA/ML</td>
<td>53</td>
</tr>
<tr>
<td>5.2</td>
<td>The proposed architectural view notations (model kinds) concerning DEA/ML</td>
<td>56</td>
</tr>
<tr>
<td>6.1</td>
<td>The expertise fields and levels of the evaluators</td>
<td>74</td>
</tr>
<tr>
<td>6.2</td>
<td>The scores of the 4 evaluators (Eval. #1-4)</td>
<td>75</td>
</tr>
<tr>
<td>6.3</td>
<td>Experimental results for validating RQ 2. RPI, Ard., and Q. stand for Raspberry Pi, Arduino, and Quantized, respectively.</td>
<td>84</td>
</tr>
<tr>
<td>6.4</td>
<td>The average MAE of the studied NIALM approaches</td>
<td>87</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Motivation

Smart systems, which can mimic the human intelligence and cognition to some extent, are becoming prevalent in our everyday lives: from healthcare, mobility, education, finance, economy, defense, security, and energy to the entertainment and leisure industry. Wherever we look today, we can see the trend towards innovation of increasingly intelligent / smart systems. This trend relies primarily on Artificial Intelligence (AI). In particular, a sub-discipline of AI, namely Machine Learning (ML) is currently revolutionizing almost every industry. Even in academia, a large proportion of scientific papers published recent years focused on applications of ML in their specific domains. Thus, ML is currently a vital topic in many domains.

A lot of theorems, algorithms, and methods on which ML technologies have been based, are not new. They mostly date back to the 18th century. However, the abrupt fall of the price of computers, in particular, embedded microcontrollers and sensors, the prevalence of connectivity to the global computer network (i.e, the Internet), as well as the unprecedented advances in computer hardware technologies, have enabled numerous new applications of ML in real-world problems. There are ubiquitous sensors in the surrounding environment, and inside equipment, appliances, objects, and devices. They provide massive datasets and data streams. Thanks to the ongoing expansion of the Internet of Things (IoT), many such devices are connected to the Internet, thus making the recorded data available. Furthermore, the IoT enables more computational resources for processing this data since distributed computational resources can be exploited through the global network. In addition, the exponential increase of the computational capacities
1 Introduction

that was formulated in Moore’s “law”\(^1\) lets us unlock the value of the data in the ways that were previously impracticable.

The above-mentioned paradigm shift has enabled a new class of complex systems of systems that consist of distributed and heterogeneous computing devices, including embedded microcontrollers, sensors and actuators, which can collaborate with each other, often in a dynamic and ad-hoc manner, to achieve a common goal. These systems, which bring the physical and the cyber/virtual/digital worlds together, are called Cyber-Physical Systems (CPS) [20]. A CPS may or may not be connected to the IoT. In fact, CPS and IoT are closely related concepts with “distinct origins but overlapping definitions” [21].

In this doctoral dissertation, we concentrate on IoT-connected CPSs that deploy AI, specifically ML, thus called smart CPSs. The core idea is to enable software developers to use ML methods and techniques efficiently in order to create smart services for the IoT, which can be deployed on heterogeneous and distributed platforms of smart CPSs.

1.2 Problem statement

There is a global shortage of skilled labor in the area of Information and Communication Technologies (ICT) in general, and specifically in the field of data science, which inherently includes ML, Data Analytics (DA) and Data Engineering (DE). Therefore, in many cases, software engineers themselves may have to design, implement, test, simulate, verify, deploy, and maintain ML-enabled software systems, such as smart CPSs, on their own. They may need to use DAML methods and techniques, and practice DE, without having deep knowledge and skills in these fields. For instance, a software developer might need to customize or modernize an existing ML component of a system.

In any case, regardless of ML, developing software for the highly heterogeneous and distributed IoT services and CPSs is challenging. In contrast to traditional distributed software-intensive systems, CPSs pose three new challenging dimensions of complexity: (i) The cross-* dimension with issues, such as cross-application domains, cross-technologies, and cross-organizations. (ii) The self-* dimension with issues, such as self-

\(^1\)Based on empirical observations, Moore’s law (which is not really a law, as in Physics, for example) states that the number of transistors in an Integrated Circuit (IC) roughly doubles every two years. This law might soon become outdated as the trend cannot continue indefinitely, due to Physical limitations regarding the size of transistors. Hence, parallel and multi-core processor architectures, and more efficient algorithm technologies [19], as well as emerging technologies, such as Quantum Computing (QC, see Chapter 7) are expected to serve as the main drivers of computing breakthroughs in the future.
monitoring, self-diagnosis, self-learning, self-adapting, self-optimizing, and self-recovery from a non-deterministic state space. (iii) The live-* dimension with issues, such as live-configuration, live-update, and live-enhancement [22]. In addition, guaranteeing X-by-design might be desired, where X can be any non-functional requirement of the system, such as efficiency (time-to-Y, power consumption, or cost), prediction accuracy, precision and recall/sensitivity, throughput, latency, modularity, (re-)usability, concurrency, semantic interoperability (openness), security (software, data, and DAML-models), transparency, reliability, availability, privacy-preserving, accountability, auditability and self-audit, user-acceptance and social acceptability, ethical compliance, safety compliance, quality assurance, and asset management.

One way of handling the complexity of CPS/IoT is to adopt software and system models at different levels of abstraction to hide the unnecessary details, and devise automation mechanisms that can transform one model to another model, as well as to another artifact other than a model, such as source code in the target programming language(s) with the required APIs. This is exactly what the Model-Driven Engineering (MDE) paradigm, also known as Model-Based Engineering (MBE) can offer. In particular, in Software Engineering (SE), we are concerned with the Model-Driven Software Engineering (MDSE) paradigm, also known as Model-Based Software Engineering (MBSE). The underlying idea has been around at least since the 1980s when Computer-Aided Software Engineering (CASE) tools were supposed to generate skeletons of the code out of model instances that would conform to fixed general-purpose modeling languages with their one-size-fits-all code generators [23].

In this work, we are interested in the Domain-Specific Modeling (DSM) methodology with full code generation as proposed by Kelly and Tolvanen [23]. This is already applied to the IoT/CPS domain, for example, via the ThingML (Things Modeling Language) project [4, 24, 25, 26]. One may use the APIs of different libraries and communication protocols at the design/modeling time. The source code, build scripts, documentation, and optionally the Unified Modeling Language (UML) diagrams, can be generated automatically out of the software models through model transformations (e.g., code generators that are also ambiguously called “compilers” in the ThingML project [4, 24, 25, 26]). However, none of the libraries and frameworks for DAML are supported at the modeling level. This poses a considerable challenge to the practitioners aiming for model-driven development of ML-enabled/ML-intensive software systems, such as smart CPS and smart IoT services.
1.3 Contributions

The contribution of this work is to address the above-mentioned problem by providing a holistic approach for model-driven software and AI engineering, or more precisely, MDSE and ML. Software models and DAML models are integrated to enable software models to build, train, test, deploy, and re-train ML models automatically. Also, other parts of the DAML pipelines, such as data preparation, as well as some analytics operations (data engineering) are taken into account. To this aim, the Domain-Specific Modeling Language (DSML) of the above-mentioned tool ThingML [4, 24, 25, 26] is extended.

First, we showed that the fully automated process concerning code generation out of the software model instances can be maintained, while supporting new target platforms and libraries for ML. For instance, we enabled Python code generation for the DAML part of smart IoT services. The Python code could deploy the APIs of Keras [27] with the TensorFlow [28] backend, and Scikit-Learn [29]. This could be useful to software developers, and increase their productivity, as well as their satisfaction, as illustrated in Chapter 6. This contribution is related to Research Question 1 (RQ 1) set out in Section 4.1. Figure 1.1 illustrates the core idea regarding synthesizing all artifacts, including ML models, out of software models in the proposed approach, called ML-Quadrat [3, 5, 14, 15, 16, 17, 18].

Second, we supported both Platform-Independent Models (PIM) and Platform-Specific Models (PSM). Code generation is only possible out of the latter. However, the former is used as a model with a higher level of abstraction that is parallel to the PIM level of the Model-Driven Architecture (MDA) framework [30, 31], and the Meta-Object Facility (MOF) [32, 33] standard. This contribution is related to RQ 2 set out in Section 4.1.

Moreover, we enabled Automated ML (AutoML) in the said model-driven process, meaning that certain design-time decisions of the practitioners concerning ML, such as the selection of the ML model architecture/family, as well as the hyperparameter choices, could be assisted in an automated manner. This could be useful for non-experts (i.e., novice practitioners) in ML, but might also be beneficial to ML experts in certain cases, as we can see in Chapter 6. This contribution is related to RQ 3 set out in Section 4.1.

Finally, as an additional contribution, we showed that software architecture frameworks must be updated to become AI/ML-enabled, and proposed a new research direction at the intersection of software and systems engineering with AI. This is based on RQ 4 set out in Section 4.1.
As of today, there exist six peer-reviewed research publications for this doctoral dissertation, in which the author is the first (co-)author. These are mentioned below:


1 Introduction


Moreover, the following papers are currently under peer-review:


2. Workshop vision/position paper: Moin, A., Challenger, M., Badii, A., Güninemann, S., MDE4QAI: Towards Model-Driven Engineering for Quantum Artificial Intelligence, Preprint available on arXiv [34].

1.5 Structure

This dissertation is structured as follows: Chapter 2 offers some required background information on a number of selected topics that are relevant to this work. Afterwards, we review the literature and show the position of this doctoral research with respect to the state of the art in Chapter 3. Moreover, we elaborate on the research methodology and the research questions in Chapter 4. This will be followed by proposing the novel approach of this doctoral study in Chapter 5 that is implemented, validated, and evaluated in Chapter 6. Finally, Chapter 7 concludes and points out a number of possible future directions for the research work of this doctoral dissertation.
2 Background

In this chapter, some background information on a number of related topics is provided.

2.1 The Internet of Things (IoT) and Cyber-Physical Systems (CPS)

The IoT is based on the concept of ubiquitous (pervasive) computing where sensors and actuators will become capable of connecting to the global computer network, called the Internet/IoT, through a unique addressing scheme, and will be able to interact and collaborate with each other [36]. Thus, in the same way as servers, PCs, laptops, tablets, and smartphones, other network-enabled computing devices ranging from smart wearables, such as smartwatches, to embedded microcontrollers, will also be able to become involved in interactions and collaborations for delivering smart services.

Everyday objects which are not capable of becoming network-connected since they lack a computer chip, will become uniquely identifiable via technologies, such as Radio-Frequency Identification (RFID) tags [36]. This will lead to having smart home appliances, smart infrastructures, smart and connected mobility, smart factories, smart hospitals, smart energy systems, smart museums, and in general, smart cities and communities.

The ongoing expansion of the Internet into the IoT, together with other ongoing trends, have enabled the next industrial revolution. For example, the advances of AI and big data analytics technologies, and the progress made even beyond the ICT sector, such as in additive manufacturing (3D printing), thanks to the expiry of the related patents for 3D printing in the recent decade, have been contributing to this.

The first industrial revolution in the 18th century was centered around mechanized machines, for example, using steam and water power. Moreover, the second industrial revolution that began in the 19th century was based upon electrification, mass production, and telecommunication. Some intellectuals, in particular in the US, call the ongoing industrial revolution, that is primarily based on the IoT, the third industrial
revolution, whereas others, in particular in Europe (e.g., in Germany) or in Japan see it as the fourth industrial revolution. The latter group often refer to the shift from analog to digital electronics, which enabled industrial automation, and was initiated in the late 20th century, as the third industrial revolution. It is clear that the IoT and CPSs will affect our lives in almost every aspect disproportionately [20, 36].

The notions of CPS and IoT are two sides of the same coin. They both refer to the integration of digital systems with network and computational capabilities in the physical world [21]. The applications of CPSs and the IoT have a broad spectrum, ranging “from intelligent vehicles to advanced manufacturing systems, in sectors as diverse as energy, agriculture, smart cities, and beyond” [21]. Note that some CPSs might not be connected to the Internet/IoT, for example for security and privacy reasons. For instance, a smart microgrid in a local region, which is an example of a CPS, might be connected to the IoT, whereas another grid that supplies electricity to the critical infrastructure of a city might be kept isolated from the IoT to mitigate the risk of cyber threats.

There also exist other examples of a CPS, such as an Unmanned Aerial Vehicle (UAV), an airplane, or a modern car. All of them have both physical (i.e., hardware) and cyber (i.e., software) components that are closely tied to each other. The former is what system engineers are trained for, whereas the latter is something that software engineers master. As CPSs and the IoT are becoming more prevalent, there is an urgent need for close collaboration between the software engineering, systems engineering, AI, and computer networks communities. In addition, MDE/MBE is widely adopted in software and systems engineering for CPSs/IoT. Note that we do not mean model-based or model-driven design in the sense that only the requirements and specifications of the System under Consideration (SuC) are captured in the abstract models, but an inherently model-driven approach in which models are used for analysis and synthesis of the SuC as well. Since AI, in particular, ML, is spreading into various components of almost every system at the present time at a fast pace, it is necessary to handle this aspect in the model-driven design and engineering of smart CPSs and smart IoT services [37].

This doctoral dissertation focuses on MDSE for ML-enabled CPSs and IoT services.
2.2 Machine Learning (ML), data science, and automated ML

Just like CPSs and the IoT, ML and Pattern Recognition (PR) are two facets of the same coin. The core idea is automatically discovering (ir)regularities in data by using mathematical and computational algorithms for the purpose of taking actions, such as making predictions, or classifying the data into different categories. The diverse set of ML model architectures or families includes, but is not limited to linear models for regression and classification (e.g., logistic regression), Decision Trees (DTs), ensemble methods (such as Random Forests, abbreviated as RFs), kernel methods (e.g., Gaussian Processes, abbreviated as GPs), sparse kernel machines (e.g., Support Vector Machines, abbreviated as SVMs), Artificial Neural Networks, abbreviated as ANNs (e.g., Multi-Layer Perceptrons, abbreviated as MLPs), Probabilistic Graphical Models, abbreviated as PGMs (e.g., Bayesian Networks), and mixture models (e.g., Gaussian Mixture Models, abbreviated as GMMs) [6].

In the recent years, various deep ANN architectures with multiple hidden layers have become dominant in the academia and industry, due to their proven power in solving challenging real-world problems. This includes, in particular, Recurrent Neural Networks (RNNs) [38], such as Long Short-Term Memories (LSTMs) [39], Gated Recurrent Units (GRUs) [40], and their other variations, Generative Adversarial Networks (GANs) [41], Graph Neural Networks (GNNs) [42], Bayesian Neural Networks (BNNs) [43], Bayesian Deep Learning (BDL) [44], transformers [45], such as Bidirectional Encoder Representations from Transformers (BERT) [46], and Deep Belief Network (DBN) [47] (based on Restricted Boltzmann Machines).

However, the performance of ML models in general, and ANNs, specifically, depends on a number of factors, such as the availability of often a very large dataset (i.e., usually in the order of millions of instances)\(^1\), an appropriate data preparation process (e.g., data cleaning, as well as standardization of numeric values in the case of ANNs), and the choice of ML model hyperparameters, such as the choice of the learning algorithm or optimizer (e.g., Adam), as well as the initialization of the model weights. Despite some best practices, the state of practice in many cases is still trial-and-error [48], and following the intuition of the practitioners.

\(^1\)Typically, by a small-sized, medium-sized, large-sized, or very large-sized dataset, we mean a dataset in the order of less than 10,000, between 10,000 and 100,000, between 100,000 and one million, and larger than one million data instances, respectively.
The field of data science uses ML algorithms, methods and techniques, “to solve problems in science, commerce, healthcare, government, the humanities, and many other fields of human endeavor” [49]. Data science used to be called data mining in the 1990s, and later big data in the 2010s [49]. Regardless of the buzz words and hype, the need for this discipline in the present time, with massive amounts of data and data streams that are being generated every second on the Internet/IoT and elsewhere, is obvious. It is necessary to unlock the value of the data that we possess or have legitimate access to them.

Note that ML models serve as only one category of DA models. DA models might be statistical models. In this case, the model is considered as an underlying probability distribution, from which the observed data are presumably drawn. Alternatively, one may consider a model for a dataset to be a summarization or an approximation of its data instances. Moreover, some models represent a dataset by its most extreme examples [49].

With the large-scale computing systems of today that often require a high throughput and a low latency, data science typically goes beyond analytics modeling (e.g., creating and training ML models), and additionally comprises analytics operations [9]. The latter involves efficient and scalable methods and techniques for deploying scoring engines (i.e., predictive models), and scalable data processing or stream processing methods and technologies, such as the ones of the Apache Hadoop and Spark ecosystems. While analytics modeling is known as Data Analytics (DA), the field of analytics operations is called Data Engineering (DE). Data science includes both DA and DE. Hence, it is sometimes called Data Engineering and Analytics (DEA).

Further, the recently emerged research and development area of Automated ML (AutoML) aims to (semi-)automate the manual tasks of ML practitioners and data scientists (i.e., those who practice DEA). For instance, some AutoML approaches focus on data pre-processing, whereas others deal with ML model family/architecture selection, for example, Neural Architecture Search (NAS), and/or hyperparameter optimization [48].

In this doctoral study, we primarily concentrate on DA, ML, and AutoML. However, some aspects of DE, such as automated deployment of ML models, as well as re-training them, will also be addressed. The latter is also related to an emerging field, called ML-Ops (similar to DevOps). “ML Ops is a set of practices that combines Machine Learning, DevOps and Data Engineering, which aims to deploy and maintain ML systems in production reliably and efficiently” [50].
2.3 Software Engineering (SE)

In this section, we briefly review a number of related subjects in SE.

SE methodologies

Since the late 1960s, various methodologies have been proposed for the software development process. These include, but are not limited to the waterfall model, Rapid Application Development (RAD), Rational Unified Process (RUP), extreme programming (XP), and agile methodologies. The latter is currently the state of the art. For instance, the Scrum framework is based on agile software development. There have also been extensions and complementary approaches, such as DevOps that combines a number of best practices of agile software development (Dev) and IT operations (Ops) to shorten the System Development Life-Cycle (SDLC), and provide continuous delivery of high-quality software. Another interesting variation of DevOps is ArchOps in which software architecture models are treated as first-class artifacts. This is analogous to the MDSE paradigm. In this doctoral dissertation, we are interested in the DSM methodology that is explained in the work of Kelly and Tolvanen [23].

Programming paradigms

There exist a number of programming paradigms that are relevant to this doctoral study. The MDSE paradigm [51] is the main pillar of our approach. Other ones include, but are not limited to, object-oriented programming, aspect-oriented programming, event-driven programming, automata-based programming, agent-oriented programming, dataflow programming, as well as both imperative and declarative programming. While they might sound contradictory, we need a combination of different paradigms to cover the requirements of smart CPSs and the IoT. For instance, imperative programming is used in the resulting systems (i.e., smart CPSs and IoT services) that we automatically generate their code (e.g., in Java and Python) out of software models, whereas a mixture of declarative and imperative paradigms are deployed in the DSML at the design/development/modeling stage.

Architecture and Separation of Concerns (SoC)

The ISO/IEC/IEEE 42010:2011 standard for architecture descriptions in systems and software engineering [52] defines the architecture of a system as fundamental concepts
or properties of the “system in its environment embodied in its elements, relationships, and in the principles of its design and evolution” [52]. An Architecture Description (AD) is a work product that expresses an architecture. Any stakeholder of a system has various concerns concerning the System-of-Interest (SoI) in relation to its environment. Concerns arise throughout the life-cycle of the system from the system requirements, design choices, implementation, and operations. Examples for concerns include, but are not limited to functionality, feasibility, performance, resource utilization, reliability, security, information assurance, complexity, evolvability, openness, concurrency, autonomy, cost, schedule, and quality of service [52]. Moreover, the distribution transparencies listed in the Reference Model of Open Distributed Processing (RM-ODP) (i.e., ISO/IEC 10746-1) standard [53] (namely, access transparency, failure transparency, location transparency, migration transparency, persistence transparency, relocation transparency, replication transparency, and transaction transparency), as well as the software properties that are stated in the SQUARE (i.e., ISO/IEC 25010:2011, 4.2) standard [54] also hold as examples for stakeholder concerns [52].

Separation of Concerns (SoC), which is a vital design principle in SE, can be applied to different layers of abstraction. On a lower level, it is interpreted as the modularity of the software system implementation, with information hiding and encapsulation in modules that have well-defined interfaces. However, on a higher layer, it means describing the system architecture from the perspective of different sets of concern. This is what an architecture view does. It is a work product that expresses “the architecture of a system from the perspective of specific system concerns” [52]. Additionally, an architecture viewpoint is a work product that establishes “the conventions for the construction, interpretation and use of architecture views to frame specific system concerns” [52]. Further, a model kind is a set of conventions for a type of modeling. For instance, “data flow diagrams, class diagrams, Petri nets, balance sheets, organization charts and state transition models” are model kinds [52]. Moreover, architecting is the “process of conceiving, defining, expressing, documenting, communicating, certifying proper implementation of, maintaining and improving an architecture throughout a system’s life cycle” [52]. It is typically conducted in the context of a project or an organization. Last but not least, an architecture framework comprises “conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders” [52]. Popular examples of architecture frameworks of systems, software, and enterprises include, but are not limited to TOGAF [55, 56], DoDAF [57], TEAF [58], MODAF [59], and NAF [60]. Many of them are built on prior
work, mainly the Zachman Framework [61] and the “4+1” View Model of Software Architecture [62]. The Zachman framework is a generic framework that applies to both information systems and enterprise architectures. However, as its name suggests, the “4+1” View Model of Software Architecture is specific to software systems. Additionally, RM-ODP [53, 63, 64, 65] is concerned with the architecture of distributed information systems.

General-purpose vs. domain-specific programming & modeling languages

General-purpose, high-level programming languages, such as Java, Python, and C/C++, as well as general-purpose modeling languages, for example, the Unified Modeling Language (UML) [66] and the Systems Modeling Language (SysML) [67], are not tied to any particular problem/application/use case, or solution domain. Hence, they are general-purpose languages. In contrast to general-purpose languages, domain-specific programming and modeling languages concentrate on a particular domain. Some of them are tied to a specific problem (i.e., vertical) domain, for example, automotive, aerospace, finance, or energy, whereas others might be tailored to a particular solution (i.e., ICT) domain, for example, cloud computing, Multi-Agent Systems (MAS), reactive systems, sensor networks, IoT/CPS, DEA/ML, or Quantum Computing (QC). A language often has an abstract syntax that must conform to a grammar or meta-model, a concrete syntax (e.g., textual or graphical), as well as semantics (static/structural and dynamic/behavioral). Domain-Specific (Modeling) Languages (DSL/DSML) are intentionally constrained in terms of expressiveness. This enables them to exhibit useful characteristics in terms of analysis and synthesis. For instance, model checking and verification in the early design phase in domain-specific MDE/MDSE (i.e., model-based program analysis), as well as full code generation out of models in an automated manner (i.e., model-based program synthesis) are possible via DSLs/DSMLs.

Model-Driven Architecture (MDA)

We elaborated on the architectural SoC above. The Object Management Group (OMG) has a different approach to architectural SoC. It has a framework, called Model-Driven Architecture (MDA) [30, 31], that provides architectural SoC through several layers of abstraction: Computation-Independent Models (CIM), Platform-Independent Models (PIM), and Platform-Specific Models (PSM). CIM is a business or domain model, thus
uses a vocabulary that is familiar to the domain experts, while hiding the Information Technology (IT) details. Additionally, a PIM includes the IT details, but exhibits a sufficient degree of independence from specific computing platforms by abstracting out their technical details. Further, a PSM combines the specifications of a PIM with the platform-specific details required on a particular computing platform [31]. MDA is not a specification or standard, but an approach to software design, development, and implementation that provides guidelines for structuring software specifications that are expressed as models [68]. MDA is based on several standards of the OMG, such as Meta-Object Facility (MOF) [32, 33], which is also an international standard.

In contrast to the above-mentioned notion of viewpoints, which is adopted in the ISO/IEC/IEEE 42010:2011 standard [52] and also used in our work, MDA considers horizontal viewpoints that are based on the levels of abstraction/detail above, namely CIM, PIM, and PSM. “A viewpoint in MDA encompasses the whole underlying system, but on its specific level of detail regarding the distance to a concrete system implementation” [69]. However, in the above-mentioned definition of viewpoints, it is the combination or union of all the vertical viewpoints of the system that provides the entire system architecture [69].
3 State of the Art

Both abstraction and automation, which are the pillars of MDSE, have already been introduced to the DAML field. Regarding the former, namely raising the level of abstraction, we may refer to the widely-used libraries and frameworks, such as TensorFlow [28] or Keras [27]. The latter is on a higher layer of abstraction than TensorFlow [28] since it can have multiple backends, including TensorFlow [28] and Theano [70]. In addition, there exist DEA workflow designers and workbenches, for example, KNIME [71], WEKA [72], MOA [73], and RapidMiner [74], as well as visualization toolkits, such as TensorBoard [75]. Furthermore, regarding the latter pillar, namely automation, we may refer to related work in the literature as follows. Concerning transforming one abstract representation (i.e., model) to another one, which is analogous to model-to-model transformations in MDSE, we see Model-Interchange Formats (MIF), for example, Predictive Model Markup Language (PMML) [8], Portable Format for Analytics (PFA) [10, 9], and Open Neural Network Exchange (ONNX) [76] as particularly relevant. For instance, ONNX supports building ANN ML models from various libraries and frameworks in an interoperable manner. Examples of these include TensorFlow [28], Keras [27], PyTorch [77], Scitkit-Learn [29], MXNET [78], Caffe2 (which is now part of PyTorch [77]), XLA (which is a domain-specific compiler for linear algebra that can accelerate TensorFlow models), Core ML [79] (that enables the integration of ML models into the iOS apps), and the Microsoft Cognitive Toolkit [80] (previously known as CNTK) [3].

Additionally, a parallel idea to the DSM methodology in MDSE with full code generation out of models [23] was proposed by Bishop through Infer.Net [81, 82]. However, rather than synthesizing the source code out of software models, as it is the case in DSM/MDSE, Infer.Net [81, 82] synthesized the source code out of ML models, in particular out of Probabilistic Graphical Models (PGMs). However, “they only supported C# for code generation. Although this approach has so far been the most relevant approach to the MDSE paradigm” [3], it had a major shortcoming for real-world IoT/CPS applications, where the expressiveness of PGMs (and other ML models) would not suffice to model the entire software system, and generate the full source code out of the model.
instances [3]. Note that the main conceptual and technical difference of our proposed approach, namely ML-Quadrat [3, 5, 14, 15, 16, 17, 18], with Infer.Net [81, 82] is the following: Instead of enabling ML models to act as software models, thus generating the implementation out of them, we enable software models to become capable of generating the full implementation, including ML models, and training them.

In addition, we are aware of a number of related work in domain-specific MDSE, such as PIM4Agents/DSML4MAS [83, 84] for Multi-Agent Systems (MAS), and AutoFOCUS [85] for embedded systems (which were both built based on the Eclipse Modeling Framework (EMF) [86] ecosystem), mbeddr [87] for embedded systems (which was based on the JetBrains MPS framework), and previous efforts in the form of UML profiles, such as MARTE [88] for real-time, embedded software of CPSs.

The approaches that focused on the IoT and CPSs include ThingML\(^1\) [4, 24, 25, 26], and HEADS [89, 90], which was based on ThingML [4, 24, 25, 26]. They supported the MDSE paradigm, specifically the DSM methodology with full code generation [23] for IoT services, which are inherently Heterogeneous and Distributed (HD). The proposed approach in this doctoral dissertation extends ThingML [4, 24, 25, 26].

Moreover, \(\mu\)-Kevoree [91] concentrated on Models@Runtime, thus fading out the borders between the design-time (modeling-time) and the runtime of IoT services. However, their target platforms for code generation were very limited. In fact, they only supported code generation for platforms, such as Java, which were not suitable for the resource-constrained IoT edge devices. These devices do not typically possess enough main memory and processor power to run a Java Virtual machine (JVM). In contrast, ThingML [4, 24, 25, 26] and HEADS [89, 90] supported code generation for a range of programming languages and platforms, including C for Posix and Arduino, as well as several communication protocols, including HTTP, MQTT, and CoAP.

However, the main shortcoming of the said prior works (i.e., ThingML [4, 24, 25, 26], HEADS [89, 90], and \(\mu\)-Kevoree [91]), was a lack of support for DAML at the modeling level. In other words, the practitioners using them could not deploy any DAML method, algorithm, or technique related to DAML while using those DSMLs. This gap was first pointed out in our research proposal for this doctoral dissertation in 2015, and published in our previous work in 2018 [16], which stated our position and illustrated our vision. Additionally, Hartmann et al. proposed GreyCat [92, 93, 94, 95], based on the Kevoree Modeling Framework (KMF) and \(\mu\)-Kevoree [91]. They integrated ML

\(^1\)Please note that ThingML stands for Things Modeling Language. Thus, ML in ThingML is not related to Machine Learning.
with software models in MDSE. Their work was conceptually similar to our approach, however, GreyCat [92, 93, 94, 95] also had the limitations of its prior work, namely, generating code for platforms that were not the most widely used IoT edge platforms. In fact, they only supported Java and Javascript/Typescript code generation.

Table 3.1 compares the related work in the literature with the proposed approach ML-Quadrat [3, 5, 14, 15, 16, 17, 18].

Table 3.1: Related work compared to the proposed approach (ML-Quadrat [3, 5, 14, 15, 16, 17, 18].

<table>
<thead>
<tr>
<th>Description</th>
<th>Work</th>
<th>Full code gen.</th>
<th>DAML support</th>
<th>IoT / CPS domain</th>
<th>Model type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML libraries and frameworks</td>
<td>TensorFlow [28],</td>
<td></td>
<td></td>
<td></td>
<td>DAML models</td>
</tr>
<tr>
<td></td>
<td>Keras [27], Scikit-Learn [29], etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAML workflow designers</td>
<td>KNIME [71], Rapid-Miner [74], etc.</td>
<td></td>
<td></td>
<td></td>
<td>DAML models</td>
</tr>
<tr>
<td>Model Interchange Formats (MIF)</td>
<td>PMML [8], PFA [10, 9], ONNX [76]</td>
<td></td>
<td></td>
<td></td>
<td>DAML models</td>
</tr>
<tr>
<td>“Model-based” ML</td>
<td>Infer.Net [81, 82]</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>ML (PGM) &amp; SE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>models</td>
</tr>
<tr>
<td>MDE4IoT</td>
<td>ThingML [4, 24, 25, 26] and HEADS [89, 90]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>SE models</td>
</tr>
<tr>
<td>Models@Runtime</td>
<td>μ-Kerveore [91]</td>
<td></td>
<td>✓</td>
<td></td>
<td>Limited SE</td>
</tr>
<tr>
<td>Models@Runtime + ML</td>
<td>GreyCat [92, 93, 94, 95]</td>
<td></td>
<td></td>
<td></td>
<td>Limited SE &amp; DAML models</td>
</tr>
<tr>
<td>MDE4IoT + ML</td>
<td>ML-Quadrat [3, 5, 14, 15, 16, 17, 18]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>SE &amp; DAML models</td>
</tr>
</tbody>
</table>

Further, there have been other related works in the literature, which conducted research at the intersection of AI/ML/DEA and SE/MDSE. For example, the MODA
framework proposed by Benoit et al. [96] for MDE of data-centric systems was relevant. They identified three types of models, namely engineering models, scientific models, and ML models. Further, they recognized the following possible roles for models: descriptive, prescriptive, and predictive. Their MODA framework aimed for synergies by combining these models for the future complex, software-intensive, data-centric systems, ranging from CPSs to sociotechnical systems (i.e., “where humans are in the loop with software-based systems, e.g., crisis management systems” [96]). Another example of a study in this direction was the one of Breuker [97] in which DSMLs/DSLs for ML were explored, and the abstract syntax of a DSML based on Infer.Net [81, 82], which was mentioned above, was proposed. However, this was also very specific to PGMs (similar to Infer.Net [81, 82]). Also, Portugal et al. [98] conducted a preliminary survey of DSMLs/DSLs for ML and big data.

Finally, at the time of writing this doctoral dissertation, we became aware of other similar recent contributions by the PhD studies of Evgeny Kusmenko [99] and Jörg Christian Kirchhof [100] at RWTH Aachen university in Germany. In particular, the MontiAnna [101] and MontiThings [100, 102] DSMLs for the modeling of ML (with a focus on deep learning) and the IoT, respectively, are relevant to this doctoral dissertation. Their DSMLs were based on the MontiCore workbench [103, 104] and the MontiArc architecture description language [105]. Similar to this work, their solutions are also compatible with the EMF since their basis, namely the MontiCore workbench [103, 104] supports EMF integration. According to their Github documentation [102], MontiThings could generate C++ code. In contrast, we generate code for a range of platforms and programming languages, including C/C++, Java, and Python.
4 Research Methodology

The research methodology is a key element of every research work that aims for reliable outcomes. However, some of the works in the field of Computer Science, and in particular in the SE and AI/ML research communities of our time, lack a sufficient level of attention to sound methodologies and methods in many “scientific” studies. This has been pointed out by a number of researchers and sources (e.g., see [106]), and can be also clearly observed by reading the literature. In the following, we first provide some general definitions related to research, methodologies, and methods. Then, we state our Research Questions (RQs), and set out the research methods that we deploy to conduct this doctoral study.

Clough and Nutbrown [107] collected alternative definitions for *research*, for example, “a way of systematic investigation of a phenomenon or area of activity” [107]. The overall goal of research is to enable the researchers to extend their knowledge, or to explore a theory in science and/or engineering from a particular perspective. In each field of science and engineering, several *research methods* are established, in order to let the researchers validate their *hypotheses* or *RQs*. “A *research methodology* shows how RQs are articulated with questions asked in the field” [107]. If we consider the research methods as some *ingredients* of research, which are analogues to the ingredients of a meal in cooking, a research methodology would then be the framework that justifies the choice of a particular recipe for cooking a specific meal. Thus, the research methodology must explain why a particular research method or a set of methods have been chosen for validating a specific hypothesis or answering a certain RQ. For instance, if the research method involves one-on-one interviews, or a survey through a self-administered online questionnaire, then the methodology should explain why the researcher has decided to carry this out, and what implications it might have. Moreover, it should clarify why they have chosen a certain number of participants, how they found them, how they ordered the interviews (e.g., random), whether they (randomly) changed the order of the presented questions, and similar concerns to minimize any possible bias. For instance, in the case of the interview methods, the sampling method for choosing the interviewees might have
been convenience sampling rather than random sampling. As another example, in the case of the experimental methods, there might have been no control group at all, thus no Randomized Controlled Trial/Experiment (RCT/RCE), or not a fully randomized one, but just a quasi-random experiment. However, these details must be clearly stated in the methodology, and the implications, as well as the possible constraints must be pointed out, such that the researchers themselves and others become aware of them [107].

Here, we are particularly interested in empirical research methods, which are based on evidence. As proposed by Adrianus de Groot in 1969, the typical empirical cycle includes observation, induction (i.e., forming hypotheses or RQs), deduction (i.e., forming experiments to test the hypotheses or answer the RQs), testing, and evaluation (i.e., interpretation of the data) [108, 109].

4.1 Research questions

Research Question (RQ) 1

We formulate the first RQ of this doctoral dissertation as follows:

Can we add DAML concepts to the modeling level of an existing state-of-the-art modeling language and tool for the domain of IoT/CPS that offers full code generation, for example, ThingML [24, 4, 25, 26], such that we generate ML models, train, deploy, and re-train them automatically, while still maintaining the automated generation of the full source code of the target services/applications [14]?

RQ 2

The second RQ of this doctoral study is also related to MDSE:

Is the option of keeping two separate modeling layers, namely Platform-Independent Model (PIM) and Platform-Specific Model (PSM) feasible and deployable [14]? More specifically, can we use the same PIM with different PSMs that extend this PIM, for example, for supporting TinyML on highly resource-constrained IoT edge devices, while also supporting ML on more powerful systems with GPUs?

RQ 3

The third RQ of this doctoral dissertation is related to AutoML and is stated below:
Can we support the practitioners using the proposed approach in carrying out the required DAML tasks through AutoML? For instance, can we give them useful hints and tips either at the design/modeling time, or at the time of program synthesis (i.e., code generation)? Can we use AutoML to assist the practitioners in selecting the best ML model family/architecture, and making the most efficient choices for the hyper-parameters (e.g., the learning algorithm or optimizer)?

RQ 4

The fourth RQ of this PhD research is concerned with software architecture frameworks, for example, architecture viewpoints and views as follows:

Can we enable AI/ML in architecture frameworks, in particular, for software and information systems? Which stakeholders must be considered, and which viewpoints, views, and model kinds are necessary to be defined? Can we adapt and re-use any existing ones, for example, any existing UML diagrams for this purpose?

4.2 Research methods

Our research methods could be broadly categorized under empirical methods. Moreover, we deployed a mixture of qualitative and quantitative methods.

The first step was to ensure that the intended research work was new, interesting, and relevant. Based on our prior observations as practitioners, we postulated the RQs above. Through informal conversations, and exchanges of ideas with both the SE and the AI/ML communities, we found that this could be a need in our era, when software-intensive systems might be AI/ML-enabled. Therefore, we considered the RQs as interesting and relevant to the communities. In particular, the SE community was excited about this [5, 14, 16]. In addition, we conducted a thorough literature review to explore the related fields, and confirmed that the RQs were novel. To establish this, we used different combinations of a number of keywords, such as model-driven, model-based, software engineering, machine learning, artificial intelligence, domain-specific, cyber-physical systems, and internet of things on the Google search engine in 2016. We have added more literature to our bibliography through the conducted research work over the period 2017-2022.

It transpired that no prior work based on the RQs had been proposed, nor had realized this research [5, 14, 16]. The literature review was also necessary to find the exact gap
Research Methodology

in the state of the art, and to see how existing work might be suitable to be deployed as
the building blocks of our research. Hence, we decided to build on top of the software
development methodology and modeling tool of the open-source ThingML project [4, 24, 25, 26]. Our literature review included a qualitative analysis of the contributions of
the prior work in terms of a methodological approach to MDSE.

Further, in order to assess and validate RQs 1, 2, and 3, we deployed the experiments
and case study research methods. Newman [110] elaborated on “four broad categories
for engineering research: (i) Enhanced Analytical Modeling Techniques (abbreviated as
EM), (ii) Enhanced Solutions (abbreviated as ES), (iii) Radical Solutions (abbreviated
as RS), and (iv) Enhanced Tools and Methods (abbreviated as ET). Although his work
originated from the field of Human Computer Interaction (HCI), it was not specific to
HCI, but rather for any engineering research work” [14]. In accordance with the RQs,
our research falls under the latter category (i.e., enhanced tools and methods). Hence, as
suggested by Newman [110], we validated the proposed approach illustrated in Chapter
5 by implementing our modeling tool and methods, and through conducting a number
of experiments via several working examples for two case studies that are set out in
Chapter 6. Thus, we showed the feasibility, effectiveness, and efficiency of the proposed
approach [14]. Here, we used both qualitative and quantitative methods.

We briefly describe the validation method, including the data collection and the use
case scenarios, in Section 6.1, and elaborate on possible threats to validity in Section
6.7.

Last but not least, concerning the assessment and validation of RQ 4, we conducted
a literature review, in addition to a number of one-on-one interviews with experts, and
a self-administered online survey. Likewise, our methods here were both qualitative and
quantitative, but primarily the former. For the literature review, we first used different
combinations of a number of keywords, such as view, viewpoint, architecture framework,
machine learning, artificial intelligence, ai, cyber-physical systems, and internet of things
on the Google search engine in 2020 and 2021. We then, used several DAML textbooks
(such as [6, 49, 111]), and other sources (such as [9, 112, 113]) in order to develop our con-
ceptual reference model presented in Section 5.4. The expert interviews and the online
survey were primarily conducted to validate our drawn conclusions from the reference
model concerning the new stakeholders, viewpoints, views, and model kinds to address
AI/ML. However, they also contributed to the identified stakeholders, viewpoints, views,
and model kinds. We used convenience sampling for choosing the interview partners and
survey participants, and invited members of our international network via email and so-
cial media (mainly LinkedIn), as well as our peers, to join. A thorough explanation of the survey method can be found in Section 6.6.
5 Proposed Approach

In this chapter, we propose the novel approach of this doctoral dissertation, called ML-Quadrat [3, 5, 14, 15, 16, 17, 18]. First, we illustrate the overall architecture of the proposed approach in Section 5.1. Second, we formalize it in Section 5.2. Further, we elaborate on the core idea and the main contribution beyond the state of the art in Section 5.3. Additionally, we propose enhancing architecture frameworks with AI, and in particular ML in Section 5.4. This chapter is mainly based on our prior publications [3, 5, 14, 15, 16, 17, 18]. In particular, Sections 5.2 and 5.3 are mainly based on our prior publication [3].

5.1 Framework architecture

According to the adopted software development methodology for each software project, the work of developing and/or maintaining the software system in its life-cycle can be broken down into a number of milestones. The first milestone typically serves functional and non-functional requirements elicitation, analysis, and specification, whereas the last one would deal with testing and evaluation. We indicate these milestones with $MS_1$, $MS_2$, ..., $MS_n$, and the delivery date for each milestone $MS_i$ with $t_{MS_i}$. The greater the time taken over the milestones, namely $\sum_{m=2}^{n}(t_{MS_m} - t_{MS_{m-1}})$ will be, the greater the Time-To-Market (TTM), and thus the risks. Hence, this is another indirect factor contributing to the actual total project cost for an organization or enterprise that will either deploy the software or bring it to the market. Further, the required human resources (e.g., development time), the necessary skills (e.g., data scientists), as well as the needed computational (hardware/software) and communication resources for each of the milestones ($MS_x$) contribute to the total project cost as direct factors. In addition, another cost and stress driver in software projects is the quality of code, such as the number of bugs/failures. Being able to reduce the end-to-end $t_{MS_n} - t_{MS_1}$ development time, or the number of bugs, or to relax any of the constraints (e.g., regarding the required skill set) over any milestone would have a positive impact on the total cost.
5 Proposed Approach

While the cost and stress drivers are measured quantitatively, for example, via the number of days for the time, or the number of bug reports over a period for the quality, other aspects, such as the job satisfaction of practitioners can be measured qualitatively (e.g., see the empirical user study in Section 6.3).

The ultimate goal of our work is to enhance the performance of practitioners in software development projects in our domain of focus (namely smart CPS/IoT services), thus reducing the total project costs. A performance leap might be introduced by increasing the effectiveness and/or the efficiency. The former relates to building the right thing, whereas the latter concerns with building it in the right way that avoids unnecessary costs. The focus of this doctoral dissertation is primarily on increasing the efficiency. Our practitioner observations and heuristics (see Chapter 4), which were later validated empirically by domain experts (see Section 6.6), emphasized the vital role of software developers who were not professional data scientists, but had held data science positions. We assumed that this situation was caused by the global shortage of data scientists as it had previously been pointed out by various analysts (such as Gartner).

We argue that assisting practitioners with proper tools and methods to collaborate in different milestones of the projects, ranging from requirements engineering (e.g., for communication with different stakeholders) to implementation (including analytics modeling), and testing is crucial to the efficiency of software development for modern, ML-enabled CPS/IoT services. In particular, we expect the proposed fully automated approach to generating the desired software solutions derived from the abstract software models to lead to more productive software development (by enabling software developers to also take over data science tasks in an efficient manner), a higher level of practitioners (i.e., developers) job satisfaction, as well as a higher quality of the automatically generated software, compared to the classic, manual development approach.

However, quality can have various facets besides the number of software failures or bugs. In the context of this work, the following provides a fine-grained categorization to shed light on the different aspects of quality, which we refer to as the criteria of merit, for the resulting software systems based on the proposed approach:

1. **General SSE criteria of merit:** For example, usability, user-friendliness / ease of use, security, trustworthiness, modularity, efficiency, robustness, reliability / dependability, availability, throughput, complexity, scalability, concurrency, and transparency.
5.1 Framework architecture

2. Criteria of merit related to distributed computing, especially IoT/CPS services: For instance, semantic interoperability of network-centric, heterogeneous, embedded systems (e.g., with respect to the hardware architectures, operating systems, communication protocols, programming languages, and APIs), open system design (i.e., open-by-design), quality of service (including availability, reliability, and adaptability), failure handling, fault tolerance, and reflexive design, which could mean self-adaptive, self-learning, self-optimizing, self-monitoring, self-auditing, self-diagnosing, fault-tolerance and graceful recovery from failure (e.g., an aircraft can fly further even if one engine fails, and can somehow glide even if both engines fail), self-healing, self-repairing, self-accountable, self-expressive, and self-explanatory.

3. Criteria of merit pertaining to DAML/DE: These can be grouped into three sub-categories:

   a) DAML-centric criteria of merit: For example, performance in terms of ML performance metrics, such as:

      i. Accuracy, Precision, Recall (sensitivity), F1-Measure, and ROC curve area; Being capable of handling (class-) labeled, unlabeled, or partially labeled data (e.g., XGBOD [114]);

      ii. Being able to deal with data that are noisy, sparse, high-dimensional, and/or unbalanced.

      iii. Explainability (explainable-by-design ML models, such as PGMs, vs. ML models that are not explainable-by-design, such as ANNs).

      iv. Ethical and legal compliance-by-design, such as dignity-preserving, handling algorithmic bias (e.g., regarding diversity and inclusiveness), privacy-preserving support, security-by-design (e.g., adversarial attacks resistance), as well as social and environmental acceptability, including environmental sustainability considerations, for example, carbon emissions minimizing, re-usability, recyclability, or the ability to re-purpose (i.e., re-purposability).

   b) DE-centric criteria of merit: For instance, being capable of dealing with data annotations (i.e., semantic data vs. non-semantic data), batch data processing scalability (i.e., coping with large or very large datasets as defined in Section 2.2), data processing mode (i.e., online/stream processing vs. open-by-design).
5 Proposed Approach

offline/batch processing), as well as online data processing throughput and delay.

c) **AI/ML application functionality-centric criteria of merit:** For example, cooperative question answering, machine translation, predictive maintenance, and automatic summarization.

In addition, we define the criteria of merit below for the MDSE tool/environment itself:

1. How complete, expressive, and suitable is the abstract syntax of the DSML for the target domain?

2. How user-friendly (i.e., practitioner-friendly) and usable are the concrete syntax of the DSML and the model editors?

3. How meaningful and complete are the semantic rules of the DSML?

4. How fit is the modeling environment/tool/framework? Does it support, for example, web-based access, collaborative modeling, different stakeholder viewpoints, fully automated code generation (vs. only semi-automated or generating a skeleton), model checking and formal verification, and simulation? How suitable the UI/UX aspects of the environment (e.g., the wizards and menus) are for the users (i.e., practitioners)?

5. Is there any integration with other tools, for example, SMT (Satisfiability Modulo Theories) solvers, or other services (e.g., model repositories)? Does the tool support any particular standards (e.g., safety standards for safety-critical domains)?

Within the scope of this dissertation, the validation and evaluation in Chapter 6 is mainly focused on the DAML-centric criteria, specifically the ML performance metrics Accuracy, Precision, Recall, and F1-Measure for the resulting IoT services, as well as a few criteria of merit that relate to the MDSE tool, mainly the first three items above, which were measured indirectly via the empirical user study (see Section 6.3), in addition to the overall satisfaction level of the users (i.e., practitioners) using the web-based version of the modeling tool provided by this work.

To ensure *Responsive and Responsible Research and Innovation*, a holistic analysis base (e.g., see UI-REF [115]) would have to consider the phenomenological [116] (i.e., pertaining to the perception and experience of phenomena, namely objects and events),
5.1 Framework architecture

teleological [117] (i.e., pertaining to the logic of action-purpose or design objective of phenomena), and epistemological [118] (i.e., associated with the examination of the nature of knowledge, its presuppositions and foundations, and its extent and validity aspects in the context of the framing of the analysis of the domain-in-focus. Accordingly, we studied the entities, objects, and events that were related to our domain of focus, such as the ones that were captured in the proposed meta-model (similar to a domain ontology) in Section 5.1, as well as in the proposed conceptual reference model in Section 5.4. Moreover, we explored the purpose of each object or event, as well as the possible actions and outcomes. Further, we examined the structure of the body of knowledge scientifically, and validated the proposed approach as reported in Chapter 6, which also includes a discussion on the possible threats to validity in Section 6.7.

Finally, to achieve the above-mentioned goals concerning increasing the efficiency of practitioners, in the following, we propose a model-driven software development methodology and a DSML, which extend the methodology and the DSML of the open-source ThingML project [4, 24, 25, 26]. The DSML comprises the following parts that are described below:

1. The abstract syntax that is realized via a MOF-based [32] meta-model, and a grammar that can generate the meta-model.

2. The concrete syntax that is provided in the textual and non-textual forms.

3. The semantics.

Software development methodology

The adopted software development methodology was essentially based on DSM [23], which offers a complete code generation out of software models in a fully automated manner. As explained in Section 2.3, this comes at a price, as the expressiveness of the DSML is intentionally reduced, such that it can only cover a narrow domain. Our domain of focus is independent of any specific vertical IoT application domain, such as healthcare or energy, but specific to the solution domain of smart, heterogeneous, and distributed IoT/CPS services. Therefore, whenever ML-enabled software services that must be deployed on distributed systems with heterogeneous computing platforms that are connected to the Internet/IoT are required, the proposed approach may enhance the productivity of software development and lead to an improved experience for the practitioners.
5 Proposed Approach

Figure 5.1 shows the UML Use case diagram of the proposed approach. Moreover, Figure 5.2 depicts the UML Activity diagram that illustrates the typical workflow in the proposed software development process for smart IoT/CPS services. First, those who are experts in the target IoT/CPS platforms, programming languages, libraries, and communication protocols, whom we refer to as *platform experts*, must add the platform-specific concepts to the meta-model, and extend the model transformations (mode-to-text/code and maybe also model-to-model transformations) to support an automated generation of solutions for the target IoT/CPS platforms. Second, the MDSE practitioners, who are the users of our modeling tool and DSML, model their desired IoT services on a higher layer of abstraction, namely the modeling layer. Further, our model transformations, for example, model-to-text transformations, generate the entire IoT solution for particular IoT platforms out of the software models. The solution will also include trained ML models as far as required, as well as all the scripts that are necessary for building and running the IoT services. Finally, the end user of the resulting IoT services will be able to interact with them, and use them once they are deployed and executed.
Figure 5.1: The UML Use case diagram of the proposed approach.
Figure 5.2: The UML Activity diagram illustrating the usual workflow of using the proposed approach. Figure from [3].
Abstract syntax

The abstract syntax of the proposed DSML is captured in a specific grammar using the Xtext framework [119] that can automatically generate the associated MOF-based [32] meta-model. The UML Class diagram in Figure 5.3 presents part of this meta-model. It is centered around the concept of a thing (in the sense of the IoT) that might be any networked agent in an IoT-connected distributed system. We envision one or more ports for each thing, through which it can communicate with the outside world, namely with other things. Each port in our framework can send and/or receive one or more messages. Each message may possess one or more parameters. Parameters may also belong to thing functions. Moreover, a mandatory component of every thing is its behavioral model in the form of a Finite-State Machine (FSM) or finite-state automata. This is a typical formalism for modeling the behavior of reactive and interactive systems, which are very common in the IoT/CPS world [24].

In addition, each thing can include one or more properties, which are local variables of things. A property assignment expression sets the value of a property. Moreover, a thing might include other (sub-)things, called fragments, and usually has a number of platform annotations that help in realizing the platform-specific semantics. For instance, a platform annotation can specify to which specific type an Integer value must be mapped if Java code is being generated out of the software model instance: short, int, or long.

The main innovation of this doctoral study compared to prior work in the literature (see Chapter 3, mainly ThingML [4, 24, 25, 26]), is the Data Analytics (DA) component. This enhances the capabilities of the behavioral model of things using DAML. We allow each of the things to optionally include one or more DA components that are in charge of carrying out DAML tasks, such as predictions.

In this doctoral dissertation, the focus is primarily on ML methods and statistical inferences rather than simple analytics that would be via some basic statistics or rule-based engines. Currently, we handle supervised, unsupervised, and semi-supervised ML for labeled, unlabeled, and partially labeled data, respectively. Note that almost every class in Figure 5.3, for example, State Machine or Data Analytics, is in practice associated with the Platform Annotation class. However, to prevent the figure from becoming cluttered, those associations are not shown here.
5 Proposed Approach

Figure 5.3: The UML Class diagram showing part of the meta-model of the proposed DSML. Figure from [3].

Finally, part of our extensions of the prior work meta-model is highlighted in Figure 5.4.
5.1 Framework architecture

Figure 5.4: Part of the Ecore meta-model of the proposed DSML. The red stars mark some of our extensions to the Ecore meta-model of the prior work ThingML [4]. Figure from [5].

Concrete syntax

With respect to the concrete syntax, we offer a textual, as well as a non-textual form. The textual concrete syntax is expected to be more suitable for software developers and
5 Proposed Approach

those practitioners with a programming background, whereas the non-textual, i.e., tree-/form-based concrete syntax might be more appropriate for those stakeholders who do not possess such a background. In fact, prior research showed that developers are often more attracted to textual languages than non-textual ones, for example, diagram-based notations [24, 120].

The vocabulary that we deploy for the textual DSML uses the IoT/CPS domain terminology, as well as the SE and ML jargon as much as possible. For instance, in the data analytics part of software model instances, the terminology of the documentations of the deployed ML libraries whose APIs are supported for the automated code generation, are used. Further, we adopt the concrete syntax of the prior work ThingML [4, 24, 25, 26], and extend it with the DAML requirements of our work.

Last but not least, we offer the textual concrete syntax through our textual model editors that are available in both the desktop version and the web-based version of our tool. However, the tree-/form-based concrete syntax that we support is only provided in the desktop version.

Semantics

Semantics are vital for any language, including modeling and programming languages. In fact, without semantics, there is no meaning associated with the structures that might be syntactically, i.e., grammatically, correct or incorrect.

In this doctoral research, one of the primary challenges, due to the inter-disciplinary nature of the research work, concerned handling the diverse notions and semantics in SE, ML, and IoT/CPS. Even the notion of the word semantics itself is often ambiguous and might convey different messages in different communities, and even within one community among different specialists [121]. For instance, when scholars or practitioners in SE, in particular in programming languages and compiler design, express their interpretations from semantics, they often think of models of computation, the processes that a computer executes to run a specific program, and what the relationship between the inputs and the outputs of a program will look like. Moreover, what they mean by semantic analysis (as opposed to syntactic analysis via a parser) is the task of determining properties and checking conditions “that are relevant for the well-formedness of programs according to the rules of the programming language, but that go beyond what can be described by context-free grammars. These conditions can be completely checked on the basis of the program text and are therefore called static semantic properties. This phase is, therefore, called semantic analysis. The dynamic semantics, in contrast, describe the
behavior of programs when they are executed. The attributes static and dynamic are associated with the compile time and the run time of programs, respectively” [122].

However, as Harrel and Rumpe clarified in their work [121], despite many different positions in the SSE community about the semantics of semantics, a precise way of defining any language must include its syntactic elements (which could possibly be an infinite set), the semantic domain of the language, and the semantic mappings from the syntactic elements to the semantic domain. Thus, the semantics of a language must necessarily include a semantic domain, as well as a set of semantic mappings from the syntax of the language to the semantic domain. Yet, a typical misconception in the modeling (sub-)community in SSE, is to confuse semantics with the behavioral model, or the executability of the model [121].

Additionally, semantics is also used beyond the SSE community. For instance, in the web engineering community and the field of IoT, semantic data/information (as in the Semantic Web) refers to machine-readable and machine-understandable data/information. Likewise, a platform is said to be a semantic platform if it can consume and generate such data/information. This is another concrete example for the challenges associated with the inter-disciplinary nature of this work, and in general, for the communication and collaboration of stakeholders in our domain of focus. Yet another specific example can be demonstrated by Figures 5.5 and 5.6, which show a sample PGM and a sample FSM, respectively. The former is an ML model, whereas the latter is a behavioral software model in MDSE. Despite the similarity of the notations, their semantics are different. The PGM is modeled via a directed graph that describes a joint probability distribution of three random variables, called $a$, $b$, and $c$. The vertices/nodes of this graph represent the random variables. However, the edges/arcs/links/arrows show the conditional distributions over these random variables. In fact, the PGM illustrated in Figure 5.5 is equivalent to the joint probability distribution of Equation 5.1 [6]:

$$p(a, b, c) = p(c | a, b)p(b | a)p(a)$$

(5.1)

In contrast, the FSM illustrated in Figure 5.6 models the behavior of a software system that has three states, which are labeled with A, B, and C. If it starts in state A, it might have a transition either to state B or state C, depending on the value of a guard condition, called alpha. If alpha is positive, it will switch to state B, whereas if it is zero or negative, it will switch to state C. From state B, there is only one option that is switching to state C. Once it arrives in state C, it will remain in that state.
5 Proposed Approach

**Figure 5.5:** A sample ML model, namely a Probabilistic Graphical Model (PGM). Inspired by [6].

**Figure 5.6:** A sample behavioral software model in MDSE using a Finite-State Machine (FSM).

In our semantic domain of focus (which is distributed, asynchronously communicating agents), semantics have been studied and formalized, for example, in the works of Broy and Stølen (2001) [123], as well as Rumpe (1996) [124]. Moreover, the work of Gul Agha in 1985 [125] (i.e., the actor model of concurrent computation in distributed systems) provided robust foundations for the semantics of such systems. In this work, our focus is on IoT-connected, intelligent agents that are powered by ML, and deployed on distributed and heterogeneous hardware and software platforms. We consider asynchronous message passing for the communication of these IoT-connected agents, namely things.

The realization of the semantics and the enforcement of the corresponding checks and rules occur at two levels. First, at the meta-model or grammar level, we have a number of model checking and validation rules. For instance, type-checking is applied here. In fact, these are conceptually the context-conditions [121]. Second, at the model
5.2 Formalization

In this section, we first formalize the DAML models with a focus on ML models. Then, we formalize the software models in MDSE. We concentrate on MDSE models for the IoT and CPS.

DAML models

We define an ML model, called $DM$ (the abbreviation of Data Model) used in analytics modeling as follows:
Here, $v$ is an argument that indicates the structure or family type of the ML model $DM$ (e.g., whether it is a DT, a PGM, or an MLP ANN), $P$ is a set that contains all of the parameters of the model $DM$ with their respective values, $\Phi$ indicates the sequence of ML features (i.e., ML attributes and their values) with their respective data types, $H$ is the set of all hyperparameters (e.g., the optimization or learning algorithm $\zeta$ that shall be used to train the model $DM$, the choice of the error/loss/cost/objective function $e$, the batch size $bs$, the number of epochs $ne$, and the learning rate $lr$ if applicable). Finally, $I$ is the set of additional information or meta-data about the model and/or the data. $I$ might include the following items: (i) Whether the model is already trained, if applicable, what the training stage is, and when the time of the last training was; (ii) The paths or URIs/URLs of the dataset(s) used for training, validation, and testing; (iii) Whether any of the data instances has a label (in that case the last item of the sequence of features $\Phi$ indicates the ML class labels and its data type\(^1\)); (iv) If the dataset is sequential (e.g., time series), so that the order of the data instances matter; (v) Whether the training is performed online (i.e., stream processing) or offline (i.e., batch processing). In the former case, the dataset is virtually unbounded, whereas in the latter case, the dataset is bounded.

Analytics modeling involves designing the model $DM$, and then training it, which means using $\zeta$ and other hyperparameters in $H$ to fine-tune the values of the parameters in $P$, so that $DM$ can then make reasonable predictions $Y_{\text{pred}}$ for the previously unobserved data instances, say $X_{\text{new}}$, where the amount of the error/loss $e$ for the prediction of $DM$ given the unobserved inputs (i.e., $\text{pred}(DM, X_{\text{new}})$) remains below a certain threshold $\varepsilon$:

$$DM = (v, P, \Phi, H, I), \quad \text{train}(DM) \rightarrow E[e(\text{pred}(DM, X_{\text{new}}))] < \varepsilon$$  \hspace{1cm} (5.3)

Here, $E$ is the expected value and $e$ is the error/loss, which might be defined according to various metrics, such as the Mean Absolute Error (MAE), also known as the $L1$-norm, for regression:

$$e = \frac{1}{n} \sum_{i=1}^{n} | \hat{y}_i - y_i |$$  \hspace{1cm} (5.4)

\(^1\)We also support array labels/outputs.
In the equation above, \( n \) is the number of data instances, \( \hat{y}_i \) is the predicted numerical label by \( DM \) for the \( i_{th} \) data instance, and \( y_i \) is the actual numerical label of this data instance.

If the data instances are labeled, the task is a *supervised* ML task, thus the prediction implies finding the correct class label for a new, previously unobserved data instance. However, if the data instances do not possess class labels, it is called an *unsupervised* ML task. For instance, in the case of *clustering*, which is an example for unsupervised learning, prediction refers to finding the right cluster for each new data instance. In many applications, only some instances may already have class labels and some or many of them may not have one. This latter case is called *semi-supervised* learning. Further, a supervised ML task with numerical class labels is called *regression*, whereas a supervised ML task with categorical class labels is known as *classification*.

**MDSE models**

We define a Software Model, or more precisely a software architecture model instance, called \( SM \) as shown in Equation 5.5, where \( \Psi \) is the set of structural elements, and \( B \) is the set of behavioral elements.

\[
SM = (\Psi, B) \tag{5.5}
\]

However, since we are interested in domain-specific MDSE with automated full code generation, we augment the said software model formulation with a set of annotations \( A \), and a set of configurations \( C \), thus as defined in Equation 5.6.

\[
SM = (A, \Psi, B, C) \tag{5.6}
\]

**Annotations** The annotations \( A \) often help attach additional semantics to model instances. For example, one may specify which of the available library (API) choices for a certain task, such as ML library/framework, or communication protocol must be used for code generation. This means, if, for example, both Scikit-Learn [29] and Keras [27] offer a certain ML model/algorithm, which is desired, for example, the MLP ANN, one may choose through an annotation whether the APIs of Scikit-Learn [29] or the APIs of Keras [27] must be generated by the model-to-text/code transformation that generates Python code.
5 Proposed Approach

**Structural elements** The structural elements $\Psi$ specify the static aspect of the software System of Interest (SoI). In the IoT/CPS context, $\Psi$ consists of the things $T$, and for each thing $\tau_i \in T$, the ports $P_i$ for communication with other things $\tau_j, j \neq i$, the messages $M_{pi}$ sent/received via each port for message-passing, and the properties or local variables $\Gamma_i$. Each message $m_{pij} \in M_{pi}$ must have a direction (inbound/outbound) and may include one or more parameter(s) $\text{par}(m_{pij}) \in \text{Par}(m_{pij})$. Both the thing properties/variables $\gamma_i \in \Gamma_i$ and the message parameters $\text{par}(m_{pij}) \in \text{Par}(m_{pij})$ are typed, for example, Integer, Float/Double, or String.

**Behavioral elements** The behavioral part $B$ specifies the dynamic aspect of the software SoI. We consider an FSM model, called $FSM_i \equiv B_i$ for the behavior of each of the things $\tau_i \in T$. We define the FSM model as follows:

$$FSM = (\Sigma, S, s_0, \delta, F, \Pi) \quad (5.7)$$

Here, $\Sigma$ is a set of inputs (explained below), which must be finite and non-empty by definition, $S$ is a set of states for the thing $\tau_i \in T$, which is also finite and non-empty, $s_0 \in S$ is an initial state, that must be specified, $\delta : S \times \Sigma \rightarrow S$ is the state-transition function, $F \subseteq S$ is a (possibly empty) set of final states, and $\Pi$ is a set of actions (i.e., operational “semantics”, illustrated below). In this work, we assume the FSM to be deterministic (which means, given an input and a particular state, there will be only one output state for the transition function $\delta$, not a set of states).

Moreover, given the adopted event-driven programming paradigm, the inputs $\sigma_i \in \Sigma_i$ in $FSM_i \equiv B_i$ (i.e., the behavioral model of $\tau_i \in T$) are basically events, for example, the incoming messages sent from other things $\tau_j \in T, j \neq i$ to $\tau_i$. However, the actions $\pi_i \in \Pi$ may be diverse executable operations, such as printing a text in the standard output, storing a message $m_{pij}$ or one of the parameters of a message $\text{par}(m_{pij})$ in a local variable (property) $\gamma_{ij}$ of the thing, or sending a message from $\tau_i$ to another thing $\tau_k \in T, k \neq i$. The new action types that we add to the existing DSML of the prior work ThingML [4, 24, 25, 26], which is extended here, are the following ones for DAML: (i) $DA_{Preprocess}$: This action results in pre-processing the data and making them ready for training the ML model. (ii) $DA_{Train}$: This action leads to performing ML model training. (iii) $DA_{Predict}$: This action enables asking the ML model for prediction. (iv) $DA_{Save}$: This action supports appending the prediction of the ML model to the dataset that was used for training the ML model. Please note that the trained ML
5.2 Formalization

models that result from the DA.Train action will be serialized and stored in any case regardless of the DA.Save action.

Configurations The configurations C include a set of instantiations of the things. This is analogous to object instantiation from classes in the Object-Oriented Programming (OOP) paradigm. Also, it is at this place of the model instance where the desired connections between the ports of the instantiated things are set out. Last but not least, configurations may optionally also include annotations, for example, specifying which model-to-text/code transformations must be used for code generation, and/or which communication protocols must be deployed (e.g., MQTT, HTTP, or CoAP). Hence, we define a configuration $C_i$ for $\tau_i \in T$ as follows:

$$C_i = (A_{C_i}, \Theta, \Xi) \quad (5.8)$$

In Equation 5.8, $A_{C_i}$ is the set of annotations for the configuration, $\Theta$ is the set of instances of things, and $\Xi$ is the set of connectors between the ports of two things. Each instance $\theta \in \Theta$ has an instance name and a type (i.e., the corresponding thing $\tau_i \in T$). Further, a connector $\xi \in \Xi$ has a starting point (which is a thing instance and its port $\theta_a.p_j$), as well as an end point (which is another thing instance and its port $\theta_b.p_k$).

Finally, in the adopted MDSE methodology, the assumption is that the software model instance $SM$ contains a sufficient amount of information (i.e., it is semantically complete) and is syntactically correct (i.e., it is valid) according to the meta-model or the context-free grammar of the modeling language, so that the model-to-text/code transformations can generate the entire implementation of the software for the respective target hardware and software platforms out of the model instance $SM$. Formally, this means:

$$\exists \Delta, \quad is\_valid(SM) \& is\_complete(SM) \rightarrow \Delta(SM) \equiv full\_source\_code \quad (5.9)$$

Here, $\Delta$ is a model-to-text/code transformation, $is\_valid$ returns a Boolean value that is true if and only if the model instance is valid, and $is\_complete$ returns a Boolean value that is true if and only if the model instance is complete. The parser and the model editor that we inherited from the prior work ThingML \cite{4, 24, 25, 26}, and extended in this work concerning the DAML functionalities, support the user of the DSML to design a valid and complete model instance that conforms to the meta-model (grammar) of the
5 Proposed Approach

DSML. The user of the DSML receives the possible error messages, warnings, and hints for each of the lines of the textual model instance if applicable.

5.3 ML-enabled MDSE models for smart IoT services

A software model is defined as shown in Equation 5.6. However, this corresponds to the classic approach to software systems, which tend to exhibit a pre-defined, fixed, or stationary structure and behavior. Many intelligent systems today, in particular in the IoT/CPS domain, pose a degree of dynamicity. Hence, their structure and/or behavior may change, based on the runtime situation, for example, based on the data coming from the surrounding environment, and the predictions/recommendations of the AI (e.g., ML) components of the system over the time. The proposed approach in this dissertation deploys ML to let the software model become adaptable. In other words, we propose considering $\Psi$ and/or $B$ as functions of ML models. We call this ML-enhanced software model a Smart Software Model (SSM), and formalize it in the following way:

$$SSM = (A, f_\Psi(DM_1), f_B(DM_2), C)$$  \hspace{1cm} (5.10)

Here, $DM_1$ and $DM_2$ are two ML models for learning and controlling the dynamicity of the structure and the behavior of the smart software model, respectively. Thus, the structure and the behavior become functions of these ML models.

To set a proper scope for this dissertation, we remove $DM_1$, thus deploy ML only for the behavior of the software model. Therefore, we consider the simplified form below ($DM_2$ is renamed to $DM$):

$$SSM = (A, \Psi, f_B(DM), C)$$  \hspace{1cm} (5.11)

In Equation 5.11, $DM$ is considered to be the ML model as defined in Equation 5.2 (which was $DM = (v, P, \Phi, H, I)$), where $\Phi$ is the sequence of ML features (attributes) of the ML model, $< \phi_1, \phi_2, ... >$, and $\phi_i \in \Gamma$, (which means, the ML features are chosen from the local variables (properties) of the respective thing $\tau$). Note that if the data instances are labeled (i.e., we have a supervised ML task, either classification or regression), the last item of the sequence of ML features $\Phi$ is considered as the class label that must be predicted by the ML model for new data instances. In practice, the local variables (properties) $\gamma_i \in \Gamma$ may be used in order to store the incoming messages and/or their parameters, so that they can be deployed as ML features. Also, they can be
used for storing the prediction of the ML model, for example, to be used in a *message*, or to trigger an *action* by the same or another *thing*.

Finally, we define a platform-independent software model instance for a smart IoT service as follows:

$$\text{PIM} = (\Psi, f_B(DM)) \quad (5.12)$$

$\Psi$ and $f_B(DM)$ represent - as before - the structure and the behavior of software models, respectively. Here, we assume that they are platform-independent. Thus, $DM$, for example, does not specify any particular ML library or framework to be deployed.

Moreover, we define a platform-specific software model, based on the PIM above, for a smart IoT service, as in Equation 5.13 below:

$$\text{PSM} = (\text{PIM}, A, C) \quad (5.13)$$

Here, $A$ and $C$ stand for the platform-specific annotations and configurations, respectively.

### 5.4 AI-enabled architecture frameworks

In Section 2.3, we provided some background information on *architecture frameworks* for software, systems, and enterprises. Here, we argue that existing architecture frameworks are not adequate for modern software, systems, and organizations since they did not consider any AI-related *stakeholders, concerns, viewpoints, views, and model kinds*. However, AI is increasingly becoming a dominant aspect of any system and organization. In fact, AI is not only assisting users, but may also affect the architecture of systems in unprecedented ways. Moreover, we concentrate on DAML, and argue that DAML models deserve to be treated as independent artifacts that are clearly distinguishable from *raw data*. Thus, this must be reflected in architecture frameworks too. In other words, we require new viewpoints, views, and model kinds that enable architecture frameworks to deal with modern AI-/ML-enabled software, systems, and enterprises, in order to address the concerns of those stakeholders who are involved in the AI/ML and cognitive aspects of software, systems, and organizations. Our focus here is more on software and information systems, in particular CPSs and the IoT. This way, we propose a new research direction at the intersection of AI and SSE.
5 Proposed Approach

Below, we first provide three illustrative use case scenarios for IoT/CPS to highlight the above-mentioned need. Then, we present a conceptual reference model that is expected to help in better understanding the DEA/ML aspects of the semantic domain of focus. Based on this model, we propose new stakeholders with AI, and in particular, DEA/ML-related concerns, as well as new viewpoints and views to frame and address their concerns, respectively. Finally, we propose using existing formalisms, notations, and model kinds to present these views.

Airport full-body scanner The headquarters of a manufacturer of advanced full-body scanners is in the European Union (EU). Their devices work based on the ML models that are trained using a huge amount of data. They wanted to install a new device at an airport in Asia. However, the data privacy regulations did not allow them to export the data, that they had acquired inside of the EU, to non-EU countries. Hence, they decided to deploy their trained ML models (instead of the raw data that had previously been used to train those models) to their new location in Asia. Note that even the data that they still have within the EU are not the data of the complete dataset that has been used to train their models in the past. For instance, in certain countries, such as Germany, privacy regulations do not allow the video surveillance data from many public places to be stored permanently or even at all. This is also a major issue for data analytics in some other domains, such as the healthcare industry. Therefore, the data instances that are used to train the models may need to be deleted afterwards. This simple, but practical example shows the importance of the new type of artifacts (namely, the DAML models), and reveals how the ownership conditions and the import/export constraints of the raw data may differ from the ones of the DAML models. Thus, these artifacts deserve to be treated in such a manner, as they possess a distinct identity from raw data. Yet, the existing architecture frameworks (see Section 2.3) totally missed the viewpoints, views, and model kinds with respect to these vital artifacts.

Smart microgrid Smart grids use Information and Communication Technologies (ICT) to enhance the performance, efficiency, reliability, and availability of the power grids. The smart microgrid used here consists of a network with one controller, hundreds of consumers, and tens of prosumers. The latter are participants in the energy network who may act as both a producer and a consumer of electrical energy, depending on the situation. The goal is to optimize the overall demand and supply of the energy in the network. To this aim, the controller tries to minimize excessive generation by
prosumers, whenever the demand is expected to be low, thus avoiding storage in the batteries as much as possible (since that is not efficient). Simultaneously, it must ensure that there will not be a shortage of energy in the network. Note that any lack of energy at peak times would mean importing (i.e., buying) energy at a very high price. Since energy prices in modern grids need to be demand-responsive, and as such dynamic, with the spot-prices often being set at the free energy stock markets, optimizing the export/import times can make a significant impact on the stakeholders’ gains. The controller takes care of such matters in the smart microgrid. In fact, it deploys a number of ML models to predict the demand and supply of the electrical energy in the near future, so that it can better plan and give proper feedback to the participants. Note that essentially the DAML models used by the controller are defining its runtime behavior. Hence, any architecture description of the system that does not take those models into account cannot claim to be a comprehensive one. At least one viewpoint and one view must be dedicated to such an important aspect of these smart data-driven systems. However, current architecture frameworks are simply unable to offer this since they lack the required concepts and principles. The ongoing trend towards the data-driven approaches versus the pure model-based engineering approaches, which follow Physics laws instead of learning from the existing data, has been observed in various engineering domains (e.g., see [128]). Therefore, it is important to make architecture frameworks ready for this paradigm shift.

**E-commerce platform** An e-commerce company with a global network of retail and logistics that is integrated by a mass-scale Internet-connected distributed system, which orchestrates the overall network, is no longer just an online platform, but rather a smart CPS. They utilize ML models and algorithms to optimize their operations and maximize their profit, in a business that has a large revenue, but a relatively small profit margin, compared to the online platforms from other vertical domains, for example, popular social networks and search engines. Therefore, it is no surprise that they have many data analytics needs, which are in part real-time, to optimize every operation as far as possible. In fact, the recommendations of the smart system to the users (buyers and sellers), the business owners/managers, and their partners could potentially have an impact not only on the behavior of the users, but even on the system behavior, architecture, and topology. Without enabling DAML in architecture frameworks, requirements traceability with respect to the learning, self-optimization, and self-adaptation capabilities of the smart CPS would also be missing. Moreover, smart CPS are often highly dynamic
systems of systems. Hence, in modeling such complex systems, it can be seen that the borders between the design-time and the runtime are gradually fading out (e.g., see [91]). This means, the models of those systems are going to become more dynamic. In fact, there will be much more focus on the DAML aspects since those are eventually going to be the majority of what is designed before the execution. Most of the other parts must be determined dynamically at runtime, based on the predictions of the DAML components using the incoming data (i.e., the interactions with the environment, which includes other systems, for example, many sensors and actuators, as well as the human users).

5 Proposed Approach

Conceptual reference model

Building smart data-driven systems involves Data Analytics (DA)/Data Science (DS), and Data Engineering (DE). The former, also known as analytics modeling [9] is performed by data scientists, including ML engineers, whereas the latter, also known as analytics operations [9] is the job of data engineers (not to be confused with database engineers/database designers).

Moreover, DAML model producers, which can be software tools or applications, create and train DAML models, for example, deep ANNs. Later, various DAML model consumers (i.e., scoring engines) can deploy and query these models for predictions, based on new incoming data instances that have not previously been observed by the DAML models. This is often called batch processing or offline learning. However, many smart CPS in the IoT era require stream processing or online learning, where the border between the training phase and the production/deployment phase vanishes. This way, upon receiving every new data instance, the ML model is incrementally improved. Nevertheless, online learning (better called incremental learning) algorithms can also be used in offline learning applications (i.e., batch processing), whenever the size of the dataset is so huge that it does not fit into the main memory. This is called out-of-core learning [11].

In this doctoral dissertation, we take the first step towards enriching architecture frameworks with AI/ML-related concepts by proposing a novel conceptual reference model for DEA/ML. Let us clarify a number of different, but closely related terms, namely reference model, reference architecture, and architecture framework. A reference model is “an abstract framework for understanding significant relationships among the entities of some environment, and for the development of consistent standards or specifications supporting that environment. A reference model is not directly tied to any
standards, technologies or other concrete implementation details, but it does seek to provide common semantics that can be used unambiguously across and between different implementations” [129]. However, a reference architecture (e.g., see AUTOSAR [130] for the automotive domain) is a reference model augmented with mappings that relate it to the software elements that implement the functionalities promised by the reference model [131]. Moreover, an architecture framework (e.g., see the work of Broy et al. [132, 133] for the automotive domain) is a set of “conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders” [52]. An architecture framework provides a conceptual and overall framework for various purposes, including specifying reference architectures (i.e., more concrete architecture templates for particular groups of systems in a domain).

Figures 5.7 and 5.8 illustrate our proposed reference model for DEA/ML.

**Framing and addressing AI/ML-related stakeholder concerns**

Based on the presented reference model in Figures 5.7 and 5.8, we identified the following stakeholders with AI- (and in particular ML-) related concerns: (i) data scientist (including ML engineer and DA researcher), (ii) data engineer. In addition, we assumed that the following groups might also have a stake in the AI/ML aspects: (iii) end-users, (iv) business stakeholders, (v) database engineers/designers, (vi) software engineers, (vii) software architects, (viii) system engineers, (ix) network experts, and (x) security experts. However, the expert interviews and the online survey that were conducted to validate this (and will be explained in Section 6.6) shed light on additional stakeholders, namely (xi) safety and regulatory compliance engineers, (xii) data protection (privacy) officers, as well as (xiii) ethics committees/boards.

After identifying the stakeholders as listed above, we started categorizing and framing their concerns into viewpoints. Also, we explored possibilities to either devise new views and new model kinds or use existing formalisms and notations in order to address their concerns. However, after a number of iterations, we decided to follow a pure empirical approach and ask the domain experts to express their wishes for such views and model kinds. We used the opportunity of the interviews and the survey to not only validate our assumptions, but also gain new insights. Table 5.1 shows the new viewpoints that we propose in order to frame the AI/ML/DEA-related concerns of stakeholders. Moreover, Table 5.2 illustrates the model kinds and view notations that we propose for the said viewpoints. In fact, we propose two new architecture viewpoints, namely Analytics.
5 Proposed Approach

modeling and Analytics operations, as well as a number of new architecture views, using existing model kinds, notations, and formalisms, for addressing the AI/ML/DEA-related concerns.

Please note that some existing architecture frameworks addressed data, but that was rather concerning raw data, not DEA/ML. For instance, a previous version of TOGAF (v8.1.1, 2006) \[55\] considered a dataflow view for the viewpoint of database engineers who are “concerned with the storage, retrieval, processing, archiving and security of data”, which in general means “assuring ubiquitous access to high quality data”. However, it did not take any data analytics into account. Similarly, the latest version of TOGAF (v9.2, 2018) \[56\] promoted a set of data-related views, such as the conceptual data diagram (intended for the viewpoint of business stakeholders), the logical data diagram (intended for the viewpoint of database designers and application developers), as well as the data dissemination diagram, the data security diagram, the data migration diagram, and the data life-cycle diagram. However, it did not consider any AI/ML/DEA aspect.

In this chapter, we elaborated on the proposed approach. First, we presented the overall architecture, including the criteria of merit. Then, we formalized DAML and MDSE models, and proposed enhancing MDSE models with ML. Finally, we proposed augmenting architecture frameworks of software and information systems with the DAML architectural viewpoints and views.
### Table 5.1: The proposed architectural viewpoints concerning DEA/ML

<table>
<thead>
<tr>
<th>-</th>
<th>Viewpoints</th>
<th>Stakeholders</th>
<th>Concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>New: Analytics modeling</td>
<td>data scientist</td>
<td>creating capable DA models</td>
</tr>
<tr>
<td>2</td>
<td>New: Analytics operations</td>
<td>data engineer</td>
<td>efficient large-scale data processing</td>
</tr>
<tr>
<td>3</td>
<td>End-user</td>
<td>end-user</td>
<td>usability / desirability</td>
</tr>
<tr>
<td>4</td>
<td>Business</td>
<td>business stakeholder</td>
<td>business viability</td>
</tr>
<tr>
<td>5</td>
<td>New: Regulation and ethics</td>
<td>safety and regulatory compliance engineers, data protection (privacy) officers, quality assurance engineers, and ethics committees / boards</td>
<td>social responsibility, privacy, explainability, fairness, and inclusiveness</td>
</tr>
<tr>
<td>6</td>
<td>Database / Logical data</td>
<td>database engineer / designer</td>
<td>database performance, data model</td>
</tr>
<tr>
<td>7</td>
<td>Development</td>
<td>software engineer / architect</td>
<td>software performance</td>
</tr>
<tr>
<td>8</td>
<td>Physical</td>
<td>system engineer and network expert</td>
<td>performance, availability, and quality of service</td>
</tr>
<tr>
<td>9</td>
<td>Adapted: Security</td>
<td>security experts, data scientists, data engineers</td>
<td>DAML model security, for example, adversarial robustness (e.g., see [134])</td>
</tr>
</tbody>
</table>
Figure 5.7: The proposed reference model for DEA using the UML class diagram. Disciplines and individuals are marked in red and yellow, respectively. PMML [7, 8] and PFA [9, 10] are standards of the Data Mining Group (DMG).
Figure 5.8: The proposed reference model for ML using the UML class diagram.
## Table 5.2: The proposed architectural view notations (model kinds) concerning DEA/ML

<table>
<thead>
<tr>
<th>Viewpoints</th>
<th>View notations (model kinds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Analytics modeling</td>
<td>(i) mathematical notation, (ii) charts / diagrams / plots, (iii) Data-Flow Graphs (DFGs) [28], (iv) PGMs [6], (v) dataflow diagrams</td>
</tr>
<tr>
<td>2  Analytics operations</td>
<td>(i) dataflow diagrams / UML activity diagrams, (ii) UML class diagrams, (iii) DFGs [28], (iv) Entity-Relationship (ER) diagrams, (v) mathematical notation</td>
</tr>
<tr>
<td>3  End-user</td>
<td>(i) text documents, (ii) charts / diagrams / plots, (iii) tables / matrices, (iv) dataflow diagrams / UML activity diagrams, (v) UML use case diagrams</td>
</tr>
<tr>
<td>4  Business</td>
<td>(i) charts / diagrams / plots, (ii) text documents, (iii) tables / matrices, (iv) UML use case diagrams</td>
</tr>
<tr>
<td>5  Regulation and ethics</td>
<td>(i) text documents, (ii) dataflow diagrams / UML activity diagrams, (iii) ER diagrams, (iv) DFGs [28] augmented with physical / deployment info, (v) tables / matrices, (vi) UML deployment diagrams</td>
</tr>
<tr>
<td>6  Database / Logical data</td>
<td>(i) ER diagrams, (ii) UML class diagrams, (iii) dataflow diagrams / UML activity diagrams, (iv) UML use case diagrams, (v) tables / matrices</td>
</tr>
<tr>
<td>7  Development</td>
<td>(i) UML class diagrams, (ii) dataflow diagrams / UML activity diagrams, (iii) UML use case diagrams, (iv) ER diagrams</td>
</tr>
<tr>
<td>8  Physical</td>
<td>(i) UML deployment diagrams, (ii) dataflow diagrams / UML activity diagrams, (iii) DFGs [28] augmented with physical / deployment info, (iv) UML class diagrams</td>
</tr>
<tr>
<td>9  Security</td>
<td>(i) dataflow diagrams / UML activity diagrams, (ii) UML deployment diagrams, (iii) DFGs [28] augmented with physical / deployment info, (iv) ER diagrams, (v) mathematical notation, (vi) UML class diagrams</td>
</tr>
</tbody>
</table>
6 Implementation, Validation, and Evaluation

In this chapter, we report on the validation and evaluation of the proposed approach. We implemented the prototype of the proposed approach and conducted our experiments with this research prototype. This chapter is mainly based on our prior publication [3] and [15]. Further, Sections 6.4 and 6.5 are specifically based on [18] and [35], respectively.

6.1 Validation method

In Section 4.2, we described the deployed research methods. Here, we elaborate on the validation method for the RQs (see Section 4.1). To validate the research questions, we implemented the open-source prototype, and showed the feasibility and efficiency of the proposed approach using two case studies from the smart energy systems, and predictive maintenance application domains. We used implementation, simulation, and testing to validate the proposed approach following empirical and experimental research, based on two case studies, a user study with four participants, a number of one-on-one expert interviews, as well as a self-administered online survey. Below, we explain our data collection method, as well as the specific use case scenarios.

Data collection

We used the open data provided by four public datasets. First, for the smart energy systems case study (which is described below), we deployed three reference datasets for energy disaggregation, called REDD [135], REFIT [136, 137], and UK-DALE [138]. The first one was from the US, whereas the last two were from the UK.

For the experiment concerning the validation of RQ 1, we used the data from House/Building 1 of the REDD [135] dataset, which was a single-family dwelling with two inhabitants (a couple). Various sensors had recorded different conditions in their environment over a period of 21 months starting from October 2013. The parameters of
interest here were the individual loads (i.e., active power measured in Watts) of the following electrical appliances, as well as the aggregate load (i.e., the total power consumption of the entire house). The samples were recorded at a frequency of 0.125 Hz (i.e., once every 8 seconds). They included the following appliance loads: (i) fridge, (ii) freezer-1, (iii) freezer-2, (iv) washing machine, (v) dishwasher, (vi) computer, (vii) television site, (viii) electric heater, and (ix) washer dryer.

Moreover, regarding the *predictive maintenance* case study (which is introduced in the following), we used the public dataset provided by ZeMA gGmbH in Germany, which is available on Kaggle [11, 139].

**Use case scenarios**

**Smart energy systems** We studied a smart grid that had a number of smart homes in its network. As mentioned in Section 5.4, it is vital for smart (micro)grids to become capable of making accurate predictions about the possible load of the energy network at different points in time in the near future in various geographical locations. Therefore, we deployed ML methods to enable this prediction. However, the ML methods required observing data to *train* ML models. Hence, we needed a mechanism to measure the loads (i.e., active powers in Watt) of electrical appliances in different buildings that were connected to the grid at different times. Moreover, another reason for needing this information was that some electricity providers, especially those who possessed smart grids might offer certain discounts if the electrical appliances with higher consumption levels were avoided during peak hours. However, to check whether the agreements had been respected by the consumers, they needed access to the energy consumption levels of these appliances on the consumer side at certain points in time.

Generally, there are two ways to acquire these data: either they must attach *smart meters* to the appliances and monitor the energy consumption in an *intrusive* manner, or they can follow a *non-intrusive* approach, thus use virtual sensors, for example, based on ML methods to perform *energy disaggregation* using the aggregate power signal that is available for every household through its counter. The former setup (i.e., intrusive), which is depicted in Figure 6.1, was taken into consideration in validating RQ 1 (see Section 6.3), whereas the latter setup (i.e., non-intrusive) was adopted in validating RQ 3 (see Section 6.5).

Energy disaggregation, also known as *Non-Intrusive (Appliance) Load Monitoring (NIALM/NILM)*, is a well known problem in the field of smart energy systems. Given an *aggregate* signal of the electrical energy consumption of a household over a period of
6.1 Validation method

Figure 6.1: The use case scenario concerning load monitoring using smart meters (i.e., the intrusive way) for validating RQ 1.

time, we aim to *disaggregate* the signal into a number of individual loads. This is called Non-Intrusive (Appliance) Load Monitoring (NIALM/NILM) or energy disaggregation, and is similar to the single-channel source separation problem in Physics and Signal Processing, but generally more challenging.

**Predictive maintenance**  *Predictive maintenance*, also known as *condition-based monitoring* is an alternative to classic approaches to maintenance. Instead of waiting for a system failure and then planning maintenance, or conducting periodic visual inspections or non-destructive testing, one could monitor certain qualities and conditions of a system using sensor data, and use DAML to predict system failures before they actually occur or damage the system or its environment. This way, the owner/operator can better plan for the maintenance of the system, for example, by substituting or repairing a broken part. Thus, one can reduce the system maintenance and operations costs, and increase the user satisfaction and convenience. Also, this can prevent some avoidable risks and hazards.

In this doctoral research, we were interested in the condition-based monitoring of a hydraulic system for our case study in this application domain, namely predictive maintenance, which is a typical use case domain for the IoT and CPS. We monitored
the data of a number of sensors, such as vibration sensors, in order to predict the possible leakage of the main pump as shown in Figure 6.2.

![Figure 6.2: The use case scenario concerning the predictive maintenance of a hydraulic system for validating RQ 2. Figure inspired by [11].](image)

### 6.2 Open-source prototype

We implemented the proposed approach ML-Quadrat [3, 5, 14, 15, 16, 17, 18] and released our open-source prototype, which has the same name as the proposed approach, namely *ML-Quadrat*, under the Apache 2.0 software license on Github [17]. This was built on top of the Eclipse Modeling Framework (EMF) [86], using the Xtext framework [119], and by extending the open-source ThingML project [4, 24, 25, 26]. The UML Component diagram, which illustrates the logical view of a number of key functional software components of our prototype, is presented in Figure 6.3. Most of them were
also present in the prior work ThingML [4, 24, 25, 26]. However, we adapted and extended them for our purpose of enabling DAML at the modeling level. According to the legend of the diagram, the unchanged, adapted/extended, and generated components are depicted in blue, red, and green, respectively. Here, we skipped the rest of the code generators that were inherited from the prior work, such as the C/C++ code generators. Likewise, the figure does not exhibit the additional model-to-text/code transformation that we developed in order to support the TensorFlow Lite [140] and the TensorFlow Lite for Microcontrollers [141] libraries (the latter to enable TinyML support). This is explained in Section 6.4.

![Figure 6.3: The UML Component diagram illustrating part of the logical architecture view of our prototype.](image)

In addition to the desktop version of the prototype that is available as open-source software, we offer a web-based version of the prototype that is not included in the open-source distribution, but is available upon request for the reproducibility of the results of the empirical evaluation. The web-based interface helped us conduct the experiments with the external evaluators as they did not need to install any software on their side, but simply used the web application in their web browsers.

In the following, we first illustrate the implementation of the abstract syntax and the concrete syntax of the proposed DSML. Then, we explain the model transformations...
6 Implementation, Validation, and Evaluation

(e.g., code generators) that could realize the semantics and generate the full implementation of the target IoT services out of the software model instances. Further, we elaborate on the DAML matters, specifically on the ML methods that have been supported out-of-the-box in the DSML, as well as how to deploy them. However, we also enabled the practitioners (e.g., software developers and data scientists) who used the proposed approach, to deploy any arbitrary ML method in the so-called black box ML mode. This is also explained below. Finally, we demonstrate a sample IoT service that is a basic client-server interaction (i.e., ping-pong) to highlight the advantages of our work compared to the prior work ThingML [4, 24, 25, 26].

Abstract syntax of the DSML

The abstract syntax of the proposed DSML was defined in its grammar that was implemented with the Xtext framework [119]. This is available in the source code repository of the open-source project on Github [17]. The Ecore meta-model of the DSML could be generated automatically out of this Xtext grammar. We illustrated part of this meta-model in Figure 5.3 (see Section 5.1) using the UML Class diagram.

The Data Analytics class shown in Figure 5.3 realized DM in Equation 5.2 (see Section 5.2). This was the main innovative aspect of our extension of the meta-model from the prior work. We describe this in more detail below, and illustrate a sample IoT service model that shows how to use this in practice to create smart IoT/CPS services using DAML.

Moreover, the imperative action language of ThingML [4] that supported event-driven programming on the state machines, which realized the behavioral models (i.e., B in Section 5.2) of things, was extended. Using this action language, one might specify which actions (see II in Section 5.2) must be taken upon the occurrence of a particular event, such as upon the receipt of a certain message on a specific port of a thing. For example, a state transition might happen due to this event. Various types of actions, such as conditional actions, loop actions, and print actions were possible with the prior work. However, we introduced the new action types, which were named in Section 5.2, in order to enable the DAML functionalities. They included creating and running the data pre-processing pipeline (i.e., DA_Preprocess), conducting ML model training (i.e., DA_Train), making predictions using the trained ML models (i.e., DA_Predict), and optionally saving the predictions in the dataset (i.e., DA_Save)\(^1\).

\(^1\)Obviously, DA_Preprocess and DA_Train are skipped in the case of a pre-trained ML model in the black box ML mode.
Additionally, *Thing*, *Thing Fragment*, *Platform Annotation*, *Port*, *Message*, *Parameter*, and *Property* (see Figure 5.3) realized $\tau \in T$, $a \in A$, $p \in P$, $m \in M$, $\text{par}(m) \in \text{Par}(m)$, and $\gamma \in \Gamma$, respectively, that were mentioned in Section 5.2. Last but not least, other elements, such as *Function* and *Property Assign*, as well as those which were not shown in Figure 5.3, fell outside of the scope of this work, thus could be found in the related work, for example, [4, 24, 25, 26].

**Concrete syntax and model editors**

We provided three model editors. First, we made a model editor based on Xtext [119] available in the EMF-based [86] tool (i.e., the desktop version). This possessed a textual concrete syntax, as well as the syntax highlighting and auto-complete features, and could give a number of hints and tips to help the practitioner (i.e., the user of the modeling tool) in designing a valid and complete model instance, out of which code generation for a working IoT service with the desired functionality was feasible. Figure 6.4 shows this model editor.

Second, we offered a tree-based (form-based) model editor through the EMF. This was automatically generated in the EMF out of the Ecore meta-model of the DSML, which had itself been generated automatically out of the Xtext grammar of the DSML. The tree-based model editor is demonstrated in Figure 6.5. While the textual version might be more suitable for developers, the tree-based editor might suite domain experts of the target IoT domains without software development skills well, so that they could modify certain properties of the software model instances, for example, for the maintenance, upon possible future changes in the requirements.

Last but not least, we developed a web-based prototype using the Java (J2EE) Servlets technology and the Xtext web integration. This web application offered a textual model editor with the auto-complete feature and some basic syntax highlighting. This is depicted in Figure 6.6.

**Semantics and model transformations**

Part of the semantics of the DSML was included in the model-to-code/text transformations (i.e., $\Delta$ in Section 5.2), also known as code generators or *compilers*. In addition, another part of the semantics was integrated into the grammar or meta-model, to enable type-checking and enforcing certain constraints at the design-time through the model editors (i.e., before executing the code generators). The latter are better known as
context conditions [121]. Furthermore, we supported a number of annotations (i.e., \( A \) in Section 5.2), for example, concerning the datatype mappings on specific target platforms, the choice of specific libraries for DAML, particular communication protocols, and model-to-code/text transformations on the modeling layer.

The proposed approach supported code generation in Python and Java, as well as C code for particular microcontrollers (see Section 6.4). The Python code and the C code (the latter in the case of TinyML devices) were responsible for the DAML functionalities of the target IoT services. The Python code supported the APIs of Scikit-Learn [29] and Keras [27] with the TensorFlow [28] backend, as well as TensorFlow Lite [140]. However, the C code could be used with the TensorFlow Lite for Microcontrollers library [141] for TinyML devices. The model-to-code transformations were implemented in Java and Xtend (which is a modern variant of Java). They can be found in our Github repository [17].

**Supported ML methods and techniques**

The proposed approach enabled each **thing** to possess one or more components for DAML. Thus, it supported not only analytics in the cloud, but also edge analytics. Unlike the behavioral component of a thing, which was a state machine (statechart), the DAML component, called Data Analytics (DA) was not mandatory. To exhibit DAML capabilities, a thing had to include a data analytics section in its model. This component that realized \( DM \) in Equation 5.2, might affect the behavior of the thing, modeled via the corresponding state machine. As mentioned before, this corresponded to \( f_B(DM) \) in Equation 5.11. In other words, the behavior of the thing had become a function of the DAML model. Hence, if a thing had a data analytics part, this part had to emerge before the state machine section in the textual model instance, so that the actions specified in the state machine could use and refer to the data analytics component.

Below, we list and briefly explain the possible parameters and options in the said data analytics section of the ML-enhanced software model instances that conform to the meta-model (grammar) of the proposed DSML (see Figures 6.8 and 6.9):

1. **Data_analytics**: This parameter/option determines the name of the DAML component (e.g., da_1).

2. **Dalib**: The optional @dalib annotation specifies the name of the library or framework which must be used for DAML. If this is absent, or it is set to auto, or the

---

2See https://github.com/arminmoin/ML-Quadrat/tree/master/ML2/ compilers/
desired ML method is not implemented in the selected library, the tool will try
to automatically select the best choice in the AutoML mode (if AutoML is set to
ON).

3. **Labels**: This is a binary parameter/option. If it is set to ON, it implies that the
ML task is *supervised*. Hence, the last item on the list of *features* (see below) will
be considered as the *label*. If the data type of that item, defined as the data type
of the corresponding

*property* (local variable) of the *thing* is numeric (e.g., Integer or Float/Double),
then the ML task is a *regression* task. Otherwise, it is a *classification* task. Fur-
thermore, if this parameter/option is set to OFF, then the task is *unsupervised*
(e.g., *clustering*). This parameter/option also partially realizes $I$ as referred to in
Section 5.2.

4. **Features**: This is a list of the *properties* (local variables) of the *thing* which must
be considered as the ML *features* (attributes). The local variables might include
the *messages* or message *parameters* that must be received from other *things*. As
stated above, these are all considered as ML *features* only if *labels* is set to OFF.
In the case that *labels* is set to ON, then the last item is not considered as an
ML *feature*, but rather as the *label* (i.e., the class label for classification, or the
target value for regression). This parameter/option realizes $\Phi$ as introduced in
Section 5.2. Simultaneously, the *features* are *properties* (i.e., local variables) of the
responding *thing*, thus also partially realizing $\gamma \in \Gamma$ in Section 5.2.

5. **Prediction results**: This parameter/option determines the *property* (i.e., local
variable) of the *thing* in which the prediction result (i.e., the output of the ML
model prediction) must be stored. Note that the *properties* were denoted by $\gamma \in \Gamma$
in Section 5.2. The value of this *property* can be then later used in the *actions*
of the state machine, in order to let the ML model affect the behavior of the *thing*.

6. **Dataset**: The path of the dataset on the file system that must be used for training
the ML model. This has to be a CSV (Comma-Separated Values) file without a
header line.

7. **AutoML**: This is a binary parameter/option indicating whether the AutoML
mode must be used. If set to ON, a number of AutoML functionalities will be
supported that can assist the practitioner, especially the novice users in the DAML
field. By default, this is set to OFF.
8. **Sequential**: This is a Boolean parameter/option that indicates whether the input data are sequential (e.g., time series), such that the order of data instances matters. In this case, shuffling and *cross-validation* must be avoided. This parameter/option partially realizes $I$ as referred to in Section 5.2.

9. **Timestamps**: This binary parameter/option states if the data instances have timestamps or not. If this is set to ON, it has at least two implications. First, if new *messages* or message *parameters* need to be appended to the dataset (using the DA_Save action), timestamps will be automatically added as well. Second, the DAML method will be informed that the first column in the dataset, which is the CSV file, must be considered as the timestamp. The expected format is *dd-mm-yyyy HH:MM:SS* (e.g., *17-03-2021 22:49:06* for *March 17, 2021 at 10:49:06 pm*). Obviously, if the timestamps parameter/option is set to ON, it is very likely that we are dealing with time series (and consequently sequential) data. Therefore, if the sequential parameter/option is not specified, the AutoML functionality of the tool, if it is set to ON, will automatically set the *sequential* parameter/option to *True*. However, if the user explicitly states that *sequential* is *False*, then the decision will not be overridden. The *timestamps* parameter/option also partially realizes *I* as referred to in Section 5.2.

10. **Preprocess_feature_scaling**: This parameter/option specifies the *feature scaling* technique that must be used in the data preparation (pre-processing) pipeline. If it is not present, in the case that AutoML is set to ON, then the best choice of scaling for the respective ML model/algorith method (see below) will be selected. For instance, for the higher performance of Artificial Neural Networks (ANNs), having numerical data that possess a relatively similar scale is an extremely important factor. Thus, for example, *standardization* (also known as the *Z-Score normalization*) is automatically activated in the AutoML mode. This parameter/option partially realizes $H$ as set out in Section 5.2.

11. **ML Model/Algorithm**: Here, one can specify the particular ML method, including the ML model architecture (family) that must be deployed (e.g., Multi-Layer Perceptron (MLP) ANN, Decision Tree, etc.). Additionally, the hyperparameters (e.g., the choice of the error/loss function $e$), the learning/optimization algorithm (Q), the learning rate (lr), etc. might be given in parenthesis. Each family of ML models may have a different set of possible hyperparameters. The auto-complete feature (usually activated by pressing the *Control* and *Space* keys together for the
6.2 Open-source prototype
textual model editors) helps in finding the possible options. Further, the doc-
umentation of the prototype, as well as the API documentations of the target
frameworks and libraries (e.g., Scikit-Learn [29]) must be studied. Also, a num-
ber of exception handling and logging mechanisms are available to support the
user of the tool. This parameter/option realizes \( v \), as well as \( H \) in Section 5.2.
The parameters of the ML model (i.e., \( P \) in Section 5.2) are controlled by the
hyperparameters \( (H) \) during the learning process.

12. Training Results: This is the path of the text file in which the log of ML
model trainings will be stored. The log includes information about the time of
each training and the chosen ML model/algorithm/method. This parameter also
partially realizes \( I \) mentioned in Section 5.2.

Currently, the following ML models and algorithms are supported for supervised ML
(i.e., for labeled data) out-of-the-box: (i) Linear Regression, (ii) Logistic Regression
for linear classification, (iii) Naïve Bayes (the Gaussian, Multinomial, Complement,
Bernoulli, and Categorical variants), (iv) Decision Tree (both Regressor and Classi-
fier), (v) Random Forest (both Regressor and Classifier), (vi) the Multilayer Perceptron
(MLP) ANN. The APIs of Scikit-Learn [29] are used for the items (i) to (v). However,
for the MLP ANN (i.e., (vi)) both Scikit-Learn [29] and Keras [27] are supported. By
default Keras [27] will be used for this family of ML models. However, the user may
explicitly set the library for DAML to Scikit-Learn [29] to override this recommended
setting. This is possible through the annotation \( \text{dalib} \) in the \text{data-analytics} section of the
model instance. Moreover, a number of other techniques, for example, for data prepara-
tion, specifically standardization or normalization of the numerical features using various
methods are provided.

Moreover, the unsupervised ML methods that are also pre-defined, thus supported
out-of-the-box are as follows: (i) K-Means, (ii) Mini-Batch K-Means, (iii) DB-SCAN,
(iv) Spectral Clustering and (v) Gaussian Mixture Model. The APIs of the Scikit-Learn
library are used for enabling them. Further, for semi-supervised ML, we enable
(i) Self-Training, (ii) Label Propagation, and (iii) Label Spreading with the APIs of
Scikit-Learn [29].

If the desired ML model, algorithm, or technique is not pre-defined, one may either
extend the open-source prototype (see the online documentation on Github [17]), or use
the so-called black box ML mode (also known as the hybrid/mixed MDSE/non-MDSE
mode) as described below. In the latter case, one can bring any arbitrary pre-trained ML model and *connect* it to the MDSE model.

**The black box ML (i.e., hybrid/mixed MDSE/non-MDSE) mode**

Suppose that one does not want to use an existing ML method that is already available in our prototype, or one has already an existing, pre-trained ML model that they want to deploy. In this case, the black box ML mode, also called the hybrid or mixed MDSE/Non-MDSE mode is preferred. The drawback here is that the software model will not include any information about the deployed ML method. Therefore, the ML model seems to the software model as a black box. However, the advantage is that the user of the DSML will achieve a much higher degree of flexibility concerning ML. Hence, they may, in principle, introduce any pre-trained ML model with any arbitrary architecture and trained with any learning algorithm, and *connect* or *plug* it into the software model.

This can be done by using a parameter, called `blackbox_ml` and setting its Boolean value to true. In this case, using the `model_algorithm` and the `training_results` parameters will not be allowed in the data analytics section of the model instance as no training is required by the AI-enhanced MDSE model. The pre-trained ML model has to be stored in a separate directory. The path of this directory must be given through a parameter, called `blackbox_ml_model` in the data analytics section of the model instance. The pre-trained ML model might have been trained with or without the proposed approach. Moreover, the ML method that is imported from the corresponding DAML library must be specified using a parameter, called `blackbox_import_algorithm`.

**Modeling a sample IoT service**

In the following, we illustrate a basic example from the ThingML project [4], and elaborate on the shortcomings of the prior work by showing our extended (smart) version of this example.

**Ping-Pong** This example originally came from the ThingML project [4]. In a distributed system, there exist two nodes (each modeled as one *thing*) that are connected to the IoT: (i) the ping client, and (ii) the pong server. The *things* are involved in a basic client-server interaction where the server simply waits for incoming ping messages from the client. As soon as a ping message arrives, the server responds with a pong message.
6.3 Validating RQ 1

**Smart Ping-Pong**  We argue that in a real-world scenario with an enormous number of clients, which may send a ping message to the server, the example above can be enhanced via ML, in order to prevent the so-called DistributedDenial of Service (DDoS) attacks. Hence, we introduce a new *thing* that is responsible for DAML, in order to predict if a client is prone to be an attacker or not. Upon receiving a ping message, the server consults this new *thing*, which might even be a *thing fragment* for the server, to see if the ping message should be responded to with a pong, or it would be safer to ignore the request, and perhaps even put the client in a blacklist for a certain period of time. Note that this was not possible using the prior work, whereas our work supports DAML at the modeling level. Using the proposed DSML, one may enhance the software model instances to become capable of DAML.

Figure 6.7 depicts the state machines that model the behaviors of the ping client, the pong server, and the data analytics server.

Below, we demonstrate part of the model instance for the smart ping-pong example (see Figures 6.8 and 6.9). The full model instance resides in our Github repository.3

Finally, the user documentation that is available in our Github repository [17] provides further details for creating the desired smart IoT services using our modeling tool, as well as for getting involved in the development of the prototype as a contributor.

6.3 Validating RQ 1

We implemented the proposed approach - as explained above - that inherently included adding DAML concepts to the modeling level of ThingML [4, 24, 25, 26]. Thus, we validated RQ 1 that was stated in Section 4.1 by showing the feasibility and efficiency of this approach as explained below.

The case study used here was the one of smart energy systems, specifically the intrusive approach to load monitoring as introduced in Section 6.1 (see Figure 6.1).

Some electricity providers, especially those who possess smart grids, may offer certain discounts if the electrical appliances with higher consumption levels are avoided during peak hours. Let us assume, there existed a database server that read the values of the smart meters periodically and stored them for various smart home and ambient assisted living use cases. In this case study, we considered a smart grid that was also granted access to read this database. For the smart grid, it was only important whether a certain

---

3See https://github.com/arminmoin/ML-Quadrat/blob/master/ML2/org.thingml.samples/src/main/thingml/ML2 Demo_PingPong.thingml
high energy consuming appliance (e.g., a washer dryer) had been turned on during peak hours or not. The exact power consumption did not really matter here. However, due to various reasons, such as sensor malfunctions, power or network outages, or database failures, one might be faced with several missing values in the database. In this work, we deployed ML models as explained below, in order to predict the state (ON/OFF) of the washer dryer when the data were missing. If we required the numerical value of the missing items, for example, in order to improve the quality of predictions of the ML models for other missing values in the future, then regression would be useful (see scenario 3 below). In the following, we elaborate on the specific ML tasks.

We considered four different scenarios: (i) Classification, (ii) Clustering, (iii) Regression, and (iv) Black box ML. In each case, the software model instance comprised twelve things: the nine electrical home appliances above, the said database server, as well as a meter that measured the aggregate load of the entire house, and a DAML server that was responsible for the predictions of possible missing values in the database (see Figure 6.1). In fact, in practice, the database server and the DAML server might or might not be deployed on the same physical node. Moreover, a gateway could have been deployed at the entrance of the house. However, since the IoT vision advocates direct machine-to-machine communications and direct connections of the devices using their unique addresses [36], we skipped the gateway in the implementation. Figure 6.1 illustrates the overall architecture of the system.

Each meter would send the active power of the corresponding appliance to the database server every eight seconds. Further, the DAML server would send a query to the database server in a periodic manner (e.g., once every 15 minutes), asking for the latest sensor readings (i.e., the latest active powers of the nine appliances, and the most recent aggregate load of the house). Once the DAML server received the response from the database server, which would include the ten requested values as message parameters, the DAML server could make a prediction about the missing values that were marked, for example, by NaN in the database.

In the following, we illustrate the said scenarios. The full implementations of the respective model instances are available in [3].

**Scenario 1: Classification (Supervised ML)**

We assumed that the loads or active powers of the appliances named in Section 6.1 were given together with the aggregate load of the house for time $t_i$. The task was to predict the binary status (ON/OFF) of the washer dryer at time $t_i$. The status of the washer
6.3 Validating RQ 1

dryer was used for the binary class labels of samples in the training dataset. We let the software model train the supervised ML model using 80% of the available data. Thus, we kept 20% of samples for testing the ML model, which is common practice in ML. For example, the Scikit-Learn [29] library offers the `train_test_split` method which is widely used [111]. This method, by default, dedicates 25% of the data to the test dataset unless another value is set for the `test_size` parameter. Moreover, please note that we did not shuffle the data (i.e., we did not randomly split the data). Since they were sequential (namely time series) data, the order of the data instances mattered. The supervised ML method deployed in this example was the Multi-Layer Perceptron (MLP) classifier from the Artificial Neural Networks (ANN) family with one hidden layer of size 100, the `Relu` activation function, the `Adam` optimizer, the `Sparse Categorical Cross Entropy` loss function, and the default values for the rest of the arguments/parameters of this ML method in the Scikit-Learn [29] library.

The created software model instance had 545 lines in the textual form. The model-to-code transformations generated 4,032 Lines of Code (LoC) out of this. The generated source code contained 3,875 lines of Java code and 157 lines of Python code. The latter was responsible for the DAML functionalities, and was seamlessly integrated with the Java code using the `Java Process Builder` API. Note that the scenarios described below also exhibited the same number of LoC since we generated the APIs of the DAML library (in this case Scikit-Learn [29]) and only the name of the respective ML method, as well as certain parameters/arguments changed (but the number of the lines of code remained unchanged).

Furthermore, training the said ML model took 3,552 seconds, and it performed with 100% accuracy on the unseen test data. The ground truth came from the mentioned open data (i.e., the REFIT datasets [136, 137]). Typical ML performance metrics include but are not limited to `Accuracy`, `Precision`, `Recall`, and `F1-Measure`. In the case of binary classification, with the `positive` and `negative` classes, these are defined as follows:

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (6.1)
\]

\[
\text{Precision} = \frac{TP}{TP + FP} \quad (6.2)
\]

\[
\text{Recall} = \text{Sensitivity} = \frac{TP}{TP + FN} \quad (6.3)
\]

\[
F1 - \text{Measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (6.4)
\]
In the equations above, TP, TN, FP, and FN are the *True-Positive*, *True-Negative*, *False-Positive*, and *False-Negative* number of cases, respectively.

In the said experiment, the other ML performance metrics, namely the precision, recall and F1-measure were 99.9%, 100% and 99.9%, respectively. The high performance was foreseeable given the fact that the ML task was not challenging for the MLP ANN classifier, which is a highly capable one. In any case, the focus of this case study was not on measuring the performance of the ML methods since we only deployed the APIs of the target libraries for this purpose. The focus was rather on showing the feasibility of the proposed approach through the working examples. Hence, the reported performance figures in this section serve only for information purposes and are not supposed to contribute to the validation.

**Scenario 2: Clustering (Unsupervised ML)**

Again, we assumed that the loads or active powers of the above-mentioned appliances were given together with the aggregate load of the house for time $t_i$. Also, we had the same task as above, namely predicting whether the washer dryer was ON or OFF at time $t_i$. However, the training dataset this time had no *class labels* for the data instances. This means, we did not know which sample in the training data belonged to the OFF state of the washer dryer, and which one corresponded to its ON state. The goal was to use the available data to train a clustering ML model that could group the instances into two clusters: cluster A and cluster B. Cluster A corresponded to the OFF state of the washer dryer. In contrast, cluster B meant the washer dryer had been ON. The unsupervised ML method deployed in this example was the K-Means clustering method with the values 2 and 10 provided for the arguments/parameters regarding the desired number of clusters and the random state of the algorithm, respectively. For the rest of the arguments/parameters of the method, the default values for this method in the Scikit-Learn [29] library were considered.

Furthermore, training the said clustering model took only 13 seconds (extremely fast compared to the supervised model above), and it performed with 92% accuracy on the unseen test data.

Figures 6.10 and 6.11 show a small part of the corresponding software model instance using the textual and the tree-based views of the concrete syntax in the EMF.
**Scenario 3: Regression (Supervised ML)**

This use case scenario was very similar to the first scenario above. However, instead of predicting the ON/OFF class labels, the task was to predict the numerical values of the active power of the washer dryer. We deployed the MLP ANN Regressor in Scikit-Learn [29].

For measuring the performance of regression, the typical error measures, *Mean Absolute Error (MAE)*, also known as the *L1-Norm*, as well as the *Mean Squared Error (MSE)*, also known as the *L2-Norm* or the *Euclidean Norm* are common choices. These are defined as follows:

\[
MAE = \frac{1}{n} \sum_{i=1}^{n} | \hat{y}_i - y_i |
\]

(6.5)

\[
MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2
\]

(6.6)

Here, *n* is the number of data instances, \( \hat{y}_i \) is the predicted numerical label for the *i* – th data instance, and \( y_i \) is the actual numerical label for this data instance.

The achieved MAE and MSE in the experiment above were 10.1 and 29.962.1, respectively.

**Scenario 4: Black box ML**

For this scenario, we trained an unsupervised ML model without using the proposed approach. Thus, we developed the ML part manually. However, we used the same dataset. Then, we connected the pre-trained ML model to the software model using the black box ML mode. The rest was the same as the unsupervised ML scenario above (including the performance). Figure 6.12 demonstrates a small part of the respective software model instance.

**User study**

So far, the validation was concerned with the feasibility of the proposed approach in the context of RQ 1. Below, we elaborate on an empirical user study that we conducted with external evaluators to highlight the productivity leap, as well as the satisfaction of the tool users (i.e., practitioners).
We asked four external experts in software engineering to use and evaluate our DSML through a number of experiments in a four-hour one-on-one video call over the Internet (Zoom) with short breaks in between. Two of them had a background in ML as well. Moreover, two of them worked in academia and the other two worked in the industry. Further, two out of four possessed a PhD, whereas the rest had a Master’s degree. Last but not least, they all belonged to the age group of 25-39 years old, and one of them was a female. Table 6.1 illustrates the self-reported levels of expertise of the evaluators in various fields, collected before carrying out the user experiments.

<table>
<thead>
<tr>
<th>Eval.#</th>
<th>Software Engineering</th>
<th>DAML</th>
<th>MDSE</th>
<th>IoT/CPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>2</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>3</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>4</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>Medium</td>
</tr>
</tbody>
</table>

The evaluators were familiar with Java and Python programming. However, none of them had any background knowledge in the deployed DSMLs (neither in ThingML [4] nor in ours). During the four-hour sessions with the evaluators, we first delivered a 50-minute tutorial for using the proposed DSML, as well as the prior work on which we had built our DSML. To this aim, we had already prepared a few samples, including a HelloWorld example. Moreover, we offered them our web-based prototype. We asked each evaluator to work on two tasks in three modes: (a) Using pure manual software development (i.e., no MDSE); (b) Using the prior work, namely ThingML [4]; (c) Using the proposed DSML. We changed the orders of the tasks, as well as the orders of the modes for the four participants to avoid any bias and make the experiments fair. Both tasks were based on the case study set out above concerning smart energy systems. However, in the first task, we asked the evaluator to use supervised ML (specifically classification as in the first scenario above), whereas in the other task we asked for unsupervised ML (specifically clustering as in the second scenario above). The use case scenario involved 12 things, and implementing each of them would give the evaluator one point. An incomplete, but satisfactory implementation might result in 0.25, 0.5, or 0.75 points, depending on the completeness and correctness of the implementation. Also, implementing the DAML component of each thing (if it should have any) would have one extra point (which might be granted only partially, depending on the status...
of the implementation as mentioned before). Table 6.2 summarizes the obtained points of the evaluators for all tasks and modes. For each task, they had 75 minutes time that included 25 minutes per mode. During the experiments, they were allowed to maintain their access to their resources, such as tutorials on the Internet, and their own prior work, to make the experiments similar to the real-world practices of software developers and data scientists.

For the pure manual developments (i.e., in mode a), we asked them to use Python for the ML part, with the APIs of the Scikit-Learn [29] library and the ANN MLP classifier for the supervised task (i.e, task 1), as well as the K-Means clustering method for the unsupervised task (i.e., task 2). For the rest of their manual implementations, they were free to choose between Python and Java. However, in mode b, they must deploy our web-based interface that offered the DSML and the code generators of the prior work too, and implement the ML part manually in Python, so that their Python code could call the Java APIs of the generated Java code. Finally, in mode c, no manual development should occur. They only used our web-based interface that offered our DSML and code generators to create their model instances. The full source code could be generated automatically. For the ML part, we generated Python code that was automatically integrated with the rest of the generated code.

As illustrated in Table 6.2, using the proposed approach (see the rows 1-c and 2-c) increased the scores of the evaluators, both compared to the prior work (see the rows 1-b and 2-b) and to the pure manual software development (see the rows 1-a and 2-a). The last column illustrates the total sum of the maximum possible scores for all of the evaluators, whereas the one before last column shows the total sum of the scores achieved by the evaluators in the experiments. Thus, we argue that the proposed approach may contribute to the improvement of the software development process efficiency. According

Table 6.2: The scores of the 4 evaluators (Eval. #1-4).

<table>
<thead>
<tr>
<th>Task-Mode</th>
<th>Eval. #1</th>
<th>Eval. #2</th>
<th>Eval. #3</th>
<th>Eval. #4</th>
<th>Total Score</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-a</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>52</td>
</tr>
<tr>
<td>1-b</td>
<td>0.5</td>
<td>5.5</td>
<td>3.25</td>
<td>10.25</td>
<td>19.5</td>
<td>52</td>
</tr>
<tr>
<td>1-c</td>
<td>2.25</td>
<td>2</td>
<td>5</td>
<td>12</td>
<td>21.25</td>
<td>52</td>
</tr>
<tr>
<td>2-a</td>
<td>0</td>
<td>1</td>
<td>2.25</td>
<td>1</td>
<td>4.25</td>
<td>52</td>
</tr>
<tr>
<td>2-b</td>
<td>2.25</td>
<td>1.25</td>
<td>4</td>
<td>2</td>
<td>9.5</td>
<td>52</td>
</tr>
<tr>
<td>2-c</td>
<td>2.25</td>
<td>1.25</td>
<td>5</td>
<td>5</td>
<td>13.5</td>
<td>52</td>
</tr>
</tbody>
</table>
6 Implementation, Validation, and Evaluation

to the experiments, the performance leap has been around 25% on average, compared
to the prior work (i.e., ThingML [4]) and around 236% compared to the pure manual
software development.

We believe that the selected ML tasks were rather easy and only for one platform. One
should be able to perceive a greater value in our proposed approach once heterogeneous
IoT cloud and edge platforms need to be deployed. In the conducted experiments, many
evaluators just started working on the DAML part from the very beginning. This should
have resulted in a smaller difference between the productivity of software development in
modes b and c. Nevertheless, even 25% productivity leap is considerable and beneficial.

Finally, we asked the opinions of the evaluators about their overall experience and
satisfaction through a brief questionnaire at the end of the session. Compared to the
prior work (ThingML [4]), two evaluators (#1 and #4) rated their level of satisfaction
about the proposed approach as high. Moreover, the other two evaluators chose the
option medium. The options were high, medium, and low. In contrast, when compared
to pure manual software development, one of the evaluators selected the option low.
However, they emphasized that this answer was given based on the exercises they worked
on, in which the selected IoT platforms were not heterogeneous and it was rather easy
for them to implement it manually. The other evaluators chose the answer options
high, medium and again high concerning this question. Hence, all in all, we argue that
the proposed approach may contribute to the users (i.e., practitioners) experience and
satisfaction.

6.4 Validating RQ 2

RQ 2, which was stated in Section 4.1, was validated by showing the feasibility and
deployability of the option of keeping two separate modeling layers, namely Platform-
Independent Model (PIM) and Platform-Specific Model (PSM). To this aim, we illus-
trated how we use the same PIM with different PSMs that extend this PIM, for example,
for supporting TinyML on highly resource-constrained IoT edge devices, while also sup-
porting ML on more powerful systems with GPUs. A PIM, which is similar to the PIM
layer in MDA [30, 31], can be considered as the common denominator of the PSMs for

---

4In the case of deploying pre-trained ML models on the resource-constrained microcontrollers with
ultra-low power consumption in the range of 1 milliwatt, we are dealing with TinyML. These micro-
controllers possess main memories (RAM) in the order of tens to hundreds of kilobytes. In addition,
their persistent flash memories can be in the order of kilobytes to megabytes. Moreover, their CPU
clock speeds might be as low as just tens of MHz. Hence, they are small and highly energy efficient.
Last but not least, they are relatively inexpensive and can be ordered in large quantities [142].
a particular IoT service that must run on several heterogeneous IoT platforms. PIMs are at a higher level of abstraction than PSMs and specify the business logic of the IoT services regardless of the platform-specific details. Hence, the practitioner may concentrate on the overall, platform-independent structural and behavioral design of the software system architecture without any concerns about the possible lack of knowledge and skills in the diverse hardware, software, network, and AI technologies that are used in the heterogeneous IoT edge devices and in the cloud.

The case study used here for validating RQ 2 was the one of predictive maintenance (i.e., condition-based monitoring) of the hydraulics system depicted in Figure 6.2 in Section 6.1. In fact, this system was a hydraulic test rig deployed in Saarbrücken, Germany [11]. A test rig or test station is used to test and assess the capability and performance of components for industrial use [143]. The hydraulic test rig was equipped with multiple sensors and the sensor data, as well as the data about the working conditions and status of the system had been provided by the ZeMA gGmbH research center for Mechatronics and automation technologies as open data [144]. We used the data from the following 3 sensors in order to predict any possible internal leakage of the main pump: (i) The vibration sensor (VS1) of the main pump. Its readings had been recorded in the mm/s unit and at a frequency of 1 Hz (i.e., once a second). (ii) The Electrical Power Sensor (EPS1) of the main pump. Its readings had been recorded in Watts and at a frequency of 100 Hz. (iii) The System Efficiency (SE) factor that was not a real (i.e., physical) sensor, but a virtual one. Hence, its values had been determined by combining different directly measured values [11]. Moreover, it was a percentage and had a frequency or sampling rate of 1 Hz. The hydraulic test rig repeated periodic constant load cycles of 60 seconds. We required the sensor data for one cycle in order to predict whether the main pump was prone to any internal leakage or not. We let an ANN model perform this prediction. The ML features that were used to train this model were the above-mentioned sensor values, namely VS1, EPS1, and SE. As the sampling rates or frequencies were not identical, one could, for example, down-sample EPS1 that had a frequency of 100 Hz to 1 Hz. However, for our use case, we kept it as it was. Therefore, there existed 60 features for VS1, 6,000 features for EPS1, and another 60 features for SE per system cycle. One Boolean/Binary class label was already assigned to each cycle: True (i.e., leakage positive) or False (i.e., leakage negative). The dataset contained 2,205 data instances (i.e., system cycles). Out of these 2,205 instances, 1,221 instances/cycles (55%) corresponded to no leakage (i.e., False), and the rest corresponded to leakage (i.e., True).
Since we were dealing with time series data, in which the order of the data instances matters, we avoided shuffling the data samples/instances. We separated the dataset into two parts, and dedicated 80% of the available data to the training and validation dataset, and the rest to the test dataset for the evaluation. The latter must remain unseen by the ML model to make a fair evaluation possible.

We standardized the numeric training data using Z-Scores, and trained an ANN model using the data. It transpired that an MLP ANN with three layers (input, hidden, and output) could accomplish the prediction task with a high accuracy, precision, and recall. The hidden layer was a Dense layer with 32 units in the first experiment, and 8 units in the second experiment described below, as well as the Relu activation function. Further, the output layer had 2 units, and the Sigmoid activation function. Moreover, we used the Adam optimizer, the Binary Crossentropy loss function, a learning rate of 1e-5, a batch size of 100, 200 training epochs, as well as early-stopping with a patience level of 3. Figure 6.13 depicts the changes of the loss function and the accuracy during the training of the ML model.

We could deploy more complex and advanced ANN architectures, for example, Recurrent Neural Networks (RNNs), such as Long-Short Term Memories (LSTMs) [39]. However, since the architecture already performed well for our purpose, and we preferred a more compact ML model, to avoid unnecessary complexity, we contend that the architecture in its present form was sufficient. According to the experimental results on the test dataset, that are illustrated in Table 6.3, the accuracy, precision, and recall are 97%, 97%, and 97%, respectively, for the first experiment (i.e., with 32 units in the hidden layer of the above-mentioned ML model), and 80%, 86%, and 80%, respectively, for the second experiment (i.e., with 8 units in the hidden layer of the above-mentioned ML model) on an Intel x86 platform with the Linux Operating System (OS) and Python code that deployed the TensorFlow [28] library. This Linux server had 45 GB of main memory (RAM), and 10 Intel Xeon 2.3 GHz Processors. The respective rows in Table 6.3 are colored in gray. As shown, the more compact ML model (namely the latter experiment) performed faster. Thus, it required only 86 milliseconds for the entire test dataset instead of 119 milliseconds in the first experiment (i.e., 38% time reduction). However, the accuracy and recall were reduced by 17.5% each, and the precision fell by 11.3% in the second experiment with the more compact ML model that was 74.5% smaller in size.

Note that we trained the ML model, which conducted the predictive maintenance, only on one platform, namely the above-mentioned Linux server. However, we deployed
and executed the predictive maintenance service, once the ML model training was accomplished, on three different platforms: (i) The said Linux server. (ii) A Raspberry Pi board (as explained below). (iii) An Arduino microcontroller board (as explained below). Figure 6.14 presents the PIM and the PSM layers, as well as the said platforms. Note that the IoT services were generated fully automatically out of the PSMs. In addition, the generated artifacts included source code, build scripts, as well as the right ML model format.

The second and the sixth rows in Table 6.3 demonstrate the experimental results for the first and the second experiment on the second platform, namely a Raspberry Pi (RPI) board, respectively. This was a Raspberry Pi 3 B+ [13] board with the Raspberry Pi OS (formerly known as Raspbian) and Python code that deployed the TensorFlow Lite [140] library. This library provides an API for an ML model converter that enables generating an optimized ML model in the FlatBuffers [145] serialization format, and with the .tflite file extension [140]. As we can see in the table, this conversion resulted in 67% and 68% ML model size reduction in the first and the second experiments, respectively, without compromising the ML model performance in terms of its accuracy, precision, and recall. Nevertheless, it is clear that predictions on the RPI platform needed more time, due to the limitations of the computational resources, compared to the Linux server. The increases in the prediction time for the entire test dataset were 1,100% and 694% for the first and the second experiments, respectively.

In addition, we applied a technique, called post-training quantization [140] in both experiments. The results are illustrated on the third and the seventh rows in Table 6.3. Hence, with a negligible compromise in the ML model performance in terms of accuracy, precision and recall, we reduced the ML model size considerably and sped up its predictions too. For instance, in the case of our first experiment on the RPI platform, we did not face any reduction in the accuracy, precision, or recall. Also, in the second experiment, the accuracy and recall remained the same, while the precision was reduced by only 1%. However, the quantization technique resulted in an ML model size reduction of 75% and 74% in the first and the second experiments, respectively. Note that this quantized ML model made predictions 60% and 29% faster in the first and the second experiments, respectively. In this case, quantization led to converting all of the float32 weights of the ML model to int8 values.

While both the non-quantized and the quantized variants of the above-mentioned ML model fitted into the main memory of the RPI board, for the TinyML platform, namely the Arduino Nano 33 BLE Sense microcontroller [12], the situation was different. To
deploy the ML model on this platform, we had to use the xxd Unix/Linux command to generate a hexadecimal dump of the mentioned FlatBuffers [145] model as a C Byte Array. We stored the resulting ML model in a C++ source file with the .cc extension. This could be used via the TensorFlow Lite for Microcontrollers [141] library on the Arduino microcontroller. However, the main issue was that the hexadecimal dump required more space on the disk than the efficient FlatBuffers [145] serialization. Note that in the case of the first experiment, the C Byte Array had a size of 198 KB (i.e., with 198 thousands elements). However, its hexadecimal dump required 1.2 MB disk space. This was 506% larger. Since the main memory of the microcontroller had only 1 MB, which was even relatively large compared to many TinyML platforms, we could not deploy this ML model on the Arduino platform. In fact, this was the reason that we conducted the second experiment with a more compact ML model. Here, we had a C Byte Array of 51 KB (i.e., with 51 thousands elements). Again, the hexadecimal dump required more space. In this case, it took 316 KB on the disk. Therefore, it could fit into the main memory of the TinyML platform. The results of both experiments on Arduino are shown on the fourth and the eights rows of Table 6.3.

For the final part regarding the actual assessment and validation of RQ 2 specifically, we realized the above-mentioned use case with our proposed DSML and let the full implementation become generated automatically out of the software model instances (PSMs). The generated code could create, train, and deploy the said ML models. In fact, it could generate the right formats of the ML models for each target platform as required. We implemented the required PIMs and PSMs. Two PIMs and two PSMs were needed for training the ML model on the x86 Linux machine. The two PIMs were responsible for creating and training the ML models in the two experiments, respectively (namely, rows 1-4 and rows 5-8 in Table 6.3). Here, we illustrate the PIM for the first experiment. Figure 6.15 and Figure 6.17 show part of the platform-independent and platform-specific software model instances for the training on the x86 Linux server. If we wanted to train the ML model on another platform, such as a Raspberry Pi, we would take the same PIM and import it in another PSM that would have been specific to that platform. The choice of the target platform for code generation could be specified through the @compiler annotation of the configurations (see Figure 6.17). For instance, @compiler python java would generate Python and Java code for the default platform, namely an X86 Linux machine. The other PIM was similar, however, the value of the parameter hidden_layer_sizes was 8 rather than 32 there.
In addition, we required two PIMs for making predictions using the ML models. Again, the two PIMs corresponded to the two experiments (namely, rows 1-4 and rows 5-8 in Table 6.3). These two PIMs for prediction were imported in the PSMs for prediction. In principle, we needed one PSM for each of the target platforms. Since we had three target platforms (x86, RPI, and Arduino) and two experiments, we had in total 6 PSMs for prediction. We depict part of one of the PSMs for prediction on the x86 platform in Figure 6.20. The other ones were also similar to this one. However, their @compiler annotations in their configurations had the values rpi_3b+_python, rpi_3b+_python_quantized, and arduino_nano_33_ble_sense_cpp for each of the respective target platforms. As said, the model-to-code/text transformations (code generators) not only produced the source code that had the APIs of these platforms, but also automatically converted the ML model to the right format for each of the specific platforms.

To make it more comprehensible, we show the behavioral part of the model instances of Figure 6.15 and Figure 6.18 in the graphical form in Figures 6.16 and 6.19, respectively. The implementation of the state machines was carried out through the statechart section of the textual model instances (see the last lines in Figures 6.15 and 6.18).

```plaintext
thing Predictive_Maintenance Training_Server includes Hydraulic_Rig_Msgs {
  provided port training_service {
    sends training_done
    receives training_request
  }
  property vs1_value: Double
  property eps1_value: Double
  property se_value: Double
  property leakage: Boolean
  data_analytics predictive_maintenance {
    labels ON
    features vs1_value, eps1_value, se_value, leakage
    prediction_results leakage
    dataset "data/hydraulic_rig.csv"
    sequential TRUE
    timestamps ON
    model_algorithm nn_multilayer_perceptron my_nn_mlp
      (hidden_layer_sizes [32], activation relu, batch_size 100,
       learning_rate_init "1e-5", epochs 200
      
    training_results "data/training_predictive_maintenance.txt"
  }
  statechart Predictive_Maintenance_Training_Server_Behavior init Preprocess {
```

**Figure 6.15**: Part of the PIM for training the ML model.
Figure 6.16: The behavioral model of the PIM for training the ML model.

```python
import "Hydraulic_Rig_Training_PIM.thingml"

configuration Hydraulic_Rig_Training_Cfg @compiler "python_java" {  
  instance dB Server : DB_Server  
  instance vS1 : vS1  
  instance ePS1 : ePS1  
  instance sE : sE  
  instance predictive_Maintenance_Training_Server :  
    Predictive_Maintenance_Training_Server  
    connector dB_Server.sensor_service => vS1.sensor_service  
    connector dB_Server.sensor_service => ePS1.sensor_service  
    connector dB_Server.sensor_service => sE.sensor_service  
    connector dB_Server.predictive_maintenance_service =>  
      predictive_Maintenance_Training_Server.training_service
}
```

Figure 6.17: Part of the PSM for training the ML model on an x86 Linux platform. The PIM of Figure 6.15 is imported here.
6.4 Validating RQ 2

Figure 6.18: Part of the PIM for predictions using the ML model.

Figure 6.19: The behavioral model of the PIM for predictions using the ML model.
6 Implementation, Validation, and Evaluation

```python
import "Hydraulic_Rig_Prediction_PIM.thingml"

configuration Hydraulic_Rig_Prediction_Cfg @compiler "python_java" {
    instance dB_Server : DB_Server
    instance vS1 : VS1
    instance ePS1 : EPS1
    instance sE : SE
    instance predictive_Maintenance_Prediction_Server : Predictive_Maintenance_Prediction_Server

    connector dB_Server.sensor_service => vS1.sensor_service
    connector dB_Server.sensor_service => ePS1.sensor_service
    connector dB_Server.sensor_service => sE.sensor_service
    connector dB_Server.predictive_maintenance_service => predictive_Maintenance_Prediction_Server.predictive_service
}
```

**Figure 6.20:** Part of the PSM for predictions using the ML model on an x86 Linux platform. The PIM of Figure 6.18 is imported here.

**Table 6.3:** Experimental results for validating RQ 2. RPI, Ard., and Q. stand for Raspberry Pi, Arduino, and Quantized, respectively.

<table>
<thead>
<tr>
<th>Experiment &amp; Platform</th>
<th>Prediction time (s)</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>ML size</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1, x86 Linux</td>
<td>0.119</td>
<td>97%</td>
<td>97%</td>
<td>97%</td>
<td>2.4 MB</td>
<td>P</td>
</tr>
<tr>
<td>2 1, RPI</td>
<td>1.43</td>
<td>97%</td>
<td>97%</td>
<td>97%</td>
<td>785 KB</td>
<td>P</td>
</tr>
<tr>
<td>3 1, RPI, Q.</td>
<td>0.572</td>
<td>97%</td>
<td>97%</td>
<td>97%</td>
<td>198 KB</td>
<td>P</td>
</tr>
<tr>
<td>4 1, Ard., Q.</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>1.2 MB</td>
<td>P</td>
</tr>
<tr>
<td>5 2, x86 Linux</td>
<td>0.086</td>
<td>80%</td>
<td>86%</td>
<td>80%</td>
<td>613 KB</td>
<td>P</td>
</tr>
<tr>
<td>6 2, RPI</td>
<td>0.683</td>
<td>80%</td>
<td>86%</td>
<td>80%</td>
<td>197 KB</td>
<td>P</td>
</tr>
<tr>
<td>7 2, RPI, Q.</td>
<td>0.482</td>
<td>80%</td>
<td>85%</td>
<td>80%</td>
<td>51 KB</td>
<td>P</td>
</tr>
<tr>
<td>8 2, Ard., Q.</td>
<td>218</td>
<td>80%</td>
<td>85%</td>
<td>80%</td>
<td>316 KB</td>
<td>P</td>
</tr>
</tbody>
</table>

6.5 Validating RQ 3

To assess and validate RQ 3, which was set out in Section 4.1, we supported the practitioners using the proposed approach in carrying out the required DAML tasks through AutoML in two ways. First, we provided them with useful hints and tips at the design/modeling time, and at the time of program synthesis (i.e., code generation). If they set
the AutoML mode of the tool to ON, we enforced the recommendations. Otherwise, we simply informed them about our recommendations. Second, we provided assistance to the practitioners concerning the selection of the most appropriate ML model family/architecture for their task at hand, as well as for making the most efficient choices regarding the hyperparameters (e.g., the learning algorithm or optimizer).

The former (i.e., hints and tips at the design time) was realized based on the documentation of the APIs of the target ML libraries and frameworks, as well as the principles and best practices in DAML, through the model-to-code/text transformations, as well as via the context-conditions (static semantics) at the grammar (meta-model) level. However, for the latter (i.e., selection of the ML model family/architecture and hyperparameters), which was more challenging, we constructed a tree-structured search space by considering a number of ML methods that had proven useful for the selected use case scenario of the case study that is described in the following.

Figure 6.21 illustrates the mentioned search space for a number of ML methods and the possible hyperparameters that must be automatically tuned for each of them should they be selected by the AutoML engine.

We considered the possible choices for setting the hyperparameters as listed below:

(a) criterion $\in \{\text{MSE (Mean Squared Error), Friedman\_MSE, MAE (Mean Absolute Error)}\}$; (b) min sample split $\in \text{Uniform [2,200]}$; (c) n estimators (i.e., number of estimators) $\in \text{Uniform [5,100]}$; (d) optimizer $\in \{\text{Adam, Nadam, RMSprop}\}$; (e) learning rate $\in \{1 \cdot 10^{-2}, 1 \cdot 10^{-3}, 1 \cdot 10^{-4}, 1 \cdot 10^{-5}\}$; (f) loss function $\in \{\text{MSE, MAE}\}$; (g) n layers (i.e., number of layers) $\in \text{Uniform [5, 8]}$; (h) dropout probability $\in \text{Uniform [0.1, 0.6]}$; (i) sequence length $\in \{64, 128, 256, 512, 1024\}$.

We deployed Bayesian Optimization (BO) to enable the automated selection of the best ML model architecture/family and the most appropriate hyperparameters for the selected ML model given a specific use case and a dataset. There exist various open-source libraries that offer BO. We picked the most widely-used library, namely Hyperopt [146]. This library required a Tree-structured Parzen Estimator (TPE) model, and provided distributed, asynchronous hyperparameter optimization. Moreover, it supported search spaces with different types of variables (continuous and discrete), varied search scales (e.g., uniform vs. log-scaling), as well as conditional variables that were only meaningful for certain combinations of other variables [146].

The case study deployed for validating RQ 3 was the non-intrusive variant of the smart energy systems use case that was described in Section 6.1, namely energy disaggregation
6 Implementation, Validation, and Evaluation

(NIALM/NILM). To this aim, we used the data of the REDD [135] and the UK-DALE
[138] datasets from the US and the UK, respectively.

Figures 6.22 and 6.23 illustrate the overall architecture, as well as the behavioral model
of the energy disaggregation server of the IoT service for the case study validating RQ
3, respectively.

We used the implementation of the AutoML engine that was produced through the
co-advisory of Ukrit Wattanavaekin’s Master’s Thesis, and is available as open-source
software on Github [147], for choosing the ML model family/architecture and optimizing
the hyperparameters. This is a standalone prototype. However, there exist another Au-
toML part, which is already seamlessly integrated into our prototype (i.e., ML-Quadrat).
This part is essentially rule-based. In order to use this, the user of our tool may set the
AutoML parameter/option in the model instance to ON. This way, certain choices, such
as hyperparameters, might be overridden automatically. If AutoML is set to OFF, there
might be some messages and warnings regarding certain choices, for example, if some
parameter value does not fit into the recommended range that is mentioned in the API
documentation of the respective ML library.

It transpired that simple ML models, such as DT that had not been studied for
this problem could actually outperform some of the more complex deep learning ML
model architectures that were proposed in the literature. Note that training a DT is
much faster and requires much less data and computational, as well as energy resources,
compared to a deep learning model. This example shows the benefits of the proposed
AutoML-based approach, even for the ML experts themselves, since they could avoid
trial-and-error practices for selecting the ML model and tuning its hyperparameters.
In addition, another deep learning ML model, namely GRU, which was proposed by
this work, outperformed the state of the art approaches. Figure 6.24 illustrates the
experimental results of our benchmarking study with the following NIALM approaches:
(i) CO [148], (ii) DAE [149], (iii) DT, (iv) Dictionary-based (our implementation of
[150]), (v) DNN-HMM (our implementation of [151]), (vi) FCNN, (vii) FHMM [152],
(viii) GRU, (ix) LSTM [149]. Note that applying DT, FCNN, and GRU to this problem
was our contribution. Table 6.4 shows the average error (MAE) for the mentioned
approaches. Note that we used 37 days worth of data from house 1 of the REDD
dataset [135] with a sampling rate of 0.05 Hz.

86
Table 6.4: The average MAE of the studied NIALM approaches.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Approach</th>
<th>MAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>DT</td>
<td>14.86</td>
</tr>
<tr>
<td>2nd</td>
<td>GRU</td>
<td>19.55</td>
</tr>
<tr>
<td>3rd</td>
<td>LSTM [149]</td>
<td>25.77</td>
</tr>
<tr>
<td>4th</td>
<td>DAE [149]</td>
<td>41.44</td>
</tr>
<tr>
<td>5th</td>
<td>FCNN</td>
<td>48.96</td>
</tr>
<tr>
<td>6th</td>
<td>Dictionary-based [150]</td>
<td>61.15</td>
</tr>
<tr>
<td>7th</td>
<td>DNN-HMM [151]</td>
<td>118.52</td>
</tr>
<tr>
<td>8th</td>
<td>FHMM [152]</td>
<td>139.40</td>
</tr>
<tr>
<td>9th</td>
<td>CO [148]</td>
<td>204.83</td>
</tr>
</tbody>
</table>

6.6 Validating RQ 4

We conducted four one-on-one interviews with external subject matter experts in the field of SE, as well as a self-administered online survey with 121 participants. Out of these 121 survey participants, 42 submitted answers to all of the questions of the survey questionnaire, whereas 79 submitted only partial answers or no answer at all. The one-on-one interviews were rather exploratory, in order to shape the survey questionnaire. However, the goal of the self-administered online survey (which was also a form of interview, but not one-on-one) was to validate the proposed approach to enabling AI/ML in architecture frameworks, mainly the identified stakeholders, their concerns, viewpoints/views, and model kinds, as presented in Section 5.4.

The interviewees in the first phase had the following profiles: (i) gender: female, age group: 25-39 years old, job and expertise: doctoral researcher in the field of SE working at a research center in Munich, Germany; (ii) gender: male, age group: 25-39 years old, job and expertise: senior software engineer with a PhD working at a company in Munich, Germany; (iii) gender: male, age group: 25-39 years old, job and expertise: sales software engineer working at a large corporate in Munich, Germany; (iv) gender: male, age group: 25-39 years old, job and expertise: data scientist and software engineering expert working at a financial institute in Zurich, Switzerland.

The first two interviews were focused on viewpoints/views, whereas the last two were concentrated on modeling in general, and modeling specifically with our approach, respectively. These exploratory interviews were mostly carried out with open questions, such as asking about the opinion and feedback of the interviewees about the identified gap in the state of the art, their state of practice at their workplace regarding this topic, the proposed solution to the problem in our work, the need for introducing new
architecture viewpoints/views, different positions on viewpoints/views, the identified stakeholders and their concerns, as well as the need for new model kinds and notations. The main lessons learned from these interviews were the following:

1. The identified gaps in the state of the art, and state of practice were indeed new, relevant, and interesting in the view of the subject matter experts who were asked about it (namely the first and the second interviewees).

2. When asked whether they would introduce new architecture viewpoint(s) or new architecture view(s) for AI/ML, the first interviewee was leaning towards introducing a new viewpoint, whereas the second interviewee preferred introducing new views for this topic. However, they both agreed that new viewpoint/views were required for addressing AI/ML in architecture frameworks.

3. The fact that the first interviewee believed that no new view was required, whereas the second interviewee saw new views as necessary to handle AI/ML at the architecture level, highlighted a major disagreement. When asked for more details in each interview, they shared their rather opposite and conflicting opinions on this matter. The first interviewee believed that AI/ML/DEA models, algorithms, and methods were still functions and processes, more or less like the rest of the software systems. She saw no special nature for them to be treated differently. Therefore, she thought that they could be modeled with the existing views, model kinds, and notations as we model the other functions and processes. However, the second interviewee stressed that the AI-functionalities of AI-enabled software systems were special, not just like the other functions and processes, and they could have an impact on the other ones too. Thus, he believed that we needed to treat them special as well. Hence, one would need to find out to what extent AI/ML/DEA might have an impact on other parts of the system, and also ask the AI/ML/DEA experts which information they would like to see at the design stage, and with which notations and formalisms are they working on a daily basis.

4. In the list of the stakeholders with AI-related concerns, which we had initially identified, the safety engineer and data protection (privacy) officer were missing. We added them to our list (see Section 5.4) after talking to the second interviewee.

5. The third and the fourth interviewees both pointed out the unpopularity and unsuitability of UML in the industry. Despite being an international standard and quite well-known in academia, they said that UML was neither used by them nor
6.6 Validating RQ 4

by anyone in the circle of their colleagues. However, when we talked with the fourth interviewee about our domain-specific approach, he became interested in it, and could imagine the difference with general-purpose approaches, such as UML.

6. Nevertheless, the third and the fourth interviewees were familiar with UML and could understand the diagrams since they had seen and studied them in their universities.

In the second phase, we conducted a self-administered online survey. The questionnaire comprised four sections. Section 1 consisted of four questions that were focused on the best notations and formalisms for data scientists and data engineers. Further, Section 2 of the questionnaire consisted of seven questions that were concerned with the best notations and formalisms to let data scientists and data engineers communicate and collaborate with the other stakeholders, namely (i) end-users, (ii) business stakeholders, (iii) safety and regulatory compliance engineers, data protection (privacy) officers, and ethics committees/boards, (iv) database engineers/designers, (v) software engineers and software architects, (vi) system engineers and network experts, and (vii) security experts. Moreover, Section 3 of our questionnaire tried to validate the identified stakeholders and their concerns, as well as asking for any additional ones in the view of the participants. Finally, Section 4 aimed to collect information on the demographics. Following the recommendation of Bourque and Fielder [153], we put the demographic questions at the end, in order to avoid any possible discouragement of the participants. Also, it is clear that we did not disclose the particular answers of the participants even if they opted in to share their email addresses, names, and/or affiliations, for example, in order to be contacted later to receive a preprint of our respective future research publication or being acknowledged and thanked in the acknowledgments of the respective future research publication. Last but not least, we only considered the complete answer sheets in this study.

Similar to Meyer et al. [154], we followed the work of Kitchenham and Pfleeger [155] on the survey research method in software engineering, and - as mentioned - created the survey questionnaire based on the insights derived from the one-on-one expert interviews. The survey design was cross-sectional, not longitudinal, since each participant was surveyed only once in July-September 2021. Also, it was self-administered over the web. Each participant had two to four weeks to fill out the questionnaire upon receiving the invitation. If they did not reply, they would get a reminder after one week. Anonymous participation was also possible. However, we collected their IP addresses to
prevent any redundant participation. Moreover, the orders of the questions presented to the survey participants in each section, as well as the choices for each question were set randomly. We used the open-source LimeSurvey software [156] on our own server for conducting this survey study. Further, almost all of the questions were optional except for two questions in section 4 of the survey questionnaire that were concerned with the consent of the participants. Those were explicitly marked as mandatory.

The average survey participation time was 14 minutes, whereas its median was 11 minutes. Figures 6.25, 6.26, 6.27, 6.28, 6.29, 6.30, 6.31, 6.32, and 6.33 demonstrate the demographic information concerning the survey participants based on their answers to the demographic questions in section 4 of the questionnaire.

The answers they provided through the online survey are already reflected in Tables 5.1 and 5.2 in Section 5.4, in particular, in the order of the suggested model kinds, formalisms, and notations for each viewpoint/view in Table 5.2. Below, we first present the survey questionnaire. Then, we elaborate on the answers provided to each question and analyze the answers.

Survey questionnaire

Survey questionnaire section 1 of 4

1. Which formalism, notation or means is useful for documentation and presentation purposes in analytics modeling, and for the communication of data scientists with their peers? Please select all that may apply. You may use the optional text fields for any further remarks.

*Hint: Data scientists are responsible for analytics modeling. They either create new algorithms, techniques and methods or adapt and use existing ones to create data analytics and machine learning models, for example, for prediction or clustering.*

- Computational Graphs (CG), also known as Data-Flow Graphs (DFG)
- UML Class diagram
- Mathematical notation, showing the mathematical model (e.g., probability distributions, the objective/loss function, etc.)
- Dataflow diagram (e.g., using the UML Activity diagram notation) showing the upstream and downstream components in the pipeline
- Charts/diagrams/plots (e.g., histograms, pie charts, scatter plots, etc.)
2. Which formalism, notation or means is useful for documentation and presentation purposes in analytics operations, and for the communication of data engineers with their peers? Please select all that may apply. You may use the optional text fields for any further remarks.

*Hint: Data engineers are responsible for analytics operations. They create, adapt and maintain scoring engines (i.e., model consumers), and are concentrated on large-scale batch/steam data processing, for example, using the Apache Hadoop/Spark ecosystems or similar big data analytics platforms and technologies. Data engineers should not be confused with database engineers/designers.*

- Computational Graphs (CG), also known as Data-Flow Graphs (DFG)
- UML Class diagram
- Mathematical notation, showing the mathematical model (e.g., probability distributions, the objective/loss function, etc.)
- Dataflow diagram (e.g., using the UML Activity diagram notation) showing the workflows/pipelines
- Charts/diagrams/plots (e.g., histograms, pie charts, scatter plots, etc.)
- Probabilistic Graphical Models (PGM)
- Entity-Relationship (ER) diagram
- Topic Maps and/or Knowledge Graphs (e.g., RDF Graphs) and/or Ontologies
- Others (please specify)

3. When you look at a smart software/information system or enterprise as a data scientist, which information about the software/system/enterprise would you like to see to be able to describe the architecture?

*Hint: Each stakeholder or group of stakeholders often have their own understanding and viewpoint/view, thus focus on different aspects.*

- The computational operations and the flow of data among them
• The processes and/or components, and the flow of data among them
• The processes and/or components, and the flow of control among them
• The underlying mathematical model
• The data visualization (e.g., in the case of online learning / stream processing)
• The workflow/pipeline for data analytics and machine learning
• The problem/use-case domain concepts and their relationships
• Others (please specify)

4. When you look at a smart software/information system or enterprise as a data engineer, which information about the software/system/enterprise would you like to see to be able to describe the architecture?

Hint: Each stakeholder or group of stakeholders often have their own understanding and viewpoint/view, thus focus on different aspects.

• The computational operations and the flow of data among them
• The processes and/or components, and the flow of data among them
• The processes and/or components, and the flow of control among them
• The underlying mathematical model
• The data visualization (e.g., in the case of online learning / stream processing)
• The workflow/pipeline for data analytics and machine learning
• The problem/use-case domain concepts and their relationships
• Others (please specify)

Survey questionnaire section 2 of 4

1. Which formalism, notation or means is useful for the communication of data scientists and data engineers with the end-users of software and information systems? Please select all that may apply. You may use the optional text fields for any further remarks.

Hint: End-users are one of the key stakeholders of software and information systems.

• Tables/matrices
6.6 Validating RQ 4

- Probabilistic Graphical Models (PGM)
- Topic Maps and/or Knowledge Graphs (e.g., RDF Graphs) and/or Ontologies
- UML Use Case diagram
- Entity-Relationship (ER) diagram
- Charts/diagrams/plots (e.g., histograms, pie charts, scatter plots, etc.)
- Dataflow diagram (e.g., using the UML Activity diagram notation)
- Text document
- Mathematical notation, showing the mathematical model (e.g., probability distributions, the objective/loss function, etc.)
- UML Class diagram
- Computational Graphs (CG), also known as Data-Flow Graphs (DFG)
- Others (please specify)

2. Which formalism, notation or means is useful for the communication of data scientists and data engineers with the business stakeholders of software and information systems? Please select all that may apply. You may use the optional text fields for any further remarks.

*Hint:* Business stakeholders are one of the key stakeholders of software, information systems and enterprises. Examples include, but are not limited to the upper-level/middle management, the CTO/CEO, etc.

- Tables/matrices
- Probabilistic Graphical Models (PGM)
- Topic Maps and/or Knowledge Graphs (e.g., RDF Graphs) and/or Ontologies
- UML Use Case diagram
- Entity-Relationship (ER) diagram
- Charts/diagrams/plots (e.g., histograms, pie charts, scatter plots, etc.)
- Dataflow diagram (e.g., using the UML Activity diagram notation)
- Text document
- Mathematical notation, showing the mathematical model (e.g., probability distributions, the objective/loss function, etc.)
6 Implementation, Validation, and Evaluation

- UML Class diagram
- Computational Graphs (CG), also known as Data-Flow Graphs (DFG)
- Others (please specify)

3. Which formalism, notation or means is useful for the communication of data scientists and data engineers with safety and regulatory compliance engineers, data protection (privacy) officers and ethics committees/boards? Please select all that may apply. You may use the optional text fields for any further remarks.

*Hint: This is about responsibility. Safety and regulatory compliance, data protection (privacy), and ethical aspects, including inclusiveness, fairness and explainability of AI/DEA/ML are becoming increasingly important.*

- Tables/matrices
- Probabilistic Graphical Models (PGM)
- Topic Maps and/or Knowledge Graphs (e.g., RDF Graphs) and/or Ontologies
- UML Use Case diagram
- Entity-Relationship (ER) diagram
- Charts/diagrams/plots (e.g., histograms, pie charts, scatter plots, etc.)
- Dataflow diagram (e.g., using the UML Activity diagram notation)
- Text document
- Mathematical notation, showing the mathematical model (e.g., probability distributions, the objective/loss function, etc.)
- UML Class diagram
- Computational Graphs (CG), also known as Data-Flow Graphs (DFG), for example, augmented with the deployment details (i.e., on which GPU/CPU core)
- Others (please specify)

4. Which formalism, notation or means is useful for the communication of data scientists and data engineers with database engineers/designers? Please select all that may apply. You may use the optional text fields for any further remarks.
Hint: Database engineers/designers should not be confused with data engineers. Database engineers/designers are concerned with Database Management Systems (DBMS).

- Tables/matrices
- Probabilistic Graphical Models (PGM)
- Topic Maps and/or Knowledge Graphs (e.g., RDF Graphs) and/or Ontologies
- UML Use Case diagram
- Entity-Relationship (ER) diagram
- Charts/diagrams/plots (e.g., histograms, pie charts, scatter plots, etc.)
- Dataflow diagram (e.g., using the UML Activity diagram notation)
- Text document
- Mathematical notation, showing the mathematical model (e.g., probability distributions, the objective/loss function, etc.)
- UML Class diagram
- Computational Graphs (CG), also known as Data-Flow Graphs (DFG)
- Others (please specify)

5. Which formalism, notation or means is useful for the communication of data scientists and data engineers with software engineers and software architects? Please select all that may apply. You may use the optional text fields for any further remarks.

Hint: Software engineers have the development viewpoint and are primarily concerned with the actual software module organization, layers, libraries and APIs, etc. A proper communication between the DEA/ML experts and software engineers is crucial.

- Tables/matrices
- Probabilistic Graphical Models (PGM)
- Topic Maps and/or Knowledge Graphs (e.g., RDF Graphs) and/or Ontologies
- UML Use Case diagram
- Entity-Relationship (ER) diagram
6. Which formalism, notation or means is useful for the communication of data scientists and data engineers with system engineers and network experts? Please select all that may apply. You may use the optional text fields for any further remarks.

Hint: System engineers are concerned with the system configuration from a physical viewpoint; being primarily interested in the deployment of software components on the hardware resources, as well as the non-functional requirements, such as availability, reliability (i.e., fault-tolerance), throughput, and scalability. Similarly, network experts are concerned with various layers of computer networks.

- Tables/matrices
- Probabilistic Graphical Models (PGM)
- Topic Maps and/or Knowledge Graphs (e.g., RDF Graphs) and/or Ontologies
- UML Use Case diagram
- Entity-Relationship (ER) diagram
- Charts/diagrams/plots (e.g., histograms, pie charts, scatter plots, etc.)
- Dataflow diagram (e.g., using the UML Activity diagram notation)
- Text document
- Mathematical notation, showing the mathematical model (e.g., probability distributions, the objective/loss function, etc.)
- UML Class diagram
- Computational Graphs (CG), also known as Data-Flow Graphs (DFG)
- Others (please specify)
6.6 Validating RQ 4

- Computational Graphs (CG), also known as Data-Flow Graphs (DFG), for example, augmented with the deployment details (i.e., on which GPU/CPU core)
- Others (please specify)

7. Which formalism, notation or means is useful for the communication of data scientists and data engineers with **security experts**? Please select all that may apply. You may use the optional text fields for any further remarks.

*Hint: In DEA/ML, security comprises new aspects regarding the data analytics and machine learning models, for example, the adversarial attack resistance/robustness of the ML models.*

- Tables/matrices
- Probabilistic Graphical Models (PGM)
- Topic Maps and/or Knowledge Graphs (e.g., RDF Graphs) and/or Ontologies
- UML Use Case diagram
- Entity-Relationship (ER) diagram
- Charts/diagrams/plots (e.g., histograms, pie charts, scatter plots, etc.)
- Dataflow diagram (e.g., using the UML Activity diagram notation)
- Text document
- Mathematical notation, showing the mathematical model (e.g., probability distributions, the objective/loss function, etc.)
- UML Class diagram
- Computational Graphs (CG), also known as Data-Flow Graphs (DFG), for example, augmented with the deployment details (i.e., on which GPU/CPU core)
- Others (please specify)

**Survey questionnaire section 3 of 4**

1. On a scale of 1 to 5, how much is each stakeholder affected by Artificial Intelligence (AI), specifically Machine Learning (ML) as well as Data Engineering and Analytics (DEA)? 1 means not at all (i.e., no impact), and 5 means the highest level of impact.
6 Implementation, Validation, and Evaluation

*Hint:* Please select your best own estimation. If you are not sure, you may select no answer.

a) ethics committees/boards  
b) data protection (privacy) officers  
c) data engineers  
d) database engineers/designers  
e) business stakeholders  
f) safety and regulatory compliance engineers  
g) data scientists (including Machine Learning engineers)  
h) software engineers and software architects  
i) domain/knowledge engineers  
j) end-users  
k) security experts

2. Are you aware of any stakeholder(s) with concerns with respect to Artificial Intelligence (AI), specifically Machine Learning (ML) as well as Data Engineering and Analytics (DEA), whom we have not mentioned so far?  
   *Hint:* Please use the text fields below and mention each of them in one text area. Please state their roles and briefly explain how they are affected by AI/ML/DEA.

**Survey questionnaire section 4 of 4**

1. What is your sex/gender?  
   a) Female  
   b) Male  
   c) Other

2. What is your age group?  
   a) Below 18  
   b) 18-24  
   c) 25-39  
   d) 40-60
3. What is your highest academic degree?

*Hint:* Please select the one that is already conferred by a recognized higher education institute (university).

a) Bachelor’s  
b) Master’s  
c) PhD  
d) No academic degree  
e) (optional) Make a comment on your choice here.

4. What is your educational background? Please select all that may apply.

*Hint:* If you have multiple backgrounds, please indicate the one that is most relevant to your current job. Also, if you have multiple jobs, please consider your main job (i.e., the one in which you spend the majority of your working time in a week or month).

a) No higher (i.e., university-level) education  
b) Computer Science/Computer Engineering: AI/Machine Learning/Data Science/Data Analytics/Data Engineering  
c) Computer Science/Computer Engineering: Software Engineering  
d) Computer Science/Computer Engineering: other sub-disciplines  
e) Other engineering fields (please specify)  
f) Physics  
g) Mathematics / Statistics  
h) Other fields or disciplines

5. What is your current job or occupation?

*Hint:* If you have multiple jobs, please indicate the main one (i.e., in which you spend the majority of your working time in a week or month).

- Manager  
- Freelancer / self-employed
6. Which role best describes your current position?

You may optionally use the text area to write a comment. If you have multiple roles/positions/jobs, please consider the main one (i.e., in which you spend the majority of your working time).

- Data scientist / machine learning engineer/practitioner
- Data engineer / large-scale data processing expert
- Software engineer/developer
- Database engineer/designer
- System engineer
- Software architect
- Computer networks expert
- Other (please specify)
- (optional) Make a comment on your choice here.

7. Which of the following are you familiar with? Please select all that may apply.

Hint: You may optionally use the text field to explain your answer.

- Internet of Things (IoT) / Cyber-Physical Systems (CPS) / Sensor Networks
- Artificial Neural Networks (ANN) / Deep Learning
- MapReduce / Apache Hadoop / Apache Spark / Apache Kafka
- Java programming (J2SE/J2EE)
- Python programming
- Software/system/enterprise architecture frameworks, architecture viewpoints and views

8. What is your overall level of expertise in the fields of data analytics, machine learning and/or data engineering?

Hint: Please select your own best estimation.
• I do not know anything
• Beginner
• Medium expertise
• Expert

9. What is your overall level of expertise in the fields of software engineering?

Hint: Please select your own best estimation.

• I do not know anything
• Beginner
• Medium expertise
• Expert

10. (Mandatory question) Would you like to receive a preprint of the research publication?

Hint: If yes, please make sure that you provide your email address below.

• Yes
• No

11. (Mandatory question) Are we allowed to thank you in the acknowledgments section of the research publication, thus revealing your name and affiliation?

Hint: You may select no to opt out, thus staying completely anonymous.

• Yes
• No

12. You may optionally enter your preferred email address below.

Hint: We will then use this address for sending the preprint.

Answers of participants

We presented the demographic information that we acquired based on the participants answers to the question in section 4 of the survey questionnaire in Figures 6.25-6.33. Below, we illustrate their answers to the other sections of the questionnaire in Figures
6.34-6.37 for section 1, as well as in Figures 6.38-6.44 for section 2, and in Figures 6.45-6.55 for section 3 of the questionnaire.

Further, when asked for any additional stakeholders whom we might not have mentioned (see question 2 in section 3 of the questionnaire), the only new information that we received here was *quality assurance engineer*.

**Analysis of the answers**

The survey results were already reflected in Tables 5.1 and 5.2 in Section 5.4. Here, we do not repeat that. Instead, we point out the results that were either especially outstanding and notable, or unexpected and controversial in our view.

First, most participants had an educational background in SE (see Figure 6.28), and most participants had assessed their own level of expertise in SE at the expert level (see Figure 6.33). However, simultaneously, most of them were working as data scientists rather than software engineers (see Figure 6.30), while the majority assessed their own level of expertise in DEA/ML as medium rather than expert (see Figure 6.32). This fact can itself confirm our basic assumption and the identified necessity with respect to supporting software developers without professional background in DEA/ML to carry out these tasks in combination with their development work.

Second, we saw that the majority voted in question 1 in section 1 of the questionnaire for the mathematical notation as being the most suitable formalism and *model kind* for analytics modeling (see Figure 6.34). However, when asked a closely related question on this matter (i.e., question 3 in section 1 of the questionnaire), the majority emphasized on the workflow/pipeline for DAML, which is obviously not typically possible to model via the mathematical notation. Here, dataflow diagrams (not to be confused with Data-Flow Graphs, abbreviated as DFG [28]) could be useful. For instance, notations such as the UML Activity diagram could be also used to model the flow of data instead of the flow of control and processes. They could show the upstream and downstream components in the pipeline. However, when we look at the answers to question 1 in section 1 (see Figure 6.34), this has surprisingly not been among the top choices of participants for analytics modeling.

Furthermore, a rather astonishing observation was the diversity of expert opinions on the issue of the impact of AI/ML/DEA on ethics committees/boards (see Figure 6.46), safety and regulatory compliance engineers (see Figure 6.51), and security experts (see Figure 6.55). Our position and view here was rather determined and clear, for example, similar to the impact on data protection (privacy) officers (see Figure 6.47).
leaning towards grade 4 on the scale of 1-5. This might have been due to the novelty of AI/ML/DEA components of software and information systems. In fact, we believe that some experts might have answered the questions from the lens of classic systems, where safety-critical systems did not contain any piece of AI/ML/DEA, and were not data-driven/data-centric. Moreover, security of DAML models, for example, adversarial robustness/resistance of ML models (e.g., see [134]) is a new topic and field. However, we argue that this is a very important one as the security of AI/ML-enabled systems cannot be guaranteed or addressed without the security of DAML artifacts being dealt with. This argument also holds for safety, ethical, and any other regulatory aspects, including privacy and data protection.

6.7 Discussion and threats to validity

One key strength of this work for the SE community is expected to be that they gain access to the DAML methods and techniques out-of-the-box and can deploy them in their software models for the IoT. However, ML methods cannot perform well if their hyperparameters are not tuned properly and/or the data that are used for training them are not prepared well. Therefore, we handle the former, namely hyperparameter optimization using the provided AutoML engine, whereas for the latter we enable data pre-processing in the generated DAML pipelines, for example, standardization of numeric values.

Further, a major advantage of this work for the DAML community is assumed to be that they can more easily become involved in large-scale IoT projects as they will be able to work with the abstract software models that are simpler to understand, adapt and use for them. Moreover, they may introduce any desired pre-trained ML model with any arbitrary architecture, learning algorithm, and technique. This shall bring them more flexibility as they will not be limited to the pre-defined options. However, the implication for them (as well as for the SE community) is that they have to be familiar with the DSML of the modeling tool and be willing to model their desired software using this DSML.

There exist a number of possible threats to the validity of the research results. First, we validated the research questions through two case studies in Chapter 6. We showed the feasibility of the proposed approach via a number of working examples with different use case scenarios. Although this is a well-established research method in engineering research (e.g., see [110]), we only had two cases, and the selected case studies and vertical application domains might not be representative enough for the entire domain.
of IoT/CPS. Thus, the generalized conclusions made here about the entire target domain might not be rigorous.

Second, the empirical evaluations conducted in Chapter 6 for the user study involved only four professionals. Similarly, for the architecture frameworks, we had only four experts for the one-on-one interviews, as well as 121 survey participants with only 42 complete answer sheets that we considered at the end. Consequently, the conclusions drawn may not hold for a larger sample group. In addition, an ideal research design, in particular for the user study, should have involved randomized controlled experiments. However, our study was neither randomized nor had any control group. In contrast, we used convenience sampling, and invited a number of volunteers to participate in our empirical evaluation.

Further, concerning the user study, the tasks chosen for the experiments were only two rather similar programming tasks with simple DAML requirements and no combination of heterogeneous resource-constrained IoT devices. This was due to the time and resource constraints in place for the experiments with the experts, but might be biased. Ideally, the tasks should have been more diverse and possibly more tasks would have been required, in order to be fair to different participants with different strengths. Additionally, we swapped the task and mode orders. However, we cannot rule out possible biases as a result of working on one task in a certain mode, for example, using our DSML, and then in the following slot on the same task, but in a different mode, for example, via pure manual software development. Also, it is clear that the time constraint might have had an impact on the performance of evaluators in these tasks. For example, the manual task (namely, the \textit{a mode}) is expected to require more time than the tasks in the \textit{b mode} and the \textit{c mode}. Therefore, allotment of the same amount of time may not work ideally in all the modes. In fact, this was an exploratory user study/pilot study and a more sophisticated evaluation with more evaluators is required in the future. Hence, the achieved preliminary results might not be sufficient to perform a quantitative analysis.

Additionally, it is acknowledged that a full measure of process improvement in development of ML solutions would have to incorporate both quantitative and qualitative metrics for efficiency in terms of development time, as well as effectiveness in terms of the functionality Key Performance Indicators (KPIs) of the system. However, a reasonable assumption has to be that the availability of a development support environment, such as ML-Quadrat [3, 5, 14, 15, 16, 17, 18] would enable faster iterative refinement engineering of the solution, and thereby enable the development of more effective and more efficiently enhanced solutions by virtue of the speedup and facilitation of the de-
Development process, which are fundamentally underpinning both the effectiveness and efficiency of the process end to end.

Finally, as explained in Chapter 4, we searched for the related work on Google using several keywords, and then carried out the literature review based on these results. However, we cannot rule out possible external factors, such as algorithmic biases that might have led to certain works being found through this search engine at the respective times and from the respective geographical locations, and perhaps some others not ranked high enough or not found at all. Also, the choice of the specific keywords might have played a role in the achieved results.
Figure 6.4: The textual model editor showing part of a sample model for the PingPong example.
Figure 6.5: The graphical, EMF tree-/form-based model editor showing part of a sample model for the PingPong example.
6 Implementation, Validation, and Evaluation

Figure 6.6: The web-based prototype showing part of a sample model for the PingPong example.

Figure 6.7: The state machines modeling the behaviors of the three things of the smart ping-pong example.
6.7 Discussion and threats to validity

/* This is a part of the model instance. The full model instance is available in the Git repository on Github. */

tHING PingPongDataAnalytics includes PingPongmsgs {
  /* The messages are not shown here, but defined in a thing fragment, called PingPongmsgs. */
  provided port da_service { /* This port communicates with the da_service port of pingServer. */
    receives query /* This port may receive a query message from pingServer. */
    sends prediction_positive, prediction_negative /* This port may send a response to pingServer. The response might be positive (i.e., malicious) prediction or negative (i.e., non-malicious) prediction. */
  }
}

/* The properties are the local variables of the thing. */

property client_ip_address: String /* The IP address of pingClient is a String. */

property client_code: Int32 /* This is just a secret integer code that is shared between pingClient and pingServer or alternatively a serial ID number for the ping message. */

property prediction: Boolean = false /* This Boolean property shall store the prediction of the DAML model and is initialized as false here. This mean, by default, the client is non-malicious. */

data_analytics da1 /* Please see Section 5.4. */

@dalib "scikit-learn"

labels ON

features client_ip_address, client_code, prediction

prediction_results prediction
dataSet "data/ip_dataset.csv"

automl OFF

sequential TRUE

timestamps OFF

preprocess_feature_scaler StandardScaler

model_algorithm nn_multilayer_perceptron my_nn_mlp

(activation relu, optimizer adam, loss SparseCategoricalCrossentropy)

training_results "data/training.txt"

}

statechart PingPongDataAnalyticsBehavior init Preprocess {
  /* The statechart specifies the behavior of this thing. Since this thing is responsible for DAML, its behavior can be modeled via a Finite-State Machine (statechart) that has four states: preprocess, train, ready and predict. Initially, the Preprocess state is necessary to do the data preparation. */
  on entry print "Ping Pong Data Analytics Started!"

  state Preprocess {
  on entry do
    print "Ping Pong Data Analytics: Data Preprocessing"

    da_preprocess da1 /* This action carries out the actual data preprocessing / preparation. */

  end

  transition -> Train /* This leads to the transition of the state machine (statechart) to the next state, Train. */

}

Figure 6.8: Part of the model instance of the smart ping-pong example.
6 Implementation, Validation, and Evaluation

```plaintext
state Train {
  on entry do
  print "Ping Pong Data Analytics: Training\n"
  da_train dai /* This action performs the training of the DAML model. */
end
transition -> Ready /* Once the training is done, the thing shall switch to the Ready (or idle) state to simply keep waiting for the incoming queries. */
}
state Ready {
  on entry do
  print "Ping Pong Data Analytics: Ready for Prediction\n"
end
transition -> Predict
  event m: da_service?query
  /* As soon as a message is received on the da_service port, the thing must switch to the Predict state. */
  action do
    /* Additionally, the following actions must be taken. */
    client_ip_address = m.client_ip /* First, the value of the message parameter, called client_ip needs to be stored in the thing property (local variable) client_ip_address. */
    client_code = m.client_code /* Second, the value of the message parameter, called client_code must be stored in the thing property (local variable) client_code. */
  end
}
state Predict {
  on entry do
  print "Ping Pong Data Analytics: Predicting\n"
  da_predict dai(client_ip_address, client_code) /* This action asks the DAML model to make a prediction. */
  if(prediction==false)
    da_service!prediction_negative() /* If the prediction is false, send the prediction_negative message to pingServer, stating that pingClient is not likely to be an attacker. */
  else
    da_service!prediction_positive() /* Otherwise, send the prediction_positive message to pingServer, stating that pingClient is prone to be an attacker. */
  end
  transition -> Ready /* In any case, switch back to the Ready (i.e., idle) state. */
  on exit da_save dai /* This optional action results in appending the prediction to the dataset (CSV file). */
}
```

Figure 6.9: Part of the model instance of the smart ping-pong example (continued).
6.7 Discussion and threats to validity

**Figure 6.10:** The data analytics part of the software model instance in textual form.

```plaintext
data_analytics dal
@dalib "scikit-learn" {
  //@dalib "keras-tensorflow"
  //
  labels OFF
  features fridge_load, freezer1_load, freezer2_load, washing_machine_load,
  prediction_results washer_dryer_status
  dataset "data/REFIT_house1_reordered_clustering.csv"
  automl OFF
  sequential TRUE
  timestamps ON
  preprocess_feature_scaler StandardScaler
  model_algorithm k_means my_k_means(n_clusters 2, random_state 10)
  training_results "data/training.txt"
}
```

**Figure 6.11:** The data analytics part of the software model instance in the EMF tree-based editor.
Figure 6.12: The data analytics part of the software model instance that shows the black box ML mode in textual form.

Figure 6.13: The values of the loss function and the accuracy during the training of the ML model for validating RQ 2.
Figure 6.14: From PIM to PSMs, and the full generation of the IoT services for the target platforms. The images of Arduino and Raspberry Pi are from [12] and [13], respectively.

Figure 6.21: The tree-structured search space for the AutoML engine.
Figure 6.22: The overall architecture of the IoT service for the case study validating RQ 3.
Figure 6.23: The behavioral model of the energy disaggregation server of the smart grid in the IoT service for the case study validating RQ 3.
6 Implementation, Validation, and Evaluation

Figure 6.24: The disaggregation accuracy using different ML methods. Color codes: dark blue, orange, gray, yellow, and light blue represent the results for the fridge, lights, sockets, washer/dryer, and average for all appliances, respectively.

Figure 6.25: The gender distribution of the online survey participants.

Figure 6.26: The age group distribution of the online survey participants.
6.7 Discussion and threats to validity

Figure 6.27: The highest academic degree distribution of the online survey participants. 2.38% preferred not to disclose.

Figure 6.28: The education background distribution of the online survey participants.
Figure 6.29: The current job or occupation distribution of the online survey participants.
6.7 Discussion and threats to validity

Figure 6.30: The current role or position distribution of the online survey participants.

- Machine learning researcher scientist
- AI/ML researcher
- Student
- Community Manger
- CTO
- Researcher in Computer Science
- Software engineering researcher
- Software Tester
- Postdoc (so I have to do many engineering and data science tasks. Networks, security, and databases are not my concerns typically).
Figure 6.31: The familiarity of the online survey participants with the related fields/topics.

Figure 6.32: The expertise level distribution of the online survey participants in the DEA/ML fields.
6.7 Discussion and threats to validity

Figure 6.33: The expertise level distribution of the online survey participants in SE.

Figure 6.34: Answers of the online survey participants to question 1 in section 1 of the questionnaire. The best model kind and notation for analytics modeling, and the communication of data scientists with their peers.
Figure 6.35: Answers of the online survey participants to question 2 in section 1 of the questionnaire. The best model kind and notation for analytics operations, and the communication of data engineers with their peers.
6.7 Discussion and threats to validity

Figure 6.36: Answers of the online survey participants to question 3 in section 1 of the questionnaire. The best model kind and notation for describing the architecture from the viewpoint of a data scientist.
Figure 6.37: Answers of the online survey participants to question 4 in section 1 of the questionnaire. The best model kind and notation for describing the architecture from the viewpoint of a data engineer.
6.7 Discussion and threats to validity

Figure 6.38: Answers of the online survey participants to question 1 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with the end-users of software and information systems.
Figure 6.39: Answers of the online survey participants to question 2 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with the business stakeholders of software, information systems, and enterprises.
6.7 Discussion and threats to validity

Figure 6.40: Answers of the online survey participants to question 3 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with safety and regulatory compliance engineers, data protection (privacy) officers and ethics committees/boards.
Figure 6.41: Answers of the online survey participants to question 4 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with database engineers/designers.
6.7 Discussion and threats to validity

Figure 6.42: Answers of the online survey participants to question 5 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with software engineers and software architects.
Figure 6.43: Answers of the online survey participants to question 6 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with system engineers and network experts.
Figure 6.44: Answers of the online survey participants to question 7 in section 2 of the questionnaire. The best model kind and notation for the communication of data scientists and data engineers with security experts.
Figure 6.45: Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which end-users will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level.
6.7 Discussion and threats to validity

Figure 6.46: Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which ethics committees/boards will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level.
Figure 6.47: Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which data protection (privacy) officers will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level.
6.7 Discussion and threats to validity

Figure 6.48: Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which data engineers will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level.
Figure 6.49: Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which database engineers/designers will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level.
6.7 Discussion and threats to validity

Figure 6.50: Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which business stakeholders will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level.
Figure 6.51: Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which safety and regulatory compliance engineers will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level.
Figure 6.52: Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which data scientists (including Machine Learning engineers) will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level.
Figure 6.53: Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which software engineers and software architects will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level.
6.7 Discussion and threats to validity

**Figure 6.54:** Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which domain/knowledge engineers will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level.

**Figure 6.55:** Answers of the online survey participants to question 1 in section 3 of the questionnaire: The extent to which security experts will be affected by AI/ML/DEA aspects. 1 means no impact, and 5 means the highest level.
7 Conclusion and Outlook

7.1 Summary of contributions

In this doctoral dissertation, we have proposed a novel approach, called ML-Quadrat, that can enhance the software development process of smart services for the IoT and smart Cyber-Physical Systems (CPS). We have offered abstraction and automation for software engineering, as well as Data Analytics (DA) and Machine Learning (ML). The main innovation of this work has been combining ML models and software models to enable a model-driven software development methodology. We have implemented the proposed approach in an open-source prototype and validated the four Research Questions (RQs) that were stated in Section 4.1.

First, we have shown that a fully automated code generation and ML model generation for smart IoT services and CPS is feasible. We have added DAML concepts to the domain-specific modeling language of an existing state-of-the-art modeling language and tool for the domain of IoT/CPS, namely ThingML [4, 24, 25, 26]. Moreover, we have enabled a fully automated code generation in different languages, including Python and Java. The generated Python code was responsible for DAML and deployed the APIs of various open-source ML libraries and frameworks, such as Keras [27] with the TensorFlow [28] backend, and Scikit-Learn [29]. In addition, we have supported the so-called blackbox ML mode to let the practitioners choose any arbitrary ML model family/architecture, and bring their pre-trained ML models to connect them to the software models. This way, they have not been restricted to the limited set of ML methods that we have supported out-of-the-box in ML-Quadrat [3, 5, 14, 15, 16, 17, 18].

Second, we have illustrated how abstracting from platform-specific details in order to separate the Platform-Independent Model (PIM) and the Platform-Specific Model (PSM) layers, similar to the core idea of the MDA [31, 30] layers, is deployable. To realize this, we implemented several PSMs that extended one single PIM. We generated different implementations, including the source code in different programming languages and ML models, automatically out of the PSMs. For instance, we showed how TinyML
Conclusion and Outlook

on IoT edge devices could be supported by one of the PSMs. One of the advantages of this separation of the PIM and PSM layers of abstraction should be the possibility for the practitioners and stakeholders to hide the technical details of the underlying platforms and focus on the business logic of the software on the PIM layer.

Further, we have enabled automated ML to assist practitioners who were software developers without deep knowledge of DAML in choosing the most appropriate ML model families/architectures for their specific use cases, and having their ML model hyperparameters optimized automatically. This way, not only novice users in DAML could easily use our approach, but also DAML experts could avoid their usual trial-and-error practices, thus ensuring the best quality and performance for their ML models.

Finally, we have proposed enabling AI/ML in architecture frameworks for software, systems, and enterprises. To this aim, we have devised a new conceptual reference model for DEA/ML, and used this model, as well as the insights and information gained in our expert interviews and the online survey to identify new stakeholders, and propose new viewpoints, views, as well as model kinds, formalisms, or notations that were related to the AI/ML-concerns. This has initiated a new research direction at the intersection of AI and SSE.

7.2 Limitations

The proposed approach and our implementation naturally have a number of limitations. First, it is clear that only the supported target platforms for which model transformations have been developed can be chosen. For example, there already exists a model-to-text/code transformation (i.e., code generator) for POSIX C, Arduino, Java, and Python, but there is no option for .NET. Similarly, one can get support for the TensorFlow [28] (through Keras [27]), TensorFlow Lite [140], TensorFlow Lite for Microcontrollers [141], and Scikit-Learn libraries and frameworks for ML, whereas other options, such as Core ML [79] or the Microsoft Cognitive Toolkit [80] (previously known as CNTK) are not supported out-of-the-box.

Fortunately, the above-mentioned limitation can be handled by anyone who has expertise in the target IoT/ML platform given the open-source nature of our implemented prototype and the provided documentation. This is a major difference of open-source tools, e.g., based on the Eclipse Modeling Framework (EMF) ecosystem, as opposed to the proprietary tools, such as MATLAB Simulink by MathWorks. Although the latter is widely used in the industry, its modeling language is fixed and can only be extended
by the tool vendor. In contrast, one can easily extend our modeling language grammar, as well as our model transformation framework. Further, since we released our prototype under a permissive open-source license (namely, the Apache 2.0 license), derivatives and modified versions are also authorized to be used in closed-source and proprietary projects. This is expected to maximize the impact, dissemination, and exploitation capabilities of the research and development work.

Moreover, the number of ML methods, model families/architectures, algorithms, and techniques that are supported out-of-the-box is limited. Even within one family, e.g., deep ANNs, it was not possible to support every architecture. Therefore, one would need to either use the provided blackbox ML mode, to bring one’s pre-trained ML model, or extend the modeling language grammar, as well as the model transformation framework (e.g., extend a particular code generator) to enable a new ML method, such as Gaussian Process (GP), Graph Neural Network (GNN), or Bayesian Deep Learning (BDL).

Additionally, generating code is only one aspect of model-driven approaches to SE. Other benefits and aspects include but are not limited to simulation, testing, as well as model-checking and formal verification. The present work has not addressed these aspects.

Further, the focus in this doctoral dissertation was mostly on DAML, and there were less efforts for the Data Engineering (DE) side. Also, for the DAML and DE that were supported, batch processing (i.e., offline learning) was considered. However, stream processing (i.e., online learning) is also required in many IoT/CPS use case scenarios of today. Last but not least, distributed learning lies out of the scope of this work.

Finally, one should be cautious about the general limitations of ML and data science. For instance, ML should be only used if a problem cannot be solved by devising an algorithm that tackles the problem at hand directly, instead of inferring a solution in a data-driven manner as ML does. This principle was, for example, pointed out by Leskovec et al. [49] in their book, where they referred to the failure case of a company, which attempted to use ML to locate resumes/CVs on the web. However, they were not able to do better than non-ML-algorithms to look for some of the obvious terms that appear in a typical resume or CV. In fact, ML could perform well “when we have little idea of what the data says about the problem we are trying to solve. For example, it is rather unclear what it is about movies that makes certain movie-goers like or dislike it” [49]. Similarly, data science has its own limitations and pitfalls. For instance, in discovering unusual events (anomalies) hidden within massive amounts of data, one may
often detect normal patterns as anomalies, thus leading to false positives, and possibly unnecessary costs and inconveniences as a result of this [49].

### 7.3 Future work

One obvious direction for the future work will be addressing the aforementioned technical limitations. For instance, one may extend the open-source prototype to support new target platforms, or enable new ML approaches out-of-the-box. Moreover, integrating the provided tool prototype with other tools, for example, with SMT solvers for model-checking and formal verification features can be another possible future contribution. As mentioned, model-based testing and simulation possibilities at the modeling level are also expected to be useful and interesting for the users.

In addition, developing new model-to-model transformations for software models, ML models, and in particular for our combined models (i.e., enhanced ML-enabled software models), for example, to support the DAML Model-Interchange Formats (MIF), such as ONNX [76] will be beneficial.

Further, future work is required to address the DE aspects. In particular, scalable data processing for both batch processing (offline learning) and stream processing (online learning) can be supported, for example, by integrating the proposed approach with the Apache Hadoop and the Apache Spark ecosystems. In addition, distributed learning approaches, in particular, the privacy-preserving-by-design distributed ML approaches, such as Federated Learning (FL) are expected to become an imminent need given the current regulatory and societal concerns with regard to data protection and privacy.

Also, as mentioned in Section 5.3, we implemented one specific variant of the proposed approach that concerns letting the behavioral model of intelligent software systems become adaptive using ML models (see Equations 5.10 and 5.11). However, for future work, enabling the structure of such systems to become adaptive using ML models, as well as for both the behavior and the structure, must be explored.

Additionally, Pigem [157] studied how ML can be used to learn finite-state machines. Hence, there might be some potential in adopting such approaches and integrating them with the proposed approach to make the MDSE models even more intelligent. In fact, this would mean letting them learn the behavioral model of the software, in part or completely, on their own, using the existing data, instead of having the practitioner (i.e., the user of the DSML) specify it.
Moreover, in Section 5.4, we proposed a new research direction at the intersection of AI and SEE for enabling AI/ML in architecture frameworks. We took the first step in this new field by proposing a conceptual reference model for DEA/ML, as well as new identified stakeholders with AI/ML-concerns, and new viewpoints, views, and model kinds to frame and address their concerns. However, further research will be required to extend the proposed reference model, create reference architectures based on this, and augment existing and new architecture frameworks of software, systems, and enterprises with the proposed and new stakeholders, architectural viewpoints/views, and model kinds.

Furthermore, quantum information, computation, and communication technologies are currently revolutionizing many applications. “In October 2019, Arute et al. [158] published the breakthrough results of their research and development at Google AI Quantum on Quantum Computing (QC), where they reported the total runtime of only 200 seconds for a specific computational task on their programmable superconducting quantum processor, which would take approximately 10,000 years on a state-of-the-art classical (i.e., non-quantum) supercomputer. This groundbreaking experiment validated the previously hypothetical and theoretical vision of quantum supremacy, initiated by Richard Feynman in the 1980s. Today, it is a realistic expectation that QC will eventually be a revolutionary and disruptive enabler technology in the upcoming years and decades in various domains and disciplines” [34]. “However, current software applications will not abruptly disappear or become useless. It is likely that for a relatively long period of time, we will have to deal with a mixture of classical and quantum computers, algorithms, and data. Hence, the current challenges introduced by pervasive, distributed computing, which intrinsically involve heterogeneity, will even become more crucial, due to the new hardware technologies and architectures, which will require a fundamentally different paradigm and model for the programming and execution” [34]. This will bring new opportunities for MDE, including the proposed approach ML-Quadrat. Hence, one can extend them in order to support hybrid quantum-classical applications, where a mixture of heterogeneous CPUs, GPUs, Application-Specific Integrated Circuits (ASICs), such as Tensor Processing Units (TPUs), as well as Quantum Processing Units (QPUs) with various architectures will be deployed and can be used to execute distributed IoT/CPS services.

Finally, we presented the criteria of merit for evaluating the proposed approach and the resulting software systems. Validating the proposed criteria, e.g., empirically through
expert interviews/surveys, as well as evaluating the proposed approach and the resulting systems in the case studies using these criteria remain as future work.
Bibliography


BIBLIOGRAPHY


[27] F. Chollet et al. Keras. https://keras.io, 2015.


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


