Model-driven digital thread platform for cyber-physical systems

Hafiz Ahmad Awais Chaudhary

Publication date

31-12-2023

Document Version
1

Citation for this work (HarvardUL)

Chaudhary, H.A.A. (2023) 'Model-driven digital thread platform for cyber-physical systems', available: https://doi.org/10.34961/researchrepository-ul.24982158.v1.

Model-driven Digital Thread Platform for Cyber-Physical Systems

Hafiz Ahmad Awais Chaudhary

CONFIRM Centre for Smart Manufacturing

Department of Computer Science and Information Systems

Faculty of Science and Engineering

University of Limerick, Ireland

Supervised by:

Prof. Dr.-ing Tiziana Margaria

Submitted to the University of Limerick for the degree of

PhilosophiæDoctor (PhD) 2023

Supervisor: Prof. Dr.-ing Tiziana Margaria
Chair of Software Systems
Department of Computer Science and Information Systems
University *of* Limerick
Ireland

Internal Examiner: Dr. Nikola Nikolov
Department of Computer Science and Information Systems
University *of* Limerick
Ireland

External Examiner: Prof. Dr. Mihhail Matskin
Division of Software and Computer Systems
KTH Royal Institute of Technology
Sweden

Chair: Dr. Tabea De Wille
Department of Computer Science and Information Systems
University *of* Limerick
Ireland

Day of the defence: 7th November 2023

Signature from the head of PhD committee:

11/13/2023

DocuSigned by:
105E92CD29D44B9...

Declaration

I hereby declare that I have written this thesis without the prohibited assistance of third parties and without utilizing any aids other than those specified. Any ideas or information obtained directly or indirectly from external sources have been duly acknowledged. This thesis has not been previously submitted in an identical or similar form to any Irish or foreign examination board.

The thesis work was carried out under the supervision of Prof. Tiziana Margaria at the University of Limerick from 2020 to 2023.

Hafiz Ahmad Awais Chaudhary
Limerick, 2023

Acknowledgements

I would like to express my utmost gratitude to my supervisor Tiziana Margaria for her continuous support, invaluable guidance, and expertise, and for providing me with the opportunity to be a part of her research group. Throughout this research journey, she ensured that we could innovate as a team, was always open to new ideas, and ensured we pursued a common big picture. She also always managed her time to make availability for us whenever we requested.

I want to express my sincere appreciation to the examiners of my thesis, Mihhail Matskin and Nikola Nikolov, as well as to Tabea De Wille for chairing my doctoral committee.

I am also particularly grateful to Bernhard Steffen and his team at Technical University Dortmund for their research support for the underlying platforms. They consistently provided DevOps support and resources while we conducted public events here in Ireland. Their support has been instrumental in the successful completion of this research.

This work would not have been possible without the support of Science Foundation Ireland and the CONFIRM Centre for Smart Manufacturing, who provided funding and resources for this research project. Their contributions alleviated the financial burden and allowed me to focus on the completion of this thesis.

Many thanks go to my colleagues from the SCCE research group and the University who have supported me in projects, research, and intellectual discussions outside the classroom. Their friendship, encouragement, and laughter have made this academic journey memorable and enjoyable. I extend my sincere appreciation to the staff and administrators from different departments of UL, whose behind the scenes efforts provided me with an uninterrupted research environment, access to resources, and administrative support.

I want to express my sincere gratitude to Dirk Pesch from CONFIRM Centre, University College Cork, and Jobish John from Johnson & Johnson, for their collaboration and contributions in validating the industrial aspects of the research.

Last but not least, I am grateful to my family for their unconditional love, understanding during this challenging journey, and constant encouragement. I could not have completed this without their love and support. I would also like to express my gratitude to my father and grandfather, whom I lost during this journey. Their unwavering support and encouragement have been the cornerstone of my success.

This has been extremely inspiring; thank you all for that!

To my father Prof. Awais Latif (late).

Abstract

The latest advancements in Cyber-Physical Systems revolve around the increased adoption of cutting edge technologies, such as the Internet of Things, edge computing, artificial intelligence, machine learning and data analytics. The goal is to enhance real-time interaction between digital and physical entities and enhance their predictive capabilities. The challenge of addressing requirements related to integration and achieving interoperability within the space of Industry 4.0 requires a platform approach to address the heterogeneity in the system of systems, storage capabilities, devices and the ecosystem involved. Conventional approaches are limited in their scope covering only the individual aspects and often rely on bespoke glue code that is difficult to produce and even harder to maintain. This can result in poor code quality with an overhead of increased maintenance requirements, e.g., resources, time and cost, and reduction in agility. To overcome these challenges, a model-driven Digital Thread Platform for CPS is proposed that offers a baseline architecture and systematizes the integration methodology through a layered Domain Specific Languages approach. This approach covers the integration space domain by domain, technology by technology and platform by platform; leading towards a generalizable and highly reusable solution. Altogether, this work presents the successful integration of 19 independent technologies in two distinct Low-code development environments with the production of 26 processes and pipelines and the definition of 62 SIBs in 25 distinct DSLs. The effectiveness of the proposed approach has been demonstrated through use cases, examples, and applications across multiple domains. The proposed platform brings cohesiveness to the involved heterogeneous systems and platforms, making them easier to understand, develop, and maintain. Furthermore, it also empowers domain experts to actively participate in the development cycle. There has been a significant reduction in both the new integrations and the development efforts, and over time, the extent of reuse will continue to grow, particularly for the most standard applications.

Contents

CONTENTS

List of Tables

LIST OF TABLES

List of Figures

LIST OF FIGURES

LIST OF FIGURES

List of Abbreviations

AD-DSLs Application Domain - Domain Specific Languages.

AI Artificial Intelligence.

API Application Programming Interface.

CPS Cyber-Physical Systems.

DIME Domain-specific Integrated Modeling Environment.

DSL Domain-Specific Language.

DSR Design Science Research.

DTP Digital Thread Platform.

GUI Graphical User Interface.

HTTP Hypertext Transfer Protocol.

I4.0 Industry 4.0.

ICT Information and Communications Technology.

IDE Integrated Development Environment.

IIoT Industrial Internet of Things.

IME Integrated Modeling Environment.

IoT Internet of Things.

List of Abbreviations

ISE Immersive Software Engineering.

IT Information Technology.

jABC Java Application Building Center.

JSON JavaScript Object Notation.

LCDE Low Code Development Environment.

LDE Language-Driven Engineering.

MDD Model-Driven Development.

MDE Model-Driven Engineering.

ML Machine Learning.

PL-DSLs Programming Level - Domain Specific Languages.

R@ISE Research at Immersive Software Engineering.

RQ Research Question.

SFI Science Foundation Ireland.

SIB Service-Independent Building Block.

SLG Service Logic Graph.

TRL Technology Readiness Levels.

URL Uniform Resource Locator.

Presentation of Thesis

This thesis has been prepared in the style of "Thesis by Publication". Table 1 below includes a list of published papers that are integral to this dissertation. Selected publications are presented in Part II of the thesis and are labelled as AP1, AP2, etc. The papers labelled NA1 and NA2 are referenced and discussed within the thesis but not included as attachments in Part II. The provided labels will be used for citation later in the thesis. Considering the different bibliography styles of papers from various journals and publications, the individual bibliographies for each paper in Part II are not immediately attached at the end of each paper. Instead, all of the bibliographies are presented uniformly at the end of the thesis.

Presentation of Thesis

Label	Paper
AP1	**H. A. A. Chaudhary** and T. Margaria, "Integration of micro-services as components in modeling environments for low code development," Proceedings of the Institute for System Programming of the RAS, vol. 33, no. 4, 2021.
NA1	**H. A. A. Chaudhary** and T. Margaria, "Integrating External Services in DIME," in ISoLA 2021, Leveraging Applications of Formal Methods, Verification and Validation, Cham: Springer International Publishing, 2021, pp. 41–54.
AP2	T. Margaria, **H. A. A. Chaudhary**, I. Guevara, S. Ryan, and A. Schieweck, "The Interoperability Challenge: Building a Model-driven Digital Thread Platform for CPS," in ISoLA 2021, Leveraging Applications of Formal Methods, Verification and Validation, Cham: Springer International Publishing, 2021, pp. 393–413.
AP3	**H. A. A. Chaudhary** and T. Margaria, "DSL-based Interoperability and Integration in the Smart Manufacturing Digital Thread," Electronic Communications of the EASST, vol. 81, 2022.
NA2	**H. A. A. Chaudhary**, I. Guevara, J. John, A. Singh, T. Margaria, and D. Pesch, "Low-code Internet of Things Application Development for Edge Analytics," in Internet of Things. IoT through a Multi-disciplinary Perspective, L. M. Camarinha-Matos, L. Ribeiro, and L. Strous, Eds. Cham: Springer International Publishing, 2022, pp. 293–312.
AP4	**H. A. A. Chaudhary**, I. Guevara, A. Singh, A. Schieweck, J. John, T. Margaria and D. Pesch, "Efficient Model-Driven Prototyping for Edge Analytics," Electronics, vol. 12 (18), 2023.
NA3	I. Guevara, **H. A. A. Chaudhary**, and T. Margaria, "Model-driven edge analytics: Practical use cases in Smart Manufacturing," in ISoLA 2022, Leveraging Applications of Formal Methods, Verification and Validation. Practice, Cham: Springer Nature Switzerland, 2022, pp. 406–421.
AP5	**H. A. A. Chaudhary**, I. Guevara, J. John, A. Singh, A. Ghosal, D. Pesch, and T. Margaria, "Model-driven Engineering in Digital Thread Platforms: A Practical Use Case and Future Challenges," in ISoLA 2022, Leveraging Applications of Formal Methods, Verification and Validation. Practice, Cham: Springer Nature Switzerland, 2022, pp. 195–207.

Table 1: Research publications

Part I

Introduction

1

Introduction

1.1 Challenges in CPS

The manufacturing industry in the context of Industry 4.0 (I4.0) is experiencing a paradigm shift towards smarter, connected and intelligent process systems [6]. This transformation is driven by the rapid advancements in digital technologies, redefining production processes and business models, while fostering a transition towards more sustainable and environmentally friendly manufacturing approaches through resource optimization and waste reduction [7, 8]. At the heart of this transformation, the digital twin and digital thread play a key role, where the digital twin represents a virtual replica of any physical entity, asset or process, allowing manufacturers to simulate and optimize their operations. The digital thread, on the other hand, is a data-driven approach that handles the seamless connectivity and flow of information throughout the various stages of the manufacturing lifecycle. Cyber-Physical Systems (CPS) play a critical role in shaping the future of factories [9], as they provide the essential industrial integrations of interconnected and heterogeneous virtual and physical units of a complex system [10]. Interoperability in CPS is another critical aspect that enables seamless connectivity, communication and interaction between various data sources, components and subsystems from various vendors within a smart manufacturing environment [11] Achieving interoperability within CPS among vertical and horizontal integrations is a key challenge for the success of I4.0 and requires the adoption of standard communication protocols, data formats, and interfaces to

ensure the effective coordination and data interpretation between different devices, software, and networks [12].

In this context, the high-level depiction of the *reprogrammable factory* vision of CONFIRM research centre [13] is shown in Fig. 1.1, where Digital Thread as a collection of blue lines (solid or dotted) manages the communication and interoperation between the *Business layer* (at the top), *Application layer* (in the middle), and the devices, machines and more at the *Physical layer*. The application layer itself may include platforms (e.g., EdgeX foundry, ROS), Digital Twins (individual components), and software for analytics, Artificial Intelligence (AI) and Machine Learning (ML), and more.

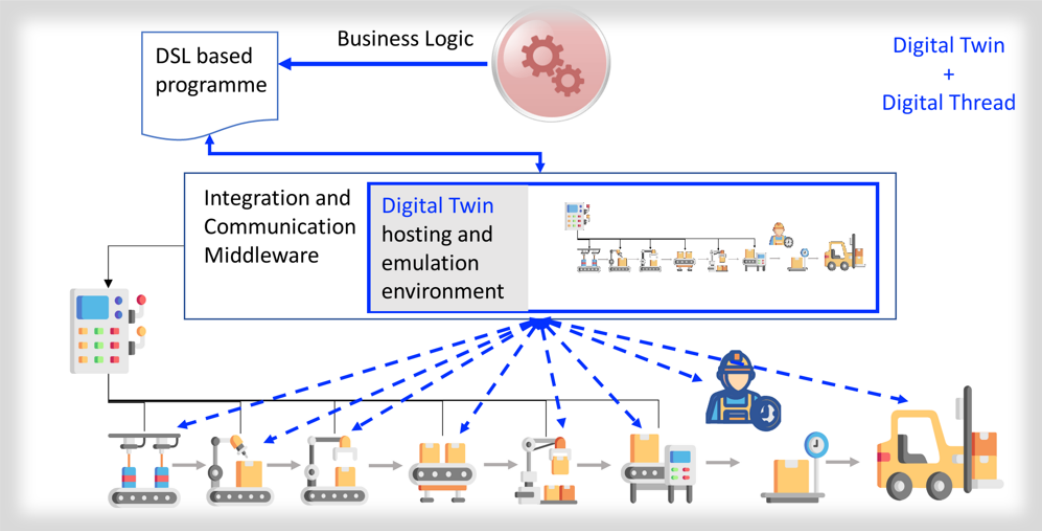


Figure 1.1: CONFIRM Hub2 CPS – the reprogrammable factory vision (source: CONFIRM Hub2).

The effort of building an interoperable system remains a significant challenge in the adoption of state-of-the-art technology. The task of integration is often overlooked and left for developers to address it as a secondary concern. Consequently, a number of experts are required again and again to reprogram the applications to address the evolving requirements and to align with the latest standards, which is both costly and time-consuming [14]. Additionally, the repeated integrations done manually with the Application Programming Interfaces (APIs) and devices reduce agility and require higher maintenance delays [15].

As CPS through interfaces is typically embedded into bigger and more complex systems, hence autonomy, reusability and modularity become vital factors for effectively adapting the upgrades and re-configurations to address the evolving requirements of the clients [16]. Considering the crucial role of the digital thread, the integration and interoperability layer should not be implemented through short-term scripting solutions that may be sufficient to address the point-specific interfacing requirements but make it difficult to comprehend, test, validate, manage, and evolve. The construction of meta-models behind these Domain-Specific Languages (DSLs) is challenging since they must capture all the domain knowledge, i.e., provide semantic and syntactic rules. DSL, both textual and graphical pave the way for new possibilities in terms of reusability, optimization, transformation, and even formal verification, which would be much harder to achieve otherwise [17]. The domain of Smart Manufacturing and CPS is captured as a part of the surrounding ecosystem, the available domain experts and due to funding agency requirements, i.e., CONFIRM Centre for Smart Manufacturing.

1.1.1 CONFIRM Context

CONFIRM[1] Centre for Smart Manufacturing is a Science Foundation Ireland (SFI) national research centre headquartered at the University of Limerick. The centre aims to harness rapid technological advancements, such as cloud computing, artificial intelligence, machine learning, advanced materials, additive manufacturing, and robotics, to enable efficient and innovative smart manufacturing. Smart manufacturing is centred around connectivity, data collection, supply chain integration, analysis, and decision-making, ushering in the next industrial revolution aimed at improving productivity, the economy, and overall quality of life while promoting sustainability.

In its current phase, the centre focused on contributions to fundamental and applied research towards establishing the infrastructure and demonstration facilities for technological advances and research in smart manufacturing. The thematic research areas within the centre are organized into four hubs as shown in

[1] https://confirm.ie/

1. INTRODUCTION

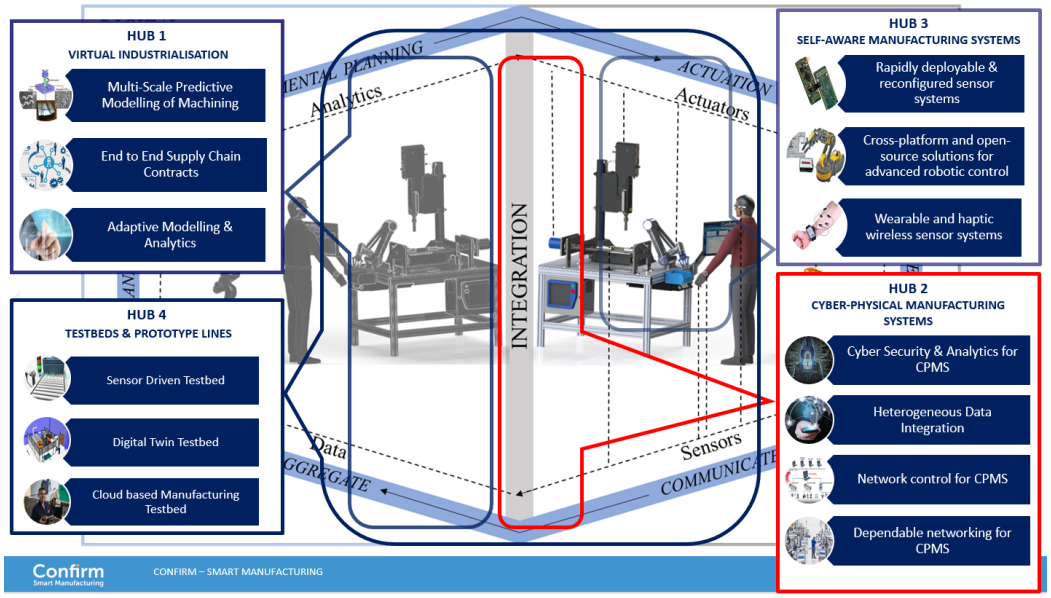


Figure 1.2: Research theme hubs in CONFIRM (source: CONFIRM).

figure 1.2; virtual industrialization, cyber-physical manufacturing systems, self-aware manufacturing systems, and testbeds and prototype lines. The physical infrastructure includes a variety of manufacturing testbeds and experimentation setups including virtual reality, extended reality, 5G mobile communications etc, with applications that focus on increased automation, intelligent control of planning, optimized supply chains, improved traceability and new delivery platforms.

We being part of the team of Hub 2 (i.e., Cyber-Physical Systems) are responsible for the research and development of an inter-operable platform that connects a wide range of heterogeneous technologies that serve the needs of CONFIRM users across different hubs. The focus was to showcase a platform at Technology Readiness Levels (TRL) 2 to 4 and aim at developing transferable solutions that are applicable to a wide range of smart manufacturing use cases. We selected DIME and Pyrus as the underlying Information Technology (IT) platforms, for which we provide the integrations and orchestrations in a formal methods-based, low-code and graphical model-driven fashion, to serve the needs of the applications addressed: manufacturing analytics, edge computing, edge analytics, and predictive maintenance are in fact core capabilities for the success of smart manufacturing operations. The generic baseline services, platforms and

devices, e.g., REST, R, Pycom etc are prepared and deployed for the purpose of integrations and tested during the development process. However, the renowned Internet of Things (IoT) related platforms and protocols, i.e., EdgeX foundry, MQTT etc were deployed using the standard images and then integrated and orchestrated into the platform. The applied case studies were designed in collaboration with the CONFIRM Centre at University College Cork and accordingly, Service-Independent Building Blocks (SIBs) and DSLs contributed iteratively to the platform service by service, domain by domain and platform by platform.

1.1.2 Research Questions

As stated in the original proposal for the CONFIRM centre and following discussions with the stakeholders and CONFIRM partners, the main requirement of the three dissertations it covers is "the proposition of a modern Digital Thread Platform for Cyber-Physical Systems in the context of smart manufacturing with an ability to support
a) connections with different technologies and peripherals b) seamless communication among heterogeneous services and platforms within the system c) heterogeneous local and cloud data storage platforms d) flexible to vertical and horizontal scalability e) a fast turnaround time by embracing agility and a DevOps methodology, and f) the empowerment of stakeholders, among whom only a few are professional coders."

In order to address these requirements, the following Research Questions (RQ) were posed.

- **RQ1:** Is it feasible to integrate (micro-)services as Application Domain - Domain Specific Languages (AD-DSLs) through the provision of domain-specific graphical SIBs?
- **RQ2:** How can the Digital Thread Platform address the challenges of interoperability within a complex system between various peripherals, platforms and technologies?
- **RQ3:** How can the Digital Thread Platform be applied to real-world industrial use cases efficiently in Cyber-Physical Systems?

- **RQ4:** How can the Digital Thread Platform with the support of Graphical DSL in an MDD paradigm, empower stakeholders to effectively utilize the platform's capabilities for smart manufacturing and expedite the turnaround time in an agile environment?

1.2 Proposed Solution

Language-Driven Engineering (LDE) [18] ecosystems represent a promising approach to addressing the complexity and diversity of modern software and systems development. These ecosystems emphasize the use of Domain-Specific Languages (DSLs) to describe various aspects of a system, facilitating the communication and collaboration among different stakeholders [19]. By combining framework thinking with a low-code approach, it becomes possible to systematically and rapidly generate application-specific complex objects from collections of reusable components. LDE fosters the creation of models and their abstractions that are tailored to address the specific needs and requirements of a certain domain [20]. Such ecosystems typically rely on Model-Driven Engineering (MDE) techniques to enable the seamless integration of models and code, promoting reusability and agility in system development [21]. Model-Driven Development (MDD) advocates the development of complex systems based on models and their refinement from the conceptual modeling stage to the automated code generation phase [22]. The primary objective of Model-Driven Development is to develop adaptable, cost-efficient, and rapid applications with shorter maintenance cycles [23]. Following the LDE principles, Fig. 1.3 shows the construction of a sample application with a complex technology stack (Web application) using the supported modeling DSLs of underlying Integrated Modeling Environment (IME). This allows domain experts, who may not necessarily be developers, to build solutions from design to deployment using the DSL, eliminating the need to become proficient in the complex technology stack required for full application development.

To a certain extent, the combination of Low Code Development Environments (LCDEs), MDD and DSLs seems, therefore a winning strategy, but there is not yet a solution that supports the heterogeneity needed for the Digital Thread, the formality needed for the MDD-based rigour for high assurance software, and the

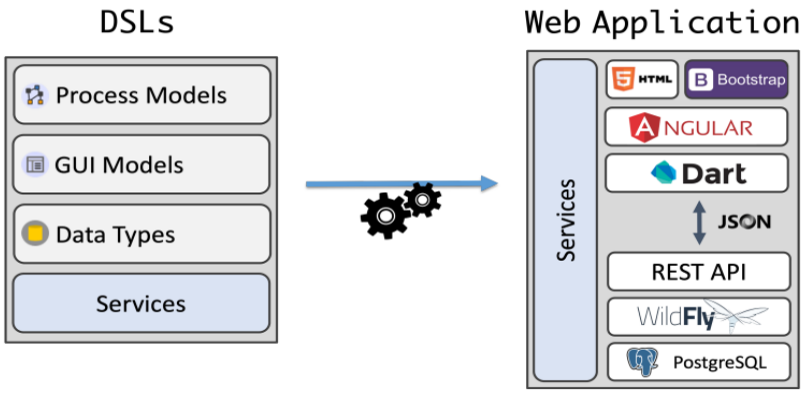


Figure 1.3: DSLs overcoming the heterogeneity of complex technology stacks (originally from [1]).

specificity embodied by LCDEs. Most domain-specific languages today are at the coding level and do not leverage a model-driven approach at the platform level. The rise in re-usability and maintainability demands paved the path to low code development environments and gained the attention of the developer community [24].

The choice of an underlying development platform is a challenging decision because the attributes of the Digital Thread Platform (DTP) will be deeply intertwined with the characteristics and features of the development platform, it is built upon and it is not something that can be readily altered or reversed at a later stage. Here, the adoption of LDE principles and DSL approach through the utilization of CINCO [25] and Pyro [26] products like DIME [2] and Pyrus [3] is a big advantage and concerning DSLs, we are keen adopters at both levels: language design DSLs as in DIME and application domain DSLs as External Native DSLs. These DSLs are valuable in addressing the knowledge, terminology, and concerns of all stakeholders within a collaborative (application) development team. Additionally, the direct source code modifications for every upgrade can be an expensive exercise and often exclude the domain experts from the development life cycle, who are the key stakeholders of the domain. This direct co-design approach is by far the most effective method for boosting productivity and reliability [27]. Therefore, the LDE paradigm combined with automated code generation and service orientation capabilities has been adopted.

1.2.1 Contributions Summary

In order to address the research questions discussed in sec. 1.1.2, this dissertation proposes a DTP for CPS, where we define the baseline architecture [11] and address the associated challenges of integration [28] and interoperability [11, 29] in the design, development and implementation of DTP. Specifically, we support the integration of external services from various (domain-specific) platforms and various programming languages in order to deliver their much simpler and better controlled interoperability. Series of integrations across various domains are validated through the implementation of multiple industrial case studies within smart manufacturing, i.e., predictive maintenance [30], edge computing [5], edge analytics [5, 31], safety operations [32], stable storage facility [33] etc. To support the fundamental operations in smart manufacturing, the platform offers a set of pre-built ready-to-use native, hierarchical and feature level DSLs [30] for systematic integrations. Table 1.1 shows the individual contributions of the dissertation against the proposed research questions with the list of published papers for further details.

Sr.	**Research Questions**	**Research Contributions**	**Publications**
1	RQ1	Integration of (micro-)services as graphical AD-DSLs See section 4.1.1	AP1 (Journal)
			NA1 (Conference)
2	RQ2 and RQ4	Interoperability challenge in Digital Thread Platform See section 4.1.2	AP2 (Conference)
			AP3 (Journal)
3	RQ3 and RQ4	Industrial use cases in Smart Manufacturing See section 4.1.3	NA2 (Conference)
			AP4 (Journal)
			NA3 (Conference)
			AP5 (Conference)

Table 1.1: Research contributions

1.3 Thesis Outline

The dissertation is primarily organised into three parts.

Part I covers the fundamental concepts of dissertation, where *chapter 2* covers the related work, *chapter 3* describes the design and implementation, *chapter 4* discusses the results and contributions and *chapter 5* concludes and discuss the future work.

Part II presents the selected papers published as a part of this dissertation.

Part III primarily is an appendix, encompassing the code for SIBs signatures and the backend Java implementation for an exemplary AD-DSLs related to R-programming and analytics capabilities. Additionally, this section includes information about the funding agency and details additional activities conducted during this dissertation, such as the design of a poster for conference participation.

Attached to this dissertation is a memory stick containing the following items:

- *folder 1*: Contains the resource files for this thesis
- *folder 2*: Contains the source code and project artefacts of the platform.
- *folder 3*: Contains the scripting files for the Docker containers to prepare the ready to use platforms.
- *folder 4*: Contains all the software used or integrated required for use during the source code and project artefacts of the platform.
- *folder 5*: Contains extra materials that are relevant to this research activity.

1. INTRODUCTION

2

Related Work

The Industry 4.0 revolution covers a broad range of key enabling technologies like the Internet of things, digital twins, IoT, AI/ML, AR/VR and big data analytics etc. [34]. The latest technological trends in I4.0 accelerated innovation in the space of smart manufacturing and CPS. The manufacturing industry under this I4.0 revolution pushed manufacturing companies to adopt and adapt data-driven strategies towards connected and smarter processes [35, 36], with the focus to meet the dynamic needs of "smart" factories and contribute towards sustainable manufacturing environments. The adoption and deployment of I4.0 demands vertical and horizontal integrations among physical and digital entities through the creation of smart solutions, which is the core of CPS [37]. The term CPS was coined by Helen Gill in 2006 to connect a physical reality with communication networks and computing resources [38].

2.1 Latest Trends in Cyber-Physical Systems

Cyber-Physical Systems are a driving force behind digital transformation in almost everywhere [39, 40]. Rapid advancements in disruptive technologies are not only reshaping industries but also redefining the roles of humans [41]. Traditional boundaries between humans and technology have significantly blurred, especially with the emergence of metaverse technologies and we are rapidly moving towards the concepts of Industry 5.0 [42, 43].

2. RELATED WORK

While it is evident that the growing use of these systems plays a role in advancing digitalization and fostering innovation, there are still uncertainties regarding how to effectively tackle the challenges posed by the unique characteristics of these new systems. Future CPS, such as smart cities, intelligent fleets, autonomous systems, and more, will not simply be upgraded versions of today's systems, rather they are expected to exhibit different characteristics compared to the systems currently in existence. In this context, the concept of human-cyber-physical systems has garnered significant attention as an emerging research paradigm, and three key research areas have been identified based on the properties of these systems: intelligence, connectivity, and electrification.

Intelligence is one of the defining characteristics of future CPS. Intelligent systems are not only meant to possess the capability to collect and analyze data but are also expected to facilitate communication with other CPS, thus enabling automated decision making within interconnected networks of systems. These complex intelligent systems employ a wide range of technologies, such as artificial intelligence, data analytics, cybersecurity, deep learning, and distributed systems, among others. The attribute of intelligence enables systems to make decisions and take action autonomously in predetermined scenarios. Currently, autonomous and intelligent systems are prominent topics of research and education within the field of CPS. [44, 45].

Another new characteristic of future CPS is high *connectivity*. Connectivity, supported by the advancements of cloud computing, has paved the way for the growth of various technologies. CPS inherently encompasses networking capabilities, but in the future, this is anticipated to expand further, incorporating improved integration with interoperable platforms, and facilitating seamless communication among these systems. This requires new development approaches within CPS research and education to harness the full potential of this level of connectivity [45, 46].

Many initiatives are focusing on transitioning to decarbonized power systems and expanding *electrification* in transportation and building sectors to reduce dependence on fossil fuels. Electrification is viewed as an opportunity to transform the transportation sector, encompassing vehicles and infrastructure. The popularity of electric vehicles is on the rise and they are achieving record sales

annually, with this trend expected to continue. This shift is closely tied to advancements in battery technologies, powertrain designs, dedicated electric vehicle platforms, component integration, and the adoption of novel materials. However, the electrification of vehicles also presents new challenges concerning transport infrastructure and requires a fresh perspective and the possibility of redesigning integrated transportation systems [46].

In response to evolving technological landscapes and labour market dynamics, the reliance on interdisciplinary skills is crucial for inventing, designing, building, and deploying these systems effectively; hence, emerging engineers and scientists should possess the modern tools and mechanisms to adapt and collaborate across various disciplines of CPS [46, 47].

2.2 Efforts Towards Human-Centric Cyber Physical Systems

A recent concept known as Operator 5.0 [48] has emerged, defining a smart and skilled operator in the future of manufacturing, supported by state-of-the-art technologies to enhance human-technology connectivity. The goal of human-centric CPS is to augment human capabilities through intelligent interfaces and informed decision strategies, enabling the development of cost-effective solutions that ensure long-term sustainability. This necessitates transparency across various stages of the process, both in development and production, to optimize manufacturing objectives. In this context, the research [13] discussed the key role of the Digital Thread in Industry 4.0 to couple the data and processes and provide traceability in the entire lifecycle, to deliver integrated smarter ecosystems.

2.2.1 Advance Analytics

In order to enhance manufacturing productivity while minimizing maintenance costs, it is important to devise an intelligent strategy through central data dashboards that enable manufacturers to make decisions beforehand through predictive maintenance analytics [48]. The Era of Big Data [49] has emerged with the

advancement of cyber technologies with the accumulation of massive and heterogeneous data leading towards data-driven sustainable manufacturing frameworks for maintenance and prediction. The dynamic manufacturing environment facilitates the continuous updating, learning, and evaluation of big data analytic models [50]. The involvement of human factors contributes to the cognitive advancement of manufacturing systems. For example, a data-driven approach for opportunistic maintenance benefits from expert supervision in constantly updating hyperparameters, leading to improved noise reduction [51]. In this direction, several studies have been carried out to address the high demands of advanced analytics. The computational methods based on deep learning are presented to improve the system performance in manufacturing machines [48]. Recent research has explored the application of various neural network architectures to predict defects, estimate remaining useful life in mechanical systems, and tackle other machine health monitoring tasks [48]. In study [52], a competitive learning-based Recurrent Neural Networks was introduced for the long-term prognosis of rolling bearing health. In this context, the work presented in section 4.1.2 addresses the domain of advanced analytics supported by case studies in manufacturing analytics and predictive maintenance.

2.2.2 Edge Computing and Edge Analytics

The rapid advancement in sensing and storage technologies has led to the generation of vast amounts of data in I4.0 [53]. Traditional cloud computing, while powerful in terms of computation, can face difficulties in processing this amount of data due to limitations in bandwidth and the long distances between terminal devices (e.g., IoT sensors) and remote cloud servers. This can result in unacceptable communication delays, especially in applications requiring real-time responsiveness. To overcome these challenges, edge computing is proposed as an alternative solution [54].

Edge computing involves deploying computing resources (edge servers) closer to the source of data generation [12]. This approach reduces latency by processing data locally or at the edge of the network, minimizing the need to send data to distant cloud servers for analysis. Edge computing is characterized by its

distributed computing paradigm, which strategically places networking, control, computing, and storage resources between terminal devices (such as IoT devices) and remote cloud servers. This distribution allows for more efficient data processing and real-time service delivery. Edge servers can preprocess data, make localized decisions, and reduce the volume of data that needs to be transmitted to the cloud. Integrating edge computing or edge-cloud computing into the design and engineering of CPS can help reduce service latency, improve real-time responsiveness, and enhance the overall performance of CPS applications [54]. In this direction, several studies have been carried out to embed edge devices into CPS applications. One of the case studies [55] supported predictive intelligence modeling within the edge network, while [56] implemented a Lightweight Convolution Neural Networks in edge for Transportation CPS. GigaSight system [57] tackled the challenge of handling video streams by the enforcement of analytics at the edge node. In addition to addressing latency issues, edge computing also tackles challenges related to security, privacy, and reliability. Data can be processed and analyzed closer to its source, enhancing data security and privacy, and improving overall system reliability. In this context, the work presented in section 4.1.3 addresses the domain of edge computing and edge analytics supported by case studies in Stable Storage Facility and Safe Operation of Machines in both cloud-centered and edge environments.

2.2.3 Digital Twin and Digital Thread

The paradigm shift in manufacturing from traditional approaches to smart manufacturing requires in fact an end-to-end integration and interoperability at different stages of different business entities, processes and systems [58]. Integration plays a pivotal role in connecting, communicating, and fostering collaboration among various sub-systems, enabling them to operate as a unified entity. The progressive vertical and horizontal integration and connectivity in the manufacturing ecosystem contribute towards improved sustainability and better resource utilization so that companies can achieve higher industrial performance, shorter production cycles and customization on demand, ideally reaching a batch size of

one [59]. The interoperability among entities enables multiple entities to comprehend each other and harness the effective use of each other's capabilities within the system. The Institute of Electrical and Electronics Engineers (IEEE) defines interoperability as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged" [60]. Monitored and optimized manufacturing processes in smart factories demand increased interoperability between IoT devices and systems with advanced capabilities, like autonomous and smart configurations based on historical settings and architectures. Therefore, the development of integrated and interoperable frameworks, centered around smart factory scenarios, facilitates collaborative work and the achievement of cooperative objectives.

Digital Twin and Digital Thread are two transformational technological elements in digitalization of the Industry 4.0 [61]. The Digital Twin covers individual aspects of physical assets, i.e., their virtual representation, their environments and the data integrations required for their seamless operations. Digital Thread [62, 63] is a data-driven architecture analogy for an integration and interoperability layer with the ability to communicate and manage interoperation between business, application and physical layer. It provides a robust reference architecture to drive innovation, efficiency and traceability of any data, process and communication along the entire system as well as the Digital Twins and the devices, machines, sensors and dashboards, which may also include analytics, AI and ML capabilities, and more. Consequently, the Digital Thread requires seamless end-to-end integration of the data sources, processes and dashboards that cooperate with each other to deliver such complex and heterogeneous interconnected systems. In this context, the work presented in section 4.1.1 and section 4.1.2 addresses the challenge of integration and interoperability through a proposition of a modern DTP.

2.2.4 Domain Specific Languages

Domain-Specific Languages serve as a well-established method for bridging the semantic gap between the two entities, e.g., programmers and domain experts,

software and machines etc. Over the years, domain-specific languages both textual and graphical have gained significant popularity as an approach of choice for designing and developing heterogeneous systems [25, 64, 65]. Pyro [26] is a web-based graphical modelling environment for the collaborative development of web applications based on DSLs. Pyrus [3] is developed on top of Pyro and supports only data-flow DSLs for data analytics. Compared to Pyro, CINCO Cloud [66] integrates both the meta IME and the generated IME aspects into a single web application, providing a full LDE experience in the true essence of the cloud. The design of graphical DSLs in CINCO Cloud is based on the same principles as those used in CINCO. DATACLOUDDSL [67] supports both textual and graphical DSLs particularly designed for big data pipelines. Its resource provider package also allows the binding of tasks with resources like fog and edge and the tool also supports the multi-cloud deployment. Ktrain [68] is a popular coding level DSL: a Python wrapper that encapsulates Tensor Flow functionalities and facilitates developers to augment machine learning tasks with fewer lines of Python code.

The DSL approach is also widely adopted for the construction of CPS applications from models. In a study [69], the author proposed DSLs that implement Keyword-driven Testing methodology to create new test suites for self-diagnosis of CPS systems. Similarly, the author [70] explores the extensive use of DSLs in the field of robotics, while [71] designed the DSLs for the safe deployment of runtime monitors in CPS. The study [13] proposed that DSLs, specifically designed for the manufacturing domain, could drastically improve the traceability and validation of manufacturing systems [13]. By using a language tailored to a specific domain, users can better express and solve complex problems related to manufacturing processes.

In general, the development of DSLs may follow any of the available implementation approaches and may require a collection of guidelines, design patterns, model checking and common language design and implementation challenges for reusability [72]. A comparison of different implementation approaches of 40 DSLs from very different domains [73] concluded that the compiler-based approach (42.1%) is the most popular. In this perspective, we have developed a number of DSLs, as presented in section 3, targeting various domains of CPS.

2.2.5 Model Driven Development

Model-driven development with adequate models is an automated approach for the design and development of flexible and cost-effective applications rapidly by means of drag & drop of visual interfaces. Holistically, MDD covers aspects from conceptual modeling to model-to-code transformation [22]. In this line of thought, the Java Application Building Center (jABC) platform [21] proposed the design and development of applications based on formal models and its Lightweight Process Coordination. It accelerated the development process of applications through the concept of reusable building blocks, orchestrated into analyzable control structures called Service Logic Graph (SLG). The next generation tools of similar categories are DIME [2] and Pyrus [3]. DIME is a graphical modeling environment for the development of applications in a low-code manner. The process model pipelines in DIME support both the control flow and data-flow aspects of an application. Pyrus is also a graphical web modeling environment for the implementation of AI/ML pipelines in a data-flow-oriented fashion. Another work of a very similar category is DATACLOUDDSL [67], which supports graphical DSLs for the designing of big data pipelines. Following similar practices, several model-driven platforms were proposed for model checking [74], early bioinformatics applications [75, 76], fostering collaboration through tools interoperability within the Working Group on Formal Methods for Industrial Critical Systems [77, 78], to compose and combine heterogeneous planning algorithms in Plan-jETI [79, 80] and also mobile apps [81] and cross platform [82] applications.

MDE for Smart Manufacturing and CPS itself is a transformative paradigm in the industrial domain that emphasizes the use of mathematical and computational models to optimize production processes, enhancing efficiency, productivity, and sustainability. Barriga et al [83] describe an interesting case in this field of IoT involving various devices, processing nodes, hardware, and software. In the domain of integration of predictive models in smart manufacturing, the study [84] demonstrated how machine learning algorithms can provide critical insights to guide better decision-making. In a study [34], it is been demonstrated how model-driven approaches contribute to sustainable production and can help

manufacturers to minimize waste, reduce energy consumption, and optimize resource allocation. Similarly, model-driven modeling has also been applied to the development of distributed control systems for the automatic transformation of block-based design models to a component-model implementation [85]. Most of the industrial solutions do not support formal models and instead, they rely on the visual composition of the orchestrations, which is more intuitive than manual coding.

For the DTP, DIME [2] and Pyrus [3] were selected as underlying platforms, which are formal method based MDD products generated from CINCO and Pyro meta-models respectively. We see the graphical presentation of coordination languages, particularly in DIME and Pyrus as an advantage for those tasks that prioritize evidence and intuition. In this regard, visualizing workflows and dataflows in their native representation allows the detection of some errors in a more self-evident manner compared to deriving them from the linear syntax of conventional code.

Extraction of the control flow and data-flow graphs from the graphical representation is a common practice for analyzing dependencies and performing established program analysis and verification. These explicit, mathematically correct representations facilitate experts, as opposed to the alternative method of extracting them from traditional program code, where they are only implicitly present.

2.3 Interfacing With External Entities

In this rapidly evolving landscape of designing CPS, a number of approaches and platforms have emerged to tackle the complex challenges of heterogeneous connectivity in the ecosystem. However, it is evident that no single platform can claim to be a universal solution capable of addressing all the subtle needs. The reasons for this limitation are twofold: first, the continuous technological advancement in both hardware and software means that any static platform would swiftly become outdated; and second, the diverse range of applications and industries for which CPS are developed demands flexibility and adaptability.

In this dynamic environment, what emerges as of paramount importance is the ability of platforms to support extensions and seamless integration with external services and entities. Platforms that embrace extensibility become more than just tools; they evolve into ecosystems that can grow and adapt alongside the latest advancements. This approach empowers designers and engineers to leverage the cutting-edge technologies and services available as they emerge, ensuring that their CPS remain relevant, efficient, and secure in the face of evolving needs and challenges. Therefore, this is a crucial step forward that the CPS design platforms support that can thrive in our ever-changing technological landscape.

2.3.1 Component-Based Software Engineering

The emphasis on the separation of concerns has given rise to Component-based Software Engineering (CBSE) [86]. CBSE is centred around the decomposition of a software system into reusable and interchangeable components, which can be implemented using a variety of technologies and communicate via defined interfaces. While this approach greatly enhances reusability and maintainability, it also requires careful management to prevent unintended dependencies between components. Dependencies among components are often handled through libraries or frameworks, and any alterations to one component may potentially impact others reliant on it. Historically, CBSE has often followed a more monolithic architectural style, tightly integrating components into a single system. Nevertheless, it has provided enhanced control over the design, implementation, and evolution of software systems. In contrast, scaling a monolithic system often entails scaling the entire system, even if only a specific component requires additional resources, which can be inefficient in terms of resource utilization.

In earlier CBSE systems, the deployment of the entire system, inclusive of all components, typically occurred as a single unit. A classification framework [87] categorizes different available component models based on parameters such as interfacing, binding, integration, and communication. This classification framework has been influential in shaping various systems, particularly in domains like embedded systems and intelligent transportation. For instance, the work [88] introduced a family of component models for embedded system design in vehicles,

aimed at preventing wheel lock or spin during braking or acceleration. Another notable component model, ProCom [89], is designed for real-time distributed embedded systems and comprises two layers: ProSys, for modeling independent components, and ProSave, for subsystem design. Over the past decade, there has been a notable shift toward the "service-first" concept, which has naturally evolved into the microservices paradigm.

2.3.2 Microservices

Microservices is an architectural style that advocates breaking down a complex application into smaller, independently deployable services capable of communicating with each other via APIs [90]. This architectural style has been introduced to address the limitations of monolithic architecture and improve scalability, flexibility, and fault tolerance within distributed systems. The role of microservices in CPS is significant, and offers several advantages:

- *Modularity:* Microservices enable modularity in CPS design, allowing different functionalities to be encapsulated in separate services. This makes it easier to manage and maintain the system, as changes or updates to one service do not necessarily impact others.

- *Scalability:* In CPS, where the scale of operation can vary greatly, microservices allow for fine-grained scalability. Specific components can be scaled up or down as needed without affecting the entire system, improving resource utilization.

- *Flexibility:* CPS often needs to interact with various external entities and services, such as sensors, actuators, or cloud-based analytics. Microservices can be designed to handle these interactions efficiently, making it easier to integrate external services and adapt to changing requirements.

- *Fault Isolation:* Microservices are designed to be resilient. If one service encounters a fault or crashes, it does not necessarily bring down the entire CPS. This fault isolation ensures that the system can continue to function despite localized issues.

- *Continuous Deployment:* Microservices support continuous deployment and DevOps practices, allowing rapid updates and enhancements. This is crucial in CPS, where staying up-to-date with the latest technologies and security measures is vital.

- *Interoperability:* Microservices can be implemented using various programming languages and technologies. This enables CPS designers to choose the most suitable tools for specific tasks, promoting interoperability.

- *Resource Efficiency:* Since microservices are lightweight and can be deployed on resource-constrained devices, they are well-suited for the resource-constrained environments often encountered in CPS, such as IoT devices.

Different types of design patterns exist for the manual composition of services. However, the author [91] has proposed a novel approach for the automatic composition of services using program synthesis from an available repository of services. Similarly, another work proposed a Jolie based textual DSLs for services integration with the system [92] and discussed the mechanism of automated deployment of microservices in a cloud-based distributed environment [93].

2.3.3 Service Independent Building Blocks

A Service Independent Building Block is a graphical modeling component used in MDD that is executable, reusable, and represents an abstraction of an associated implementation. SIBs are integrated into the system by selecting them from a list of available SIBs palette and placing them onto the modeling canvas. This action triggers the creation of a node that represents the respective SIB. A SIB can range from a model as a placeholder for some functionality to a complete implementation of that functionality, including various levels of refinement and abstraction expressible within Java. Each SIB has a single entry point where execution begins and multiple exit points, referred to as branches, representing different outcomes at the model level.

The conceptual foundation for SIBs in DIME and Pyrus is rooted in jABC4 [21, 94], a framework for graphically modeling an application's business logic using

SLG. SIBs are organized into topologies known as SLG, which define process behaviour by connecting outgoing SIB branches to the entry points of other SIBs. Within an SLG, the execution of a SIB starts when one of its incoming branches becomes active, indicating that the preceding SIB governing that branch has completed its execution with an associated outcome. Some SIBs within an SLG can be designated as start SIBs, commencing execution without any incoming active branches, effectively serving as entry points for the modeled process. Consequently, an SLG serves as a graphical, executable process description that can be hierarchically organized using (graph) SIBs to manage complex process models.

2.3.4 Discussion

All of the discussed approaches can be opted for the implementation of flexible and adaptable systems depending upon the domain and the types of services required, e.g., central or decentralized etc. Considering the nature of the approaches, the central composition of the system under consideration can be done using component models, however, for the communication and extension with external entities, the microservices approach is suitable. In addition, the concept of terminologies can be domain and paradigm specific, i.e., SIB in model-driven development, while microservices and component models are more generic design patterns. The component models do not cover the operational aspects of the implemented services and are tightly coupled with the system in comparison to microservices [93].

The SIBs inherit the properties both from component models and microservices. At compile time the functional components (alike component model) with the abstractions of graphical nodes are available in the SIB palette for modeling, while the backend functionality is independent of SIB composition and supports the independent deployment irrespective of locality (more like microservices). The visual representation of SIBs as modeling components in the MDD paradigm is much closer to the standard *Unified Modeling Language* models. The study [95] did a systematic comparison of two approaches SCA (Service Component Architecture - primarily component oriented approach) [96] and jABC (the SIB oriented approach) [21] and in addition to their semantic differences, they

concluded that jABC is more operation centred and a better candidate when it comes to agility and reducing the semantic gap by putting the domain expert in the centre of the design process. In my work presented in section 4.1.1, this SIB approach is adopted to extend the functionalities of two of the platforms in a model-driven way.

2.4 Background

The philosophy behind the proposition of design and development of the DTP is based upon the principles of LDE with DSLs emphasis on the creation of models and their abstractions that are tailored to address the specific needs and requirements of a certain domain.

This section briefly discuss the underlying platforms, the individual sensors, devices, services, tools and technologies that form the heterogeneous ecosystem involved in this case study.

2.4.1 SCCE Meta-Tooling Frameworks - CINCO & Pyro

CINCO [25] and Pyro [26] are language workbenches designed for the development of domain-specific graphical IMEs using high-level specifications. Pyro was originally intended to be an alternative generator and graphical editor for CINCO. However, over time Pyro moved away from CINCO and gained more and different functionality. As a result, Pyro should be viewed as an alternative language workbench, which generates graphical modeling editors available on the web, rather than in Eclipse as it was for CINCO. The high level specifications in both of the workbenches are defined in textual DSLs:

- *Meta Graph Language* for specifying structural information of supported models and attributes.
- *Meta Style Language* for defining visual characteristics of nodes, edges and their customization.
- *CINCO Product Definition* is a CINCO only DSL for specifying features and tool generation details such as branding and naming.

These languages are built with EMF's Xtext framework [97] and offer a generative approach at runtime based on the defined APIs during tool development. This model API serves as the foundation for developing code generators that consume models designed with the CINCO and Pyro products, enabling the creation of complete IMEs. Most of the generated products from these workbenches follow the principles of One Thing Approach [98] and eXtreme Model-Driven Design [99] and place the application expert, who may not necessarily be a programmer, at the heart of the development process. These workbenches and the products they generate cover a variety of models as compared to their predecessors METAFrame [100] and jABC [21] platforms that only support process models, and even in DyWA [101] the integration between the data model and process models happened through import/export across two tools. Some of the advanced graph model editors DIME [2], Pyrus [3], Addlib [102] and Rig [103] are example products sharing a common philosophy and semantic and syntactic characteristics with their respective models.

In summary, CINCO and Pyro simplify the development of domain-specific graphical modeling tools by providing a generative approach, reducing complexity, and offering powerful modeling languages.

2.4.2 The Underlying Low-Code Development Environment - DIME

DIME [2] is an Eclipse based graphical modeling environment for the development of applications in a low-code manner. Its comprehensive development environment supports the entire application development cycle – from design and development to deployment – in an agile way. Graphical modeling in DIME is based on modular drag-and-drop components that also empower domain experts (potentially non-programmers/coders) to participate fully in the entire development process. Fig. 2.1 shows the interface of DIME with an exemplary arrangement of views:

- *Project Explorer* in DIME helps users manage and navigate through the files and resources associated with a project.

2. RELATED WORK

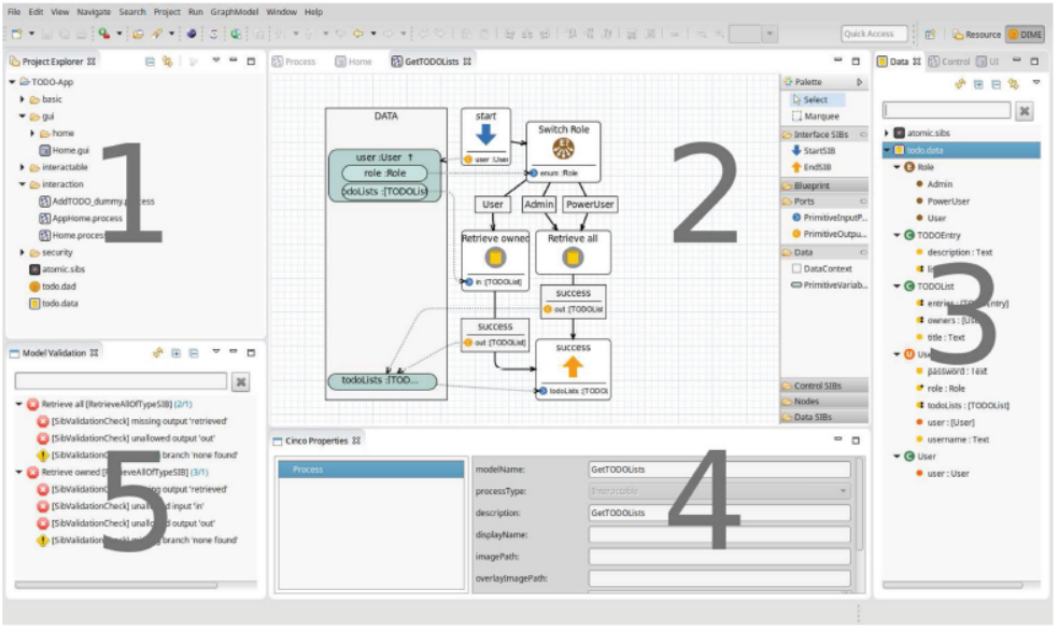


Figure 2.1: User interface of DIME: (1) Project explorer. (2) Diagram editor with the built-in component palette. (3) Model component views. (4) Properties view. (5) Model validation view. - originally from [2].

- *Diagram Editor* in DIME serves as the canvas for creating/editing and manipulating graphical models in a pipeline with the help of SIBs available in the palette section. To create a new node, simply drag and drop a SIB from the palette onto the canvas. The actions like modifying or deleting a SIB can be performed by hovering over the SIB. Every SIB has an associated predefined set of rules in regard to connectivity with other nodes via control or data flow. There is a possibility to open multiple models concurrently in separate tabs, but only one of the models can be active at a time.

- *Model Component Views* displays all data models and a list of DSLs with a set of associated SIBs within the scope of the current project. Users can drag and drop these SIBs onto the canvas to model their workflows in various contexts.

- *Properties View* displays both attributes and their current values for the currently selected component and dynamically updates its content to match

the user's selection. The properties view offers an intuitive method for editing by presenting information in a form-based layout. It further enhances usability by grouping properties logically.

- *Model Validation View* in DIME displays the syntactic and static semantic checks of an active model in real-time during the modelling process. The purpose is to guide users with a list of warnings and errors to ensure the correctness of their models. The model validation view offers validation at two levels;
  1) Model level to enlist the individual checks of an active model and
  2) Global level to provide information about the validation results for the entire project

To achieve separation of concerns, DIME supports various types of DSLs modeling that represents different perspectives on the same comprehensive model according to their responsibilities. This modularity, reusability and "write once" rule is the core principle behind the coherence by construction approach. The DIME model types encompass:

- *Data models* are a fundamental part of DIME applications, providing a structured representation of the domain and covering the different aspects of the persistence layer such as types, inheritance, attributes, and relations to describe different kinds of data and their relationships.

- *Process models* is a key part of the application's business logic. In coordination with internal and external native DSLs, they provide a structured way to capture complex workflows and can be organized into different types of operational needs and hierarchies, e.g., data retrieval and processing, interaction and communication among other processes etc.

- *Graphical User Interface (GUI) models* are declarative structures that define the user interface of a web application, including its look and feel, actions, navigation, and data bindings. A rich set of customizable reusable components for IO operations, front-end presentation, and advanced tools for structural design enables rapid prototyping and agile development.

2. RELATED WORK

- *Access control models* define the permissions and roles that govern who can perform certain actions in the application.

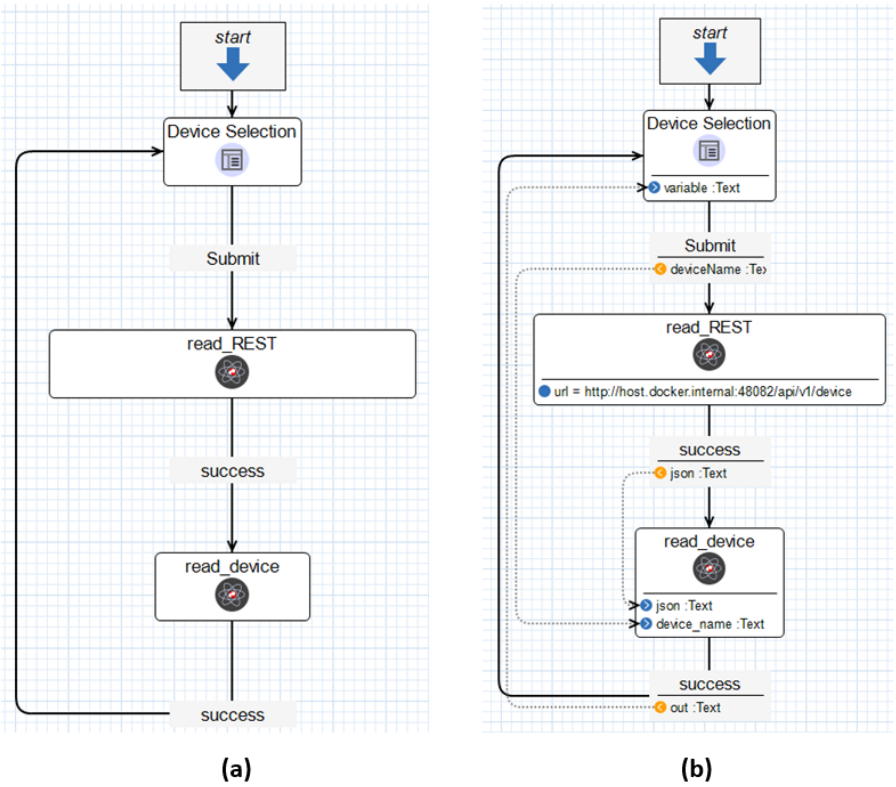


Figure 2.2: (a) Control flow and (b) Data flow in REST services pipeline.

The process model pipelines in DIME support both the control flow and data-flow aspects of an application as shown in Fig. 2.2 (a) and Fig. 2.2 (b) respectively. The control-flow aspect defines the order and sequence of activities that need to be executed in a certain order to achieve a desired outcome. It is represented by nodes and edges (solid lines) that connect them, where each node represents an activity or decision point, and each edge represents a transition from one node to another. The data-flow aspect (dotted lines), on the other hand, defines how data is passed between nodes in the control flow. When the control flow reaches a node with an input port, it executes a read/write operation on the connected variable to retrieve the required data. Each node in the pipeline is an executable

block referred to as SIBs and belongs to a family of DSLs that exposes a collection of related SIBs. The SIBs are reusable and instantiable with either an associated code implementation or a hierarchical process model. By presenting workflows and data-flows in a native representation such as in DIME and Pyrus, certain errors become more readily apparent than if they had to be inferred from the linear syntax of customary code. The extraction of the control flow and data-flow graphs helps in analyzing dependencies and performing program analysis and verification, which is an advantage for designers to use these automatically generated mathematically correct representations, rather than extracting them from traditional program code where they are only implicitly present. This simplicity [104], service-oriented agility [105] and built-in quality assurance [106] are key design principles of DIME and make it a promising "game changer" in the LDE.

2.4.3 The Underlying No-Code Analytics Environment - Pyrus

Pyrus [3] is again a graphical modeling environment for the implementation of data-driven AI/ML pipelines. Pyrus integrates its graphical web environment with an established online Integrated Development Environment (IDE) like Jupyter and allows users to combine functional components to data-flow-oriented workflow in a collaborative fashion. From the perspective of developing DSLs, annotations are added at the top of the Python functions in a simple SIB declaration language. Pyrus reads them and with that information, it presents them to the users as a SIB in the corresponding DSL. Fig. 2.3 shows the user interface of Pyrus and there are four major sections:

- *Explorer* section lists all the models in the project and facilitates users to navigate and manage the artefacts in flat or hierarchical structuring within a project.
- *Ecore* section displays a collection of default DSLs as well as external DSLs. This way, the users can access the SIBs of those DSLs and orchestrate them graphically in a data flow fashion.

2. RELATED WORK

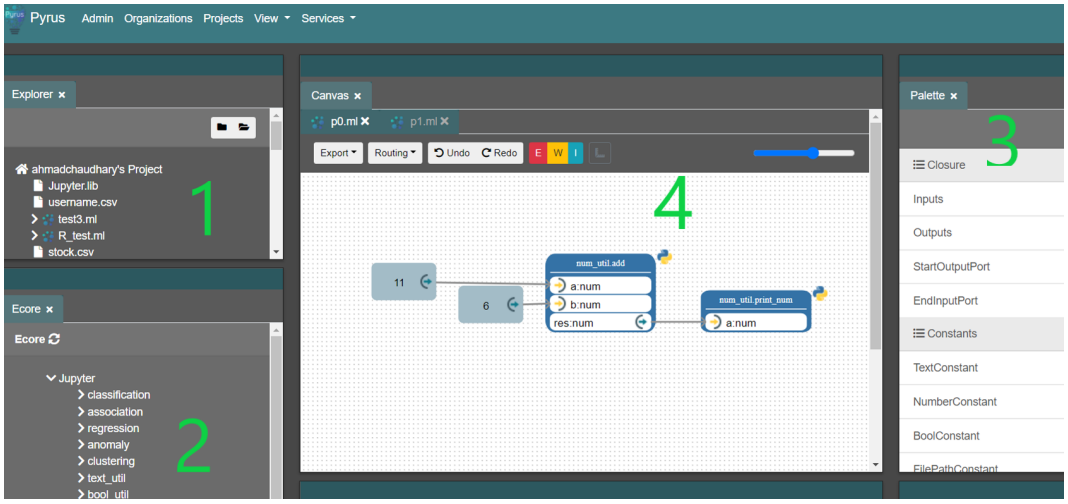


Figure 2.3: User interface of Pyrus: (1) Explorer. (2) Ecore. (3) Palette. (4) Canvas.

- *Palette* section provides a list of standard input/output holders to attach variable values at runtime.

- *Canvas* is the drawing board to design data pipelines with the help of SIBs and I/Os from the Ecore and Palette sections.

After developing the pipelines in Pyrus, users can initiate execution directly within the environment. The execution process consists of several steps as shown in figure 2.4. The Pyrus SIBs are usually in Python and on execution it generate the Python code for orchestration and configuration, based on the designed graphical pipelines. For function discovery and pipeline executions, Pyrus communicates with a connected JupyterHub over the ZeroMQ protocol. Firstly, a data-driven Process model needs to be translated into an imperative control-flow graph because the online IDE's external execution environment supports only imperative programming languages. Following the creation of the control-flow graph, it can be straightforwardly translated into the target language with the assistance of a code generator. This code generator generates the appropriate calls to external functions in the specified order and configures the variables according to the output ports. It also reconstructs the process hierarchy.

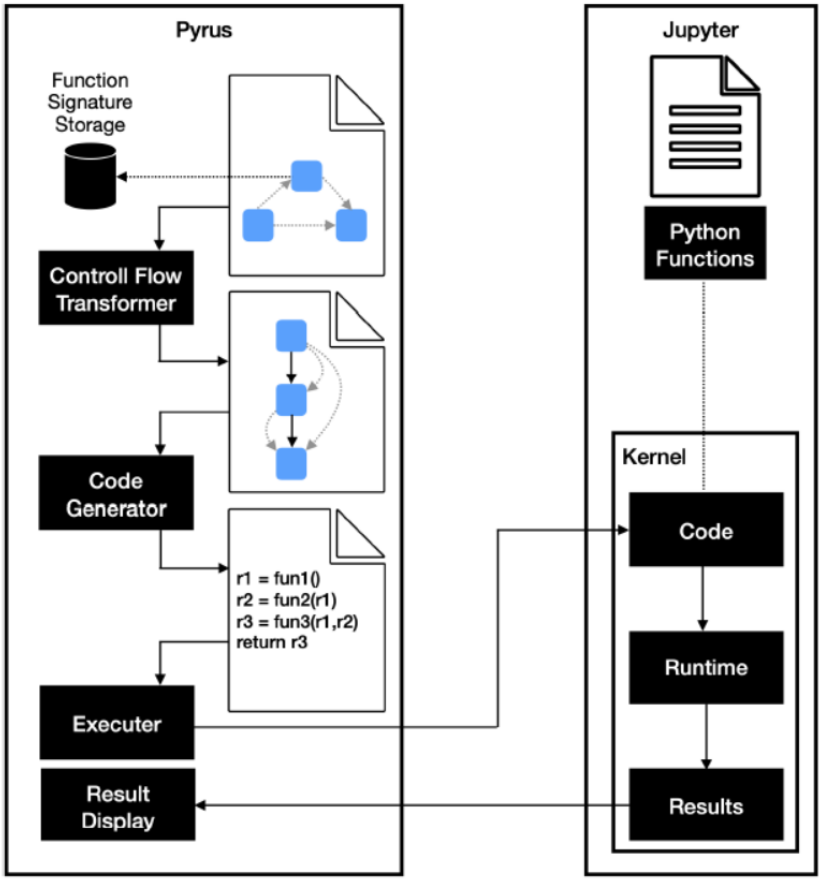


Figure 2.4: Runtime execution workflow of Pyrus - originally from [3].

Finally, Jupyter transfers the results back to Pyrus, which are then displayed to the user within the modeling environment. These results can encompass primitive data such as text, as well as complete images, charts, tables etc. For each component, a fire-rule similar to Petri nets can be specified, under certain restrictions. This approach ensures that the designed models are syntactically correct and executable, alerting the user to where incompatibilities exist.

2.4.4 EdgeX Foundry

EdgeX Foundry [4] is an open-source framework and provides a vendor-neutral platform that simplifies the development of IoT applications and services that operate at the edge of the network. It is one of the popular Industrial IoT mid-

dleware platforms that is being widely adopted by industry, particularly for industrial automation applications. It enables organizations to leverage the advantages of edge computing and edge intelligence. This highly flexible, hardware-agnostic platform is composed of loosely coupled microservices. Figure 2.5 shows the platform architecture of EdgeX Foundry, where microservices are broadly categorised into four service layers:

- *Device services layer* provides the necessary interfaces and connectors to communicate with diverse edge devices, ensuring seamless integration and compatibility with a wide array of industrial hardware, sensors, and actuators.
- *Core services layer* forms the backbone of the platform's data processing capabilities and consists of critical microservices responsible for data ingestion, storage, metadata management, and data export.
- *Supporting services layer* consists of microservices like logging, notifications, and scheduling, which offer essential support functions to enhance the reliability, monitoring, and management of EdgeX deployments.
- *Application services layer* provides a platform for developing custom applications and services that leverage data and functionality from the lower layers. It empowers developers to create innovative solutions tailored to specific IoT use cases.

The key features of the EdgeX Foundry are

- *Microservices-Based:* EdgeX Foundry adopts a microservices architecture, where various components and services are decoupled and can run independently. This modular approach makes it easier to add, remove, or update individual services without affecting the entire system.
- *Device Services:* At the heart of EdgeX Foundry are device services that communicate with IoT devices and sensors. These services handle device-specific protocols, making it possible to connect a wide variety of devices, regardless of their manufacturer or communication standard.

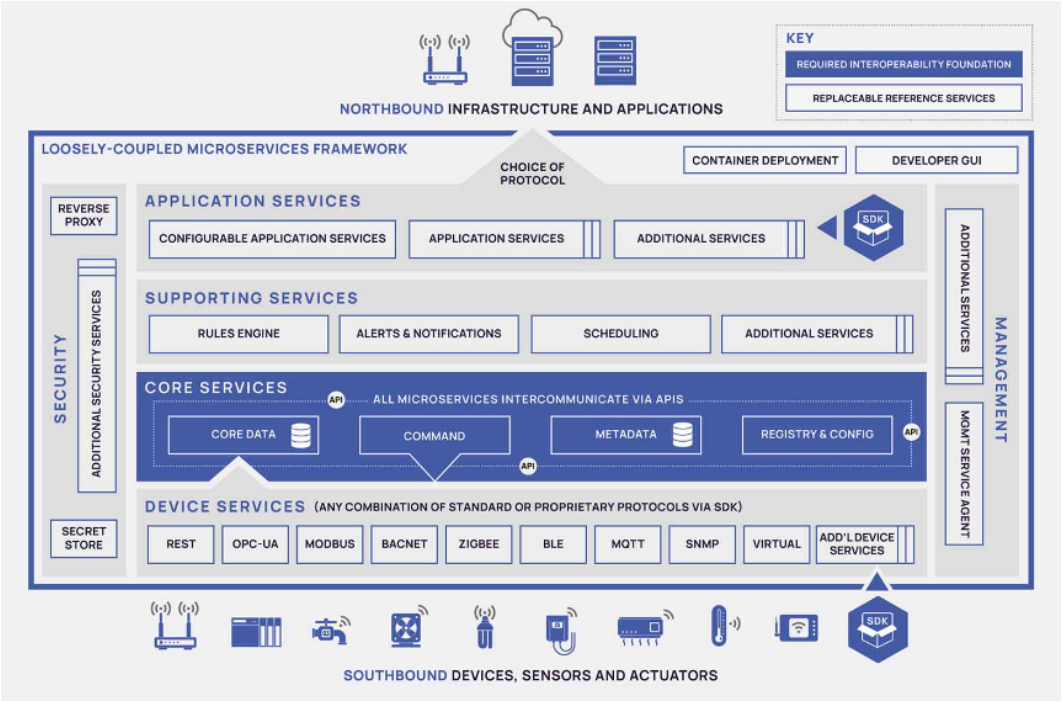


Figure 2.5: Platform architecture of EdgeX Foundry - originally from [4].

- *Security:* EdgeX Foundry incorporates security features, such as authentication, authorization, and encryption, to protect data and devices at the edge.
- *Device and Data Management:* The platform offers device discovery, registration, and management capabilities. It allows administrators to monitor and control devices remotely, ensuring their proper functioning.
- *Data Persistence:* EdgeX Foundry includes a choice of data persistence options, including databases and cloud storage, to store and manage the vast amounts of data generated by edge devices.
- *Event Handling:* EdgeX Foundry supports event-driven architectures and enables applications to subscribe to specific events and trigger actions in response to them.
- *Interoperability:* One of the primary goals of EdgeX Foundry is to ensure interoperability between different components and devices. It achieves this

through open APIs and standards, allowing developers to build solutions that work across a wide range of devices and platforms.

- *Scalability:* The architecture is designed to be scalable, making it suitable for both small-scale deployments and large-scale industrial IoT solutions. As the number of devices and data volumes grow, EdgeX Foundry can adapt to meet the increased demands.

In summary, the platform architecture of EdgeX Foundry provides a robust foundation for building edge computing solutions for IoT. Its modular, secure, and interoperable design allows developers to create scalable and flexible edge applications that can seamlessly integrate with a wide range of devices and systems.

2.4.5 Tines

Tines [107] is a no-code automation tool that was initially built for automating workflows (called stories) in the domain of security. Fig. 2.6 shows a sample story in Tines. Actions are the building blocks of Tines automation stories, i.e., generic components that are similar to highly parametric SIB templates in the world of DIME and Pyrus. Tines offers a set of seven core actions that enable users to automate and scale almost any manual workflow, making it a versatile and efficient automation solution.

- *Send Email Action* sends emails to recipients specified in the Action options.
- *Event Transformation Action* has several modes that modify the contents of incoming Events.
- *Hypertext Transfer Protocol (HTTP) Request Action* sends HTTP requests using POST or GET Methods to a specified Uniform Resource Locator (URL).
- *Receive Email Action* in Internet Message Access Protocol (IMAP) mode emits Events when it detects new emails on an IMAP server.
- *Trigger Action* compares the contents of a field from an incoming Event with predefined rules, when the rules match, an Event emit is triggered.

- *Webhook Action* will emit Events it receives through Webhooks (HTTP callbacks).
- *Send to Story Action* allows you to send data to a sub-story where it can be further processed.

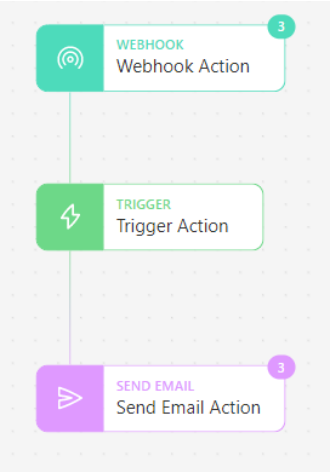


Figure 2.6: Tines automation pipeline.

2.4.6 Docker

Docker [108] is a containerization platform that allows developers to automate the deployment of applications inside lightweight, portable, and self-sufficient *containers*. These containers encapsulate all essential components for running an application, encompassing the code, runtime environment, libraries, and system utilities. Docker provides a consistent and isolated environment, ensuring that an application runs reliably across different environments, from development to testing and production.

2.4.7 Raspberry Pi

Raspberry Pi (RPi) is a versatile and affordable single-board computer that has revolutionized the world of technology, particularly in the context of the IoT. In the realm of IoT, RPi plays a pivotal role as a powerful and accessible hardware

platform. Multiple RPis can also be utilized in a master-slave configuration to address the heterogeneity of the distributed architecture. Its combination of affordability, performance, and an active community makes it a driving force behind the rapid growth of IoT applications in various industries and educational settings.

2.4.8 Sensors and Devices

The FiPy by Pycom is a powerful IoT development board featuring an ESP32 microcontroller with MicroPython support. It offers extensive wireless connectivity options, including WiFi, Bluetooth, LoRa, Sigfox, LTE-M, and NB-IoT. With GPIO pins for external sensors and expandability, it is suitable for a wide range of IoT projects. The FiPy simplifies development with its open-source ecosystem and can be used for applications such as smart agriculture, industrial IoT, smart cities, asset tracking, and environmental monitoring.

The PySense by Pycom is a sensor shield designed for IoT development boards like the FiPy. It comes equipped with various built-in sensors, including ambient light, humidity, temperature, pressure, and accelerometer sensors. PySense simplifies sensor integration for IoT projects, making it suitable for applications like environmental monitoring and asset tracking. PySense enhances the capabilities of Pycom boards, enabling users to collect diverse sensor data for their IoT applications.

The most commonly used sensors for machine or tool maintenance are vibration and temperature sensors. Some widely used models, such as the EPH-V11, EPH-V17, EPH-V18, and EPH-T20 [109], provide vibration and temperature data through wireless communication. These sensors report data using ModBus, which is one of the most widely adopted standards for industrial communication.

The cameras used are RPi High Quality Camera (HQCam) Modules equipped with CGL interchangeable lenses [110]. These HQCam modules feature 12.3-megapixel cameras, a 7.9mm diagonal image size, support for 12-bit RAW footage, adjustable back focus, and compatibility with C/CS mount lenses. Additionally, the CGL lenses are 3-megapixel 6mm HD CCTV lenses with built-in IR filters. The camera modules are connected to the RPi through 200mm ribbon cables.

3

Design and Implementation

In the modern era, a Digital Thread offers a robust reference architecture that drives innovation, efficiency, traceability, and accountability of any data, processes, and communications throughout the entire lifecycle of system of systems. This modern paradigm provides an organized and more structured approach towards integration and interoperability challenges. However, for this new approach to become mainstream, systems and their models must be connected through an integrated platform that can automatically transform, analyze, generate, and deploy data and processes and effectively leverage the formalized knowledge of the numerous entities involved.

3.1 Digital Thread Platform

According to the proposition [13], the entire data mobility, the operations and the communication between heterogeneous elements should be mediated through the Digital Thread Platform, at best in real-time or near-real time. Fig. 3.1 illustrates a seven layer architecture of the platform with a user-first approach, where complexity and heterogeneity increase from top to bottom to cater the diverse needs of various services, devices and platforms. The domain experts at the very top *Applications layer* will be interacting with the choice of their low code development environment, e.g., DIME [2] or Pyrus [3] in the second layer, i.e., *LCDEs Layer*.

3. DESIGN AND IMPLEMENTATION

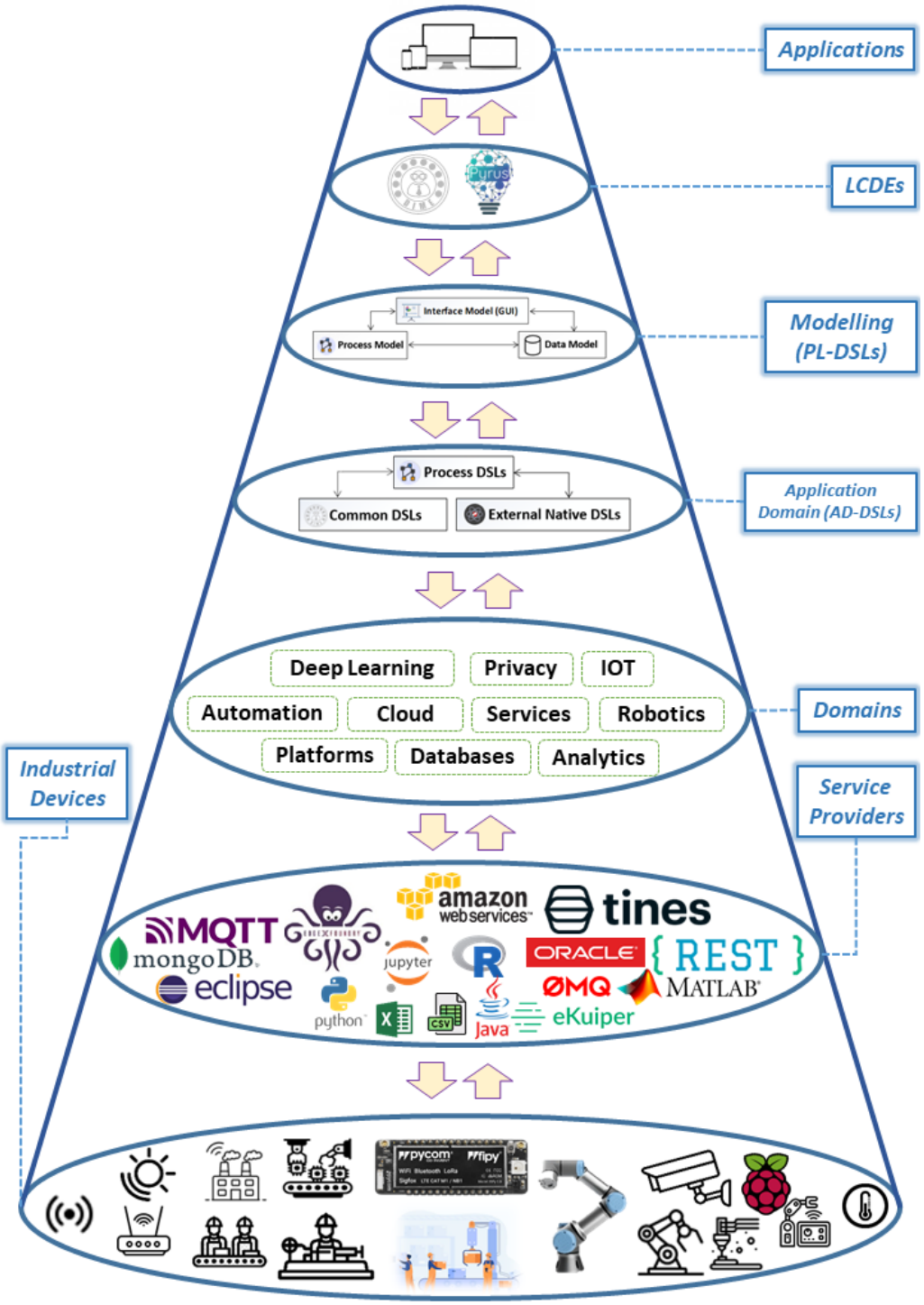


Figure 3.1: Architectural diagram of Digital Thread Platform.

The *Modeling Layer (Programming Level - Domain Specific Languages (PL-DSLs))* that comes with the standard DIME distribution[1] provides essential modeling capabilities, i.e., the user interfaces, the data models and the process models for the design and development of the supporting environment of an application.

Seen from the perspective of potential users, applications and business logic, the core of the DTP concerns the *Application Domain Layer (AD-DSLs)* and onward. As our focus is on CONFIRM applications in the domain of CPS/Smart Manufacturing and accordingly, the goal is to accommodate the conception, design and implementation of a wide range of devices, data stores, AI platforms and applications, local and cloud storages, communication protocols and IoT application dashboards for real-time analytics and control, e.g., safe operations of machine, stable storage facility etc. This is where the integration of external native DSLs plays a key role, as it allows a collection of complex functionalities and implementations to be abstracted in a way that they can easily be used as drag-and-drop SIBs for no-code application development.

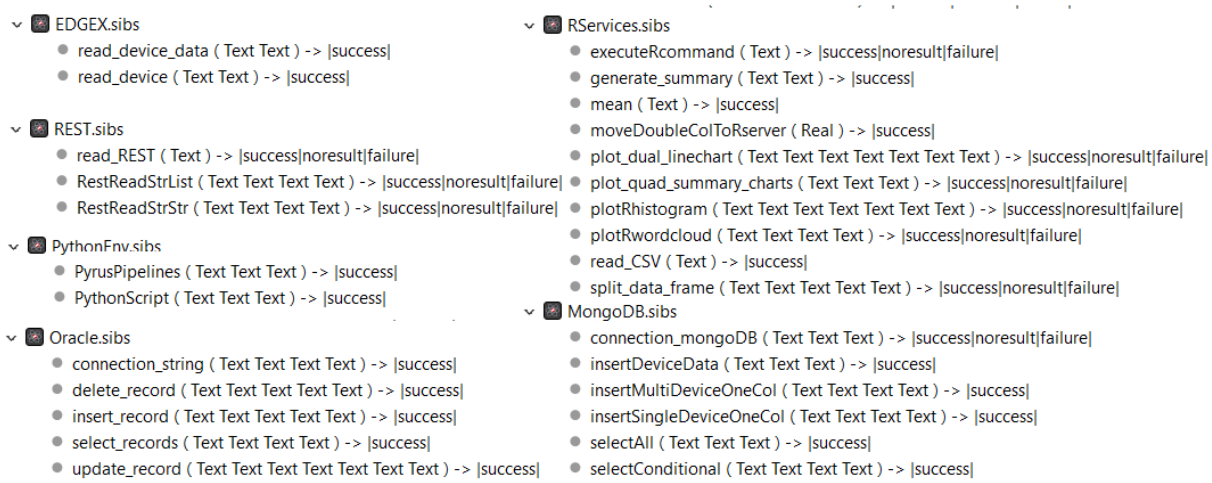


Figure 3.2: Native SIBs collections (SIBs palette developed).

Fig. 3.2 shows a selective collection of External Native SIBs that have been developed, thoroughly tested and integrated into the platform. The SIBs are then categorised into the respective Native DSLs and cover a wide range of domains, as depicted in the *Domains layer* in figure 3.1. These native SIBs are the minimum

[1] Accessible online at `https://ls5download.cs.tu-dortmund.de/dime/daily/`

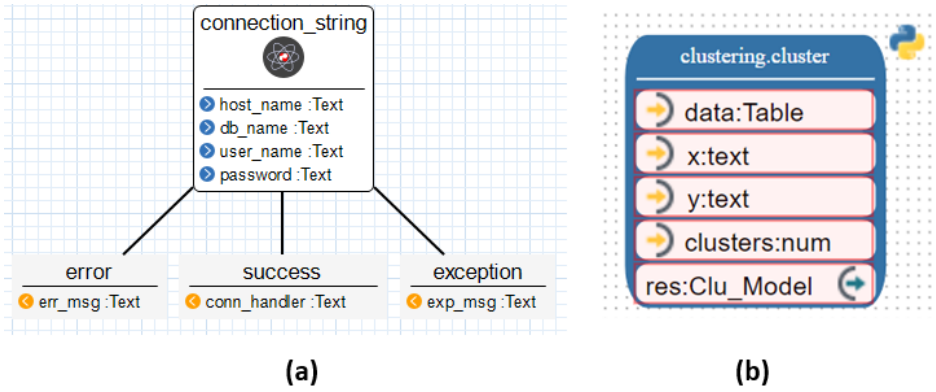


Figure 3.3: Visual representation of an example SIB in DIME and Pyrus.

granular executable unit in the platform and for every new integration in the systems, the native library approach based on our prior integration experience [29] on the analogy of microservices was opted. This integration mechanism is referred to as shallow integrations [1], and the thesis focuses on building AD-DSLs through the provision of domain-specific SIBs that are implemented in their own DSL, e.g., the language underlying DIME and Pyrus.

As an example, fig. 3.3 (a) and fig. 3.3 (b) show the visual representation of an example SIB in DIME and Pyrus respectively when used in process modeling. Here, both of the SIBs require four inputs (inward arrows) and one output (outward arrow (of the selected branch in the case of DIME)). The inputs can either be static values inserted from the properties view or the output value from another SIB using the inward data-flow. In the case of DIME, the control branch/es defines the possible outcome/s of the SIB under consideration, e.g., three in this case. Upon successful execution, the outgoing control branch labelled success is taken, with the corresponding data output to be used later in the workflow. If there are known issues at runtime (e.g., connection error), an error branch is followed with the corresponding error message and in case of unknown exceptions at runtime, the exception branch is followed with the exception message.

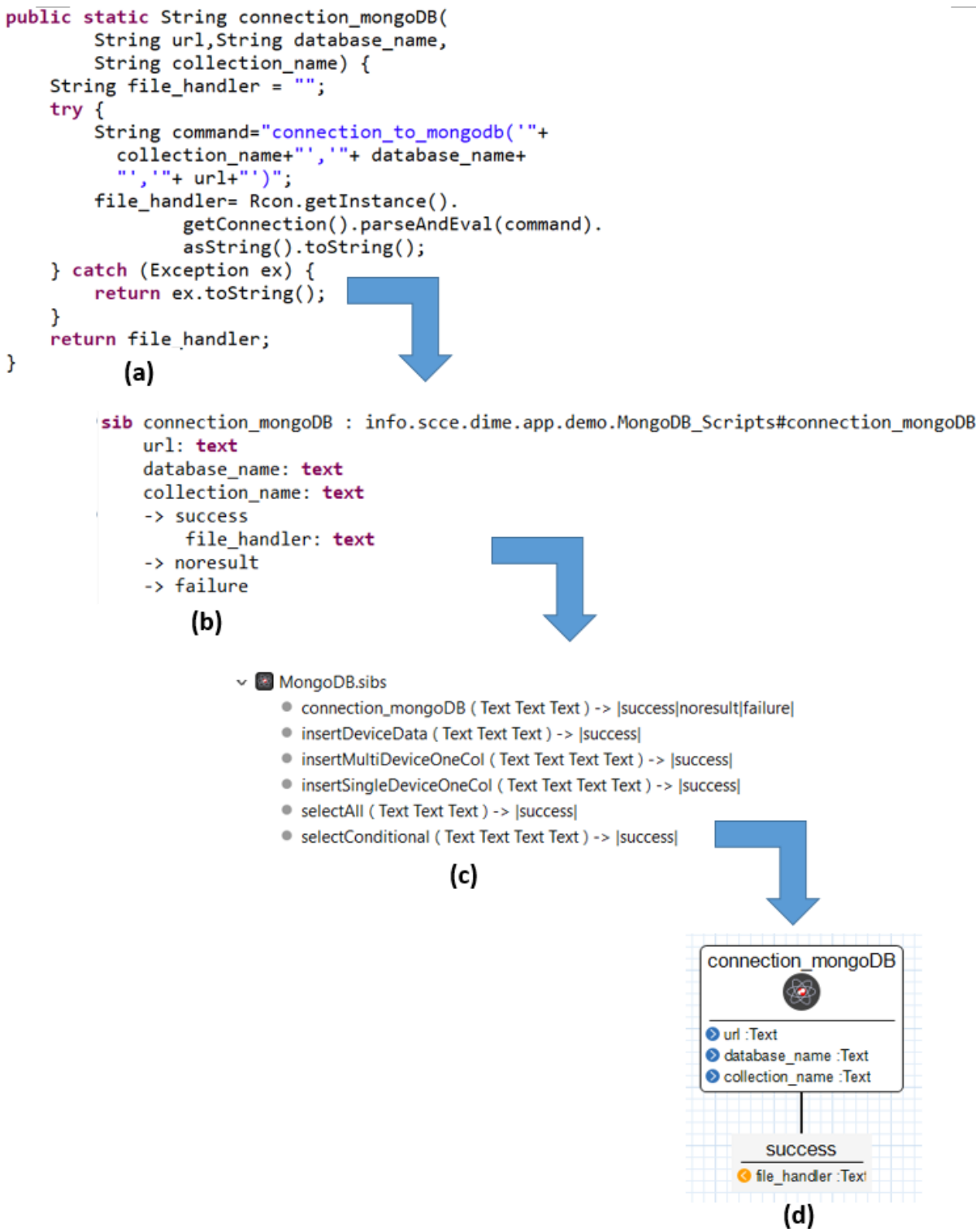
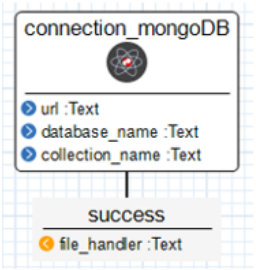


Figure 3.4: Lifecycle of a SIB in DIME (a) Backend Java implementation of SIB (b) Signatures for the SIB declaration (c) SIB in a DSL palette (d) Visual representation of the SIB.




```
public static String connection_mongoDB(
        String url,String database_name,
        String collection_name) {
    String file_handler = "";
    try {
        String command="connection_to_mongodb('"+
          collection_name+"','"+ database_name+
          "','"+ url+"')";
        file_handler= Rcon.getInstance().
                getConnection().parseAndEval(command).
                asString().toString();
    } catch (Exception ex) {
        return ex.toString();
    }
    return file_handler;
}
```

Figure 3.5: SIBs categories - the native SIB and the associated sample code implementation.

Fig. 3.4 give the four step overview for an integrated SIB connection_MongoDB in the platform. It establishes a connection with a remote instance of the MongoDB cloud database and returns a connection handler. Fig. 3.4 (d) shows the visual representation of the SIB when used in process modeling, it requires three inputs (blue inward arrow) that can either be static values or output from another SIB and one output (orange outward arrow). Accordingly, Fig. 3.4 (a), Fig. 3.4 (b) and Fig. 3.4 (d) shows the backend Java implementation of SIB to be invoked at runtime when used in process modeling, the annotated signatures for the SIB declaration, the outlook in DSL-SIBs palette respectively.

Depending on the integration requirements and accessibility to the readiness of the required functionality, the two distinct levels of extension mechanisms are described in our work [29], where in case of readily available Services, the integration will undergo the process of native library approach and otherwise for the platform approach, the additional overhead will be of the preparation of the platform in separate dockerized containers to addresses the needs of the underlying application. The central property of simplicity here is again that, once integrated, it becomes part of the SIBs palette and reusable as a draggable component in the modeling, and the backed executable implementation is integrated with the process modeling code in the phase of the model to code generation. The com-

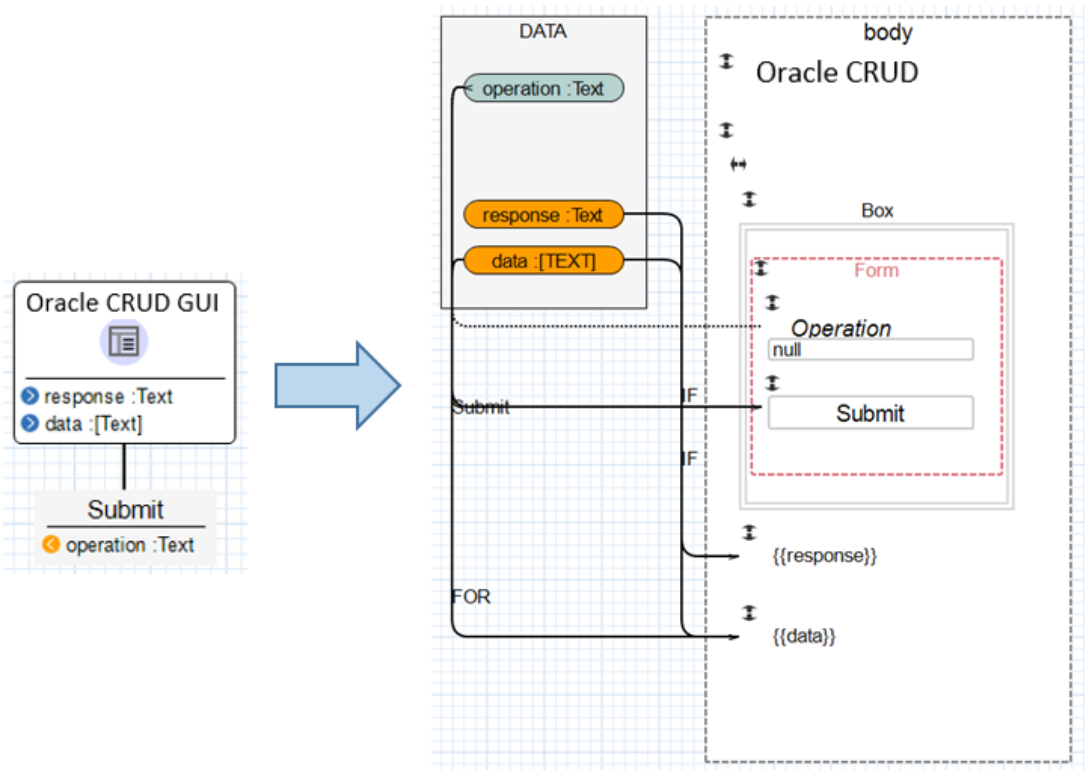


Figure 3.6: SIBs categories - the GUI SIB and the associated sample implementation through a GUI model.

plete platform artefacts and Docker containers can be found in the SCCE Gitlab repository[1]. As an example, the list of SIBs signatures and the backend Java implementation for one of the AD-DSL related to R-programming and analytics capabilities are also provided in the appendix III of the thesis.

Given an application under development, the Pyrus workflows are relatively simpler as they only support data-flow but in the case of DIME, the process is usually an orchestration of a number of sub-processes and a process model consists of control flow, data-flow and the set of possibly three different types of SIBs, as shown in Fig. 3.5, Fig. 3.6, and Fig. 3.7: the native SIB, the GUI SIB, and the hierarchical SIBs, respectively. The native SIBs are atomic executable units and have a direct binding with the backend executable code. The GUI model consists of GUI components and has an associated GUI modeling canvas. The

[1]SCCE Gitlab repository: https://gitlab.com/scce_ie/da_platform

3. DESIGN AND IMPLEMENTATION

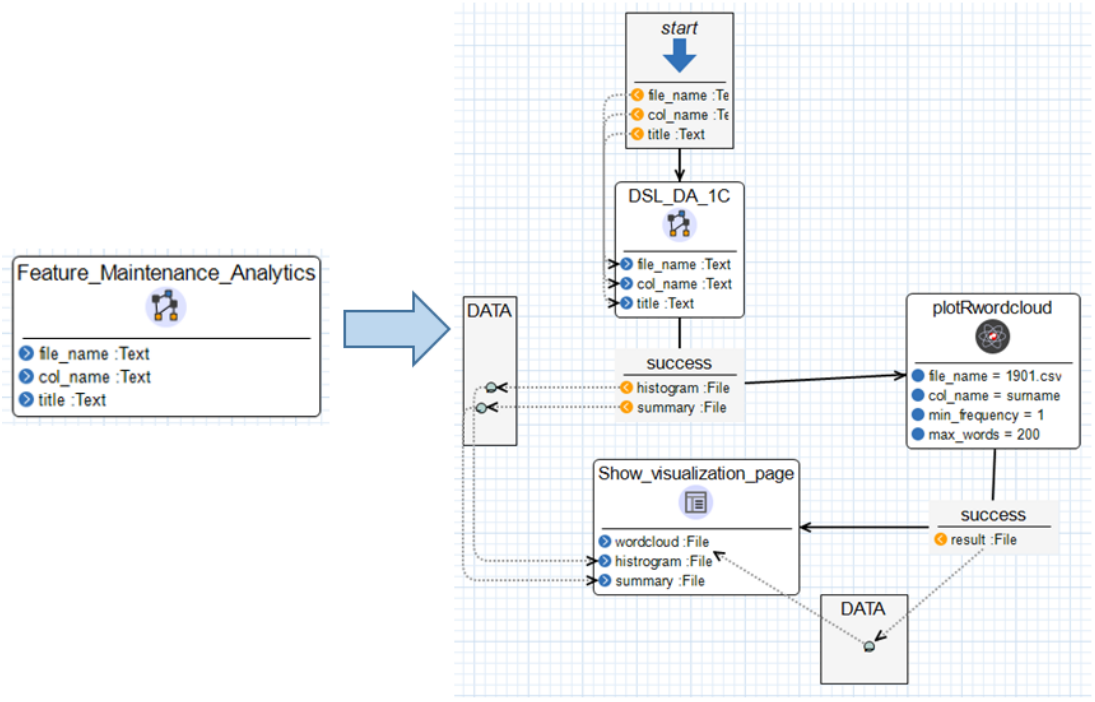


Figure 3.7: SIBs categories - the hierarchical SIB and the associated sample implementation through a process model.

hierarchical SIBs further contain a set of nested process models with possibly all the discussed SIBs types that collectively describe the control flow and data-flow of the application based on the given business logic.

The *service provider and industrial devices layer* in figure 3.1, shows a wide collection of services, platforms, devices and communication protocols that are supported in the platform, i.e., IoT through EdgeX [111, 112], Robotics with the Universal Robots Language [13], REST services [28], R, as a scripting language and an analytics platform [28], cloud services etc. We also consider ourselves a systematic user of existing platforms, such as EdgeX for IoT, Analytics in R and Python scripting etc, which is a valuable addition to the platform. These platforms adopt a bottom-up approach for component provisioning, which then can be orchestrated by an expert. In this sense, the value proposition is situated on the uppermost layer of application development, where the interoperability challenge truly lies.

3.1.1 Application Domain DSL - Example in DIME

This section presents an example process model in DIME taken originally from our work [5]. Fig. 3.8 shows the business logic of the application for the data acquisition from the IoT devices and ingestion into MongoDB.

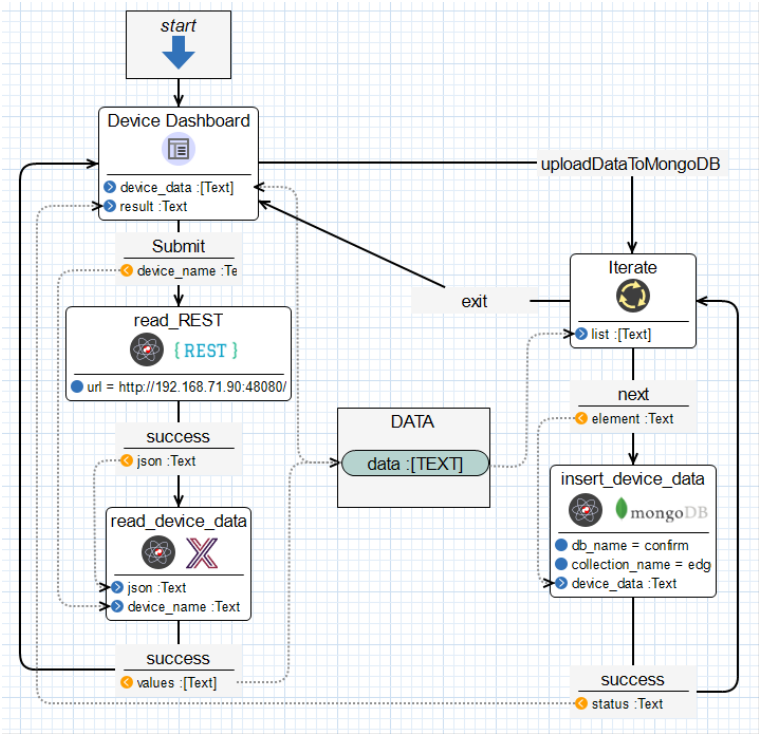


Figure 3.8: Data acquisition process in DIME (originally from labelled paper - NA2).

The start SIB indicates that we have here a stand-alone process that does not receive any inputs from the context. The Device Dashboard SIB is implemented through a GUI model that displays the application webpage to the user, where the user writes the name of the IoT device of interest. On clicking submit on the GUI, the control flow moves to successive SIB read_REST that has the server URL as a static value. This SIB reads the device data from the EdgeX Foundry instance running on the Raspberry Pi and retrieves the corresponding JavaScript Object Notation (JSON). The SIB read_device_data parses the JSON received from EdgeX Foundry and extracts the device related instances, which are passed

as input to this SIB from the GUI SIB. We see here that the data-flow and the control flow differ: while so far we have a linear pipeline in the control flow, the data for some SIBs is provided by various components, at various times. It is therefore useful for the developer to be able to see and design or check separately both the control flow (the logic of what is done, step by step) and the flow of data, which is typically prepared in a number of steps and then consumed by a SIB that collects several inputs and processes them. At this point, the control flow and data-flow return to the GUI, where the extracted data is displayed on the web page.

The alternative control flow on the right side of Fig. 3.8 is activated if the user decides to store this data, by clicking on the Upload button from the GUI: the iterate goes through all the tuples of the dataset and inserts them into the MongoDB cloud database. For this, the provided input parameters are the database connection string, the collection name, and the device. Upon completion, the control returns to the GUI SIB, to display on the webpage the status of the workflow, and to be ready to accept further inputs. There is no end SIB as this Web application does not terminate its execution: it is always available, as part of a device command and monitoring infrastructure.

3.1.2 Application Domain DSL - Example in Pyrus

Here we present an example pipeline in Pyrus taken originally from our work [5]. Fig. 3.9 shows a data analytics processing workflow that we implemented in the Pyrus platform using Python as the language and platform of choice. Here, the Pyrus pipeline establishes a connection to the MongoDB database with the connection_to_mongoDB block, whose required inputs (i.e., MongoDB database URL, database name and collection) are provided as constant strings, which are the grey input blocks in the model. It fetches the requested data in JSON format and uses the Pandas package to create a dataframe. This dataframe is passed for preprocessing to the next block, convert_to_datetime: it converts the time of observation column to the correct time format for ease of analysis and for later plotting. The dataframe and the names of the channels are then passed to the

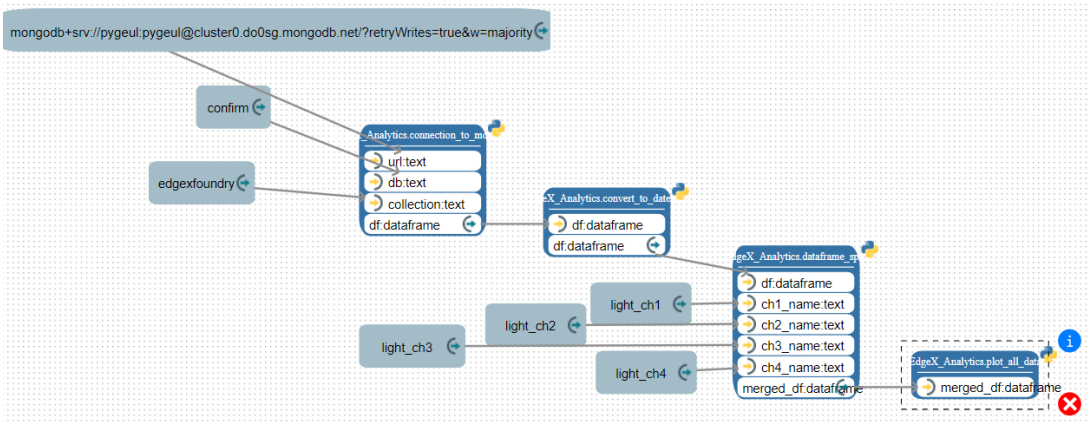


Figure 3.9: Analytics dashboard in Pyrus (originally from labelled paper - NA2).

block dataframe_split in order to create a separate dataframe for each light channel. Finally, the plot_all_data block plots them as graphs in the dashboard.

3.2 Research Methodology

The Three-Cycle View in Design Science Research (DSR) [113] serves as a practical framework for structuring research, ensuring that artefacts address real-world issues and undergo rigorous evaluation. This approach helps maintain a balance between theoretical contributions and practical solutions in information systems and technology.

In this project, we closely adhered to the principles of DSR, employing a Three Cycle View to guide my research approach. In the Relevance Cycle, we meticulously identified and defined a list of challenges and requirements for the platform, ensuring its practical significance. We set clear objectives that served as a compass throughout the project's lifecycle. During the Design Cycle, we dedicated ourself to creating a novel artefact tailored to address the identified problem. This entailed careful design and development, resulting in an innovative solution that we meticulously implemented. Finally, the Rigor Cycle was paramount in validating the effectiveness and quality of my artefacts. We conducted comprehensive evaluations to ascertain that our solution met its intended goals. Moreover, the feedback from end-users and stakeholders was continuously

obtained, fostering an iterative process of refinement and enhancement. This rigorous adherence to the Three Cycle View of DSR ensured that the project not only contributed theoretically to the field but also delivered a practical and impactful solution to a real-world problem.

The ability to carry out *agile modifications* on a use-case basis is due to the iterative and prototype-driven approach and that the research process does not stop once an artefact is created and evaluated. Instead, multiple iterations of these cycles are often carried out to continuously refine and improve the artefacts.

3.2.1 Development Life Cycle

Our application development approach supports the entire software development life cycle, from specifications to deployment and maintenance. The combination of service orientation and model-driven design is key to the platform for reducing the domain-IT gap. Fig. 3.10 shows the release cycle of an application under development and a list of processes if implemented in our tool, i.e., DIME vs. the list of possible different technologies that may be involved for the manual implementation in each phase of the development cycle. The implementation of these processes in one single tool, i.e., DIME vs. the variety of tasks in very different tools has a huge impact on the learning curve and the implementation timeline. Fig. 3.10 also shows the role of key stakeholders and the list of DIME processes involved in each phase of the development cycle. The domain expert takes the lead in most of the phases except the development phase, where the key dependency is on the software developer for the integration of new services, devices or platforms that are not already supported by the platform.

The first phase covers the specification of the data model, the design of the DIME application descriptor file and alterations to the startup process. The design phase involves the identification and modeling of different kinds of workflows, i.e., hierarchical and homogeneous processes, GUI modeling and connectivity of processes across different parts of applications for end-to-end workflow execution. Typically, once a concrete architecture has been defined, the remaining implementation mostly revolves around the design of process models. The development phase is a one-click automated code generation from models, supported

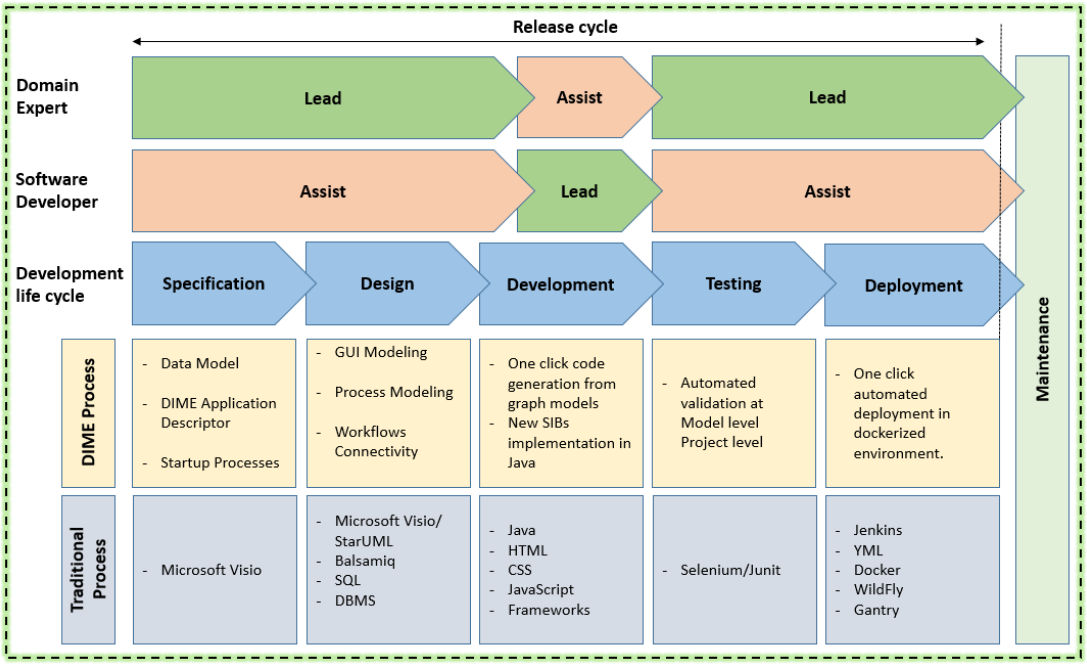


Figure 3.10: Comparison of product release cycle in DTP DIME Process vs. the traditional process.

by underlying DIME and Eclipse IDE technology. However, the implementation of the missing services as new SIBs require manual code implementation and unit testing and have a dependency on software developers. The platform supports the automated integration testing of all the involved SIBs and processes, and they are validated both at the model level individually and at the project level collectively. The deployment of the application is again a one-click automated deployment supported by the state of art technologies, i.e., WildFly, Docker and Gantry. Finally, the majority of the day-to-day adaptation and maintenance steps can be treated at the model level without requiring classical programming skills.

3. DESIGN AND IMPLEMENTATION

4

Results and Discussion

4.1 Research Contributions

The main question that this dissertation addresses is the proposition of a modern Digital Thread Platform for Cyber-Physical Systems. In order to address this question, this dissertation defines the baseline architecture [11] and addresses the associated challenges of integration [28] and interoperability [11, 29] in the design, development and implementation of DTP. The approach is validated with a series of applications and multiple industrial use cases within smart manufacturing, i.e., predictive maintenance [30], edge computing [5], edge analytics [5, 31], safety operations [32], stable storage facility [33] etc.

4.1.1 Integration of (Micro-)Services as Graphical AD-DSLs

The first contribution of this dissertation lies in the proposition of an integration methodology aimed at extending the (application) DSL within two of the LCDEs discussed, which are based on formal models, i.e., the general purpose DIME and the special purpose, web-based Pyrus. DSLs effectively encapsulate many complexities inherent to the underlying application domain. The proposed method extends these platforms by integrating external services on the analogy of microservices. The extension of functionality and integration with external systems through services broadens the capabilities of these platforms to cater a wider

communication needs. Moreover, it enables them to leverage existing sophisticated enterprise services. Encapsulation of code and abstraction to semantically faithful representations in models empowers domain experts to take advantage of these platforms. They can develop products in an efficient manner and meet the growing demands of application development without requiring in-depth expertise in software development. The implementation details are elaborated in the following journal publication:

Publication (attached as AP1)

H. A. A. Chaudhary *and T. Margaria*
Integration of micro-services as components in modeling environments for low code development
in *Proceedings of the Institute for System Programming of the RAS, vol. 33, no. 4, 2021*

The proposed method successfully integrated a wide range of services and functionalities. However, further challenges emerged at runtime, including issues such as the unavailability of integrated services, network latency delays, and conflicts in supported versions of libraries. To address these challenges, the extension mechanisms are categorized into two distinct levels: services and platforms. Service integrations concern readily available services, as discussed earlier. However, for platform integrations, dockerized environments are designed and deployed with the support of most standard functionalities. The integration mechanism and the preparation of dockerized environments are presented in the form of a step-by-step guided tutorial in the following conference publication:

Publication (not attached)

H. A. A. Chaudhary *and T. Margaria*
Integrating External Services in DIME
in *Leveraging Applications of Formal Methods, Verification and Validation, Cham: Springer International Publishing, 2021, pp. 41–54.*

Research Impact

The initial research related to integration laid down the foundation for the proposed platform with a straightforward approach to integrating external services. Each new integration is evaluated as either a service or a platform integration and contributes to the baseline architecture of DTP. This contributes to the aspects of reusability and represents a new approach for developing future applications using these integrations, without the need to write any additional manual code for the same functionality. Based on these integrations, we have also conducted a guided workshop at an international summer school called STRESS (School on Tool-based Rigorous Engineering of Software Systems).

4.1.2 Interoperability Challenge in Digital Thread Platform

The main contribution of this dissertation is the design of a robust reference architecture of the Digital Thread Platform to drive innovation, efficiency and traceability of any data, process and communication along the entire system (or system of systems) lifecycle. For this new paradigm to enter the mainstream, systems and their models need to be connected through an integrated platform that covers interoperability aspects through the production of a DSL-based integration layer so called External Native DSLs layer. This is a much more structured and organized way to look at integration and interoperability. In this respect, the value proposition sits clearly at the upper, application development layer, where the interoperability challenge truly resides.

In the context of Industry 4.0, a series of relevant DSLs have been proposed, targeting a wide range of domains, technologies, and platforms, and covering areas such as data persistence, robotics, AI/ML, analytics etc. to support the basic operations of smart manufacturing. The central property of simplicity here is that once integrated, all Native DSLs within DIME exhibit uniform usage and representation. The architectural and implementation details are elaborated in the following conference publication:

Publication (attached as AP2)

T. Margaria, ***H. A. A. Chaudhary****, I. Guevara, S. Ryan, and A. Schieweck*
The Interoperability Challenge: Building a Model-driven Digital Thread Platform for CPS
in *ISoLA 2021, Leveraging Applications of Formal Methods, Verification and Validation, Cham: Springer International Publishing, 2021, pp. 393–413*

Following the same analogy and principles, the integration architectural layer has been introduced in Pyrus. However, considering it as a special-purpose platform, only the data analytics capabilities have been included to address the heterogeneity of Python libraries and platforms. The External Native DSL in both architectural designs has been tested and validated with several case studies using open-source datasets within the domain of smart manufacturing, e.g., manufacturing analytics and predictive maintenance etc.

While designing the Proofs of Concept (POCs) for the discussed applications in the alpha testing phase, we received feedback regarding the repetition of the same steps when implementing case studies with similar functionality but in different contexts. For example, implementing data collection User Interface (UI) or data analytics pipelines from scratch using individual SIBs, which have similar structures but are intended for different domains. In response to this concern, we have introduced higher-level SIBs, known as features. These features are ready-made workflows that go beyond the manual, ad-hoc integration of individual SIBs, providing higher-order functionalities as drag-and-drop services. The details of this work are presented in the following journal publication.

Publication (attached as AP3)

H. A. A. Chaudhary *and T. Margaria*
DSL-based Interoperability and Integration in the Smart Manufacturing Digital Thread
in *Electronic Communications of the EASST, vol. 81, 2022.*

Research Impact

The research showcases the concept of a Digital Thread platform with numerous case studies and POCs. The research enables colleagues within the research group to initiate the integration of external services into their selected domains, such as business, history, medicine, etc. Demos with external entities have also fostered collaboration with external universities and research groups, resulting in several successful case studies in practical environments. The selected topics have also been utilized as teaching case studies for undergraduate and postgraduate classes. In terms of social engagement, several guided no-code workshops were conducted for the general public and in schools to provide a taste of computer science.

4.1.3 Industrial Use Cases in Smart Manufacturing

This part of the dissertation was a collaborative effort involving various research hubs within the university and stakeholders from other universities within the CONFIRM Hub. It primarily contributes to the applied aspects of the designed platform in the domain of Cyber-Physical Systems, with several use cases in Industrial Internet of Things (IIoT), e.g., IoT applications, control, edge analytics etc. A set of applications has been developed for industrial use cases in Smart Manufacturing, i.e., safe operations of machines, and a stable storage facility with a balanced combination of cloud and edge computing for real-time operations. The ecosystem connects a range of technologies and frameworks in a low code manner: DIME platform for process modeling and master application, Pyrus and R for data analytics in Cloud, eKuiper for edge analytics, Tines for automatic notification management, EdgeX Foundry platform as middleware for IIoT devices and Eclipse Mosquitto for MQTT protocol for data acquisition from heterogeneous sensors. DSLs with a family of SIBs, support orchestration of different technological layers that empower the prototype-driven application development and demonstrate how engineers can build innovative IoT applications without having the full coding expertise.

In the first case, we employed our DTP as a low-code approach for the development of an IoT application targeting Edge Analytics. In the experimental setup, Pycom FiPy devices were installed in the Lero building at the University

of Limerick to record observations at fixed intervals. This data is transmitted to an MQTT Broker, which is part of the EdgeX Foundry framework, running at the edge on the orchestrator device, a Raspberry Pi. We adopted EdgeX Foundry as an example integration platform, addressing the challenge of interoperability among a heterogeneous set of devices, protocols, and IoT objects and accordingly, the pipelines were designed for data acquisition and basic data analytics. The significance of this approach lies in its contribution to the DTP through the DSLs of EdgeX Foundry, including REST, R, and MongoDB. Further details can be found in the following conference publication, which was also nominated for the Best Paper Award.

Publication (not attached)

***H. A. A. Chaudhary**, I. Guevara, J. John, A. Singh, T. Margaria, and D. Pesch*
Low-code Internet of Things Application Development for Edge Analytics
in *Internet of Things. IoT through a Multi-disciplinary Perspective, L. M. Camarinha-Matos, L. Ribeiro, and L. Strous, Eds. Cham: Springer International Publishing, 2022, pp. 293–312.*

Later, the work was expanded to include the ability to control and maintain the environmental conditions of a Stable Storage Facility, where it is essential to maintain temperature, humidity, and light intensity levels at the required settings. Consequently, we developed control applications and new pipelines to incorporate advanced analytics features. Further details of this work are available in the following journal publication.

Publication (attached as AP4)

***H. A. A. Chaudhary**, I. Guevara, A. Singh, A. Schieweck, J. John, T. Margaria and D. Pesch*
Efficient Model-Driven Prototyping for Edge Analytics
Journal of Electronics, vol. 12 (18), 2023.

Considering that the basic integration of EdgeX Foundry has already been validated with the platform, we have developed further DSLs to support the

advanced services of the EdgeX Foundry framework. This includes eKuiper, a rule-based engine for stream processing on edge devices, enabling real-time data filtering and transformation at the network's edge, making it a valuable tool for IoT and edge computing applications. Another analytics integration from a similar category is FiWARE, an IIoT platform for developing smart solutions. More details about this work are available in the following conference publication:

Publication (not attached)

I. Guevara, ***H. A. A. Chaudhary****, and T. Margaria*
Model-driven edge analytics: Practical use cases in Smart Manufacturing
in *ISoLA 2022, Leveraging Applications of Formal Methods, Verification and Validation. Practice, Cham: Springer Nature Switzerland, 2022, pp. 406–421.*

The final industrial use case that is a part of the discussion here pertains to the Safe Operation of Machines, where we observe the surrounding work environment and conduct risk assessments for worker safety. Several sensors and cameras were installed across the factory workyard to monitor working conditions on the industrial setup floor. This implementation is not entirely housed within a single platform, instead, it relies on different tools, such as the EdgeX Foundry platform as middleware for IIoT components, Tines for system notifications, Pyrus for data analytics, and the DIME platform for monitoring dashboards. The aim was to demonstrate how a complex system of systems could come together in quasi-realistic settings as presented in the following conference publication:

Publication (attached as AP5)

H. A. A. Chaudhary*, I. Guevara, J. John, A. Singh, A. Ghosal, D. Pesch, and T. Margaria*
Model-driven Engineering in Digital Thread Platforms: A Practical Use Case and Future Challenges
in *ISoLA 2022, Leveraging Applications of Formal Methods, Verification and Validation. Practice, Cham: Springer Nature Switzerland, 2022, pp. 195–207*

Research Impact

The validation of the platform through several industrial case studies showcased the practical aspects of the approach and demonstrated the maturity level 4 of TRLs. The developed DSLs cover a wide range of services within the IoT umbrella. In terms of relevance to the IoT user community, this constitutes a transformative contribution. Instead of requiring expertise in a multitude of diverse technologies, users only need training on these low-code platforms with a minimal learning curve due to simpler abstractions compared to traditional programming languages. After initial training, IoT application developers can effortlessly design, deploy, maintain, and evolve their applications in a unified environment provided by the platforms. The proposed approach can effectively support the development and deployment of small to large-scale IoT applications using multi-node ecosystem deployment.

4.2 Heterogeneity Within the DSLs

The encapsulation of the heterogeneity within the DSLs via uniform graphical representation in DIME & Pyrus makes it possible for the domain experts that by only learning three models of DIME and one model of Pyrus, they can create high-quality applications spanning across various domains and technologies without requiring a mastery of the underlying technologies, and programming languages, or communication protocols. Altogether, this work presents the successful integration of 19 independent technologies in two distinct Low-code development environments with the production of 26 processes and pipelines and the definition of 62 SIBs in 25 distinct DSLs.

4.3 Design Choices

Every design choice made during the development of the platform carries profound implications for its functionality, security, and resilience. Whether it involves selecting an underlying development platform for DTP, specifying communication protocols to ensure real-time data exchange, choosing hardware components for

robustness, or making decisions about AI/ML, every facet of design in this domain requires meticulous consideration. Some of the design choices are

- Before deciding on the shallow integration approach, which involves achieving interoperability via separate Docker containers for the initial integration, several other options were considered. For instance, in the case of R, some integration approaches were straightforward, e.g., R Caller libraries were relatively easy to set up. However, a limitation of this approach was that the supported platform needed to be installed on the same machine. This posed practical challenges since the application could potentially run on any webserver or even in a distributed environment at runtime.

- Another design choice we had to make was whether to deploy a single Docker container for ease of deployment or separate containers for each environment. In this case, instead of using a single container for all environments and the designed app, we opted for separate containers. This approach ensures that the failure of one environment, for any reason, does not affect the entire ecosystem. Additionally, the Docker environments are configured according to standard guidelines, with support for most standard libraries and some additional specific ones tailored to our use cases. They can easily be extended without any direct dependencies on our platform.

- In one of the case studies, the initial plan was to implement a live feed of monitoring cameras in the platform's dashboard. However, due to limitations in the supported data types of the underlying platform, snapshots had to be captured at regular intervals.

- While supervising a project related to student bursaries and the development of robotics applications on our platform, we encountered an issue with displaying data on the user interface from asynchronously communicating devices. To address this, we proposed a plugin that would automatically refresh the screen at regular intervals. Unfortunately, this feature was also not supported by the underlying platform.

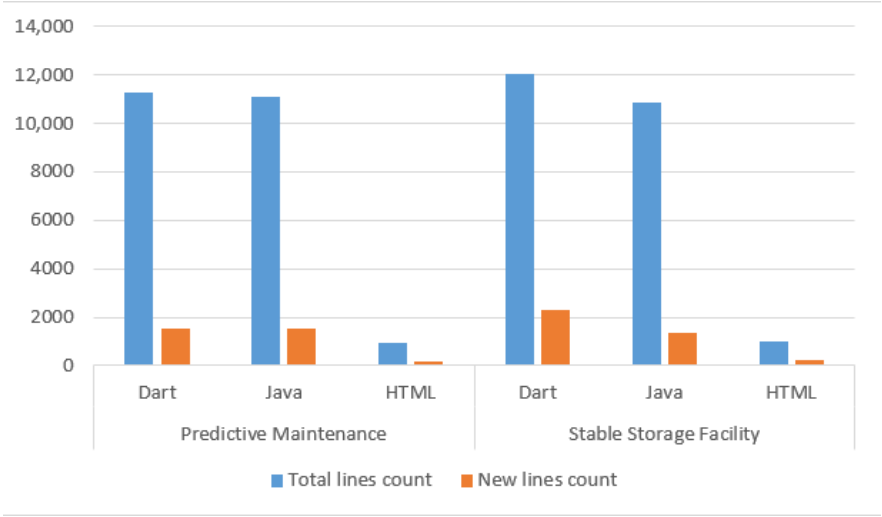


Figure 4.1: Reusability across DTP applications - Predictive Maintenance & Stable Storage Facility.

- At the beginning of the research work, the initial plan was to automatically support platform interoperability between two platforms: Pyrus and DIME. This involved automatically importing Pyrus pipelines into DIME. Initially, both underlying platforms were built upon the same CINCO version. However, halfway through the platform implementation, a newer version of CINCO was introduced and DIME was successfully ported to this new architecture, but Pyrus remained on the older architecture. Unfortunately, due to time limitations, the implementation of this interoperability as originally intended was not possible.

4.4 Benchmarking Code Generation

Embracing a platform mindset results in a single implementation of functionality with high reusability across multiple domains by very different domain experts. Additionally, a large part of the code is automatically generated from the models. To demonstrate these effects, two of the implemented case studies have been benchmarked: the stable storage facility presented in [33] and the predictive maintenance case study presented in [30]. Fig. 4.1 shows the count of total lines

	Predictive Maintenance			Stable Storage Facility		
	Dart	Java	HTML	Dart	Java	HTML
Total lines of code generated	11250	11096	943	12037	10889	1006
Total lines of code implemented	1517	1533	190	2304	1326	253
Percentage of code implemented manually	13.48%	13.82%	20.15%	19.14%	12.18%	25.15%

Table 4.1: Code stats for DTP applications - Predictive Maintenance & Stable Storage Facility.

of code generated (blue bars) for the implemented pipelines vs. the lines of code manually implemented (orange bars) as new SIBs. In table 4.1, we see that for all three languages (Dart for the frontend, Java for the business logic and HTML for the interface design), the new code written manually is a fraction of the total code. For Java in my use case, we have 1326 new lines of code in comparison with 10889 resulting from the automatic code generation, which also links to pre-existing SIBs. This means that we wrote only 12.18% of the new code needed for this application, and the data shows that this proportion is similar across languages and case studies. This comparison demonstrates that the dependency on software developers for new code is greatly reduced because we leverage the generation from models and the availability of the platform and prior DSLs.

4.5 Validation Through External Engagements

4.5.1 The Research Collaborations With Industry and Universities

The research was presented on multiple platforms, which facilitated the collaborations with University College Cork, the other CONFIRM hubs and also the industrial partners Tines, Johnson & Johnson and MongoDB. These collaborations enabled me to showcase my research, incorporate supported DSLs for the industrial libraries into the platform and test them in an industrial environment.

Several successful case studies mentioned in Section 4.1.3 were conducted as part of this collaboration, and one of the papers [5] was even shortlisted for the Best Paper Award.

4.5.2 Validation in the Education

The platform with rapid development with instant deployment capabilities has also been successful in teaching. Being the module leader and facilitator for teaching multiple courses in Immersive Software Engineering (ISE) and Computer Science and Information Systems at the University of Limerick, we used these topics as teaching case studies for nearly 200 students in classes with an average class size of 25, encompassing both undergraduate and postgraduate settings. These students have extended the supported capabilities of the platform and conducted numerous academic case studies in the domains of finance, business, and medical data analytics. The platform, along with its integration mechanism was also used in the curriculum of the CRT-AI[1] ML Research Week Summer School 2023 and the STRESS School 2021.

4.5.3 Student Bursaries

The platform also created opportunities for student bursaries both in Ireland and across Europe. Initially, the research was presented in CONFIRM's monthly R&D seminars. Through a collaboration with the Tyndall National Institute, CONFIRM funded two summer bursaries for undergraduate students in Ireland. I was in the lead role of designing the technical specifications of the projects in the bursaries and mentoring students during this program. These bursaries had a duration of 8 weeks. With an initial training period of 3 weeks, the students were able to integrate their project-specific libraries into the platform. The project focused on developing SIBs and DSLs for reconfigurable configurations of the Tyndall Smart Glove.

[1]The SFI Centre for Research Training in Artificial Intelligence aims to create an internationally connected and globally recognised centre of excellence for the training of postgraduate students and the up-skilling of industry-based staff in key technical topics in artificial intelligence and data analytics.

Furthermore, the tutorial presented in [29] enabled another bursary student from the National Institute of Applied Sciences of Rouen, France, to successfully integrate Matlab into the platform using a similar approach and to design DSLs for Matlab-related functionalities.

4.5.4 Public Engagements

As a team member of the CONFIRM Centre for Smart Manufacturing, my research was presented in the poster (III) presentation at the Tech Conference of ISE. Later a simplified version was also showcased to a general non-tech audience as part of the University of Limerick's Research Week, which had an outreach to over 1000+ people. The project was shortlisted as one of the top 10 projects to be displayed during the Research Week exhibition in 2023.

On the platform of ISE, I actively participated in an outreach program aimed at motivating and raising awareness of Information and Communications Technology (ICT) education among transition year students in the Limerick area. I took the lead in designing the curriculum and preparing the necessary teaching resources for these outreach programs. Alongside other instructors, I delivered the program. This year, we initially targeted four schools for a six-week program, encompassing 12 different classes and involving more than 250 students, with a gender ratio of 70% girls to 30% boys. After an initial demonstration of designing pipelines using the existing SIBs and their connectivity, the students were given mini-exercises, and an impressive 80–85% of them completed the tasks.

Additionally, we organized another public workshop titled "No-Code Data Analytics" as part of the "Limerick Lifelong Learning Festival," attracting participants from various domains. The exercises were categorized into three levels of complexity and difficulty. The response was overwhelmingly positive, with 90% of attendees completing two levels and 25% completing all three.

4. RESULTS AND DISCUSSION

5

Conclusion and Future work

The latest trends in Cyber-Physical Systems revolve around the increased adoption of cutting-edge technologies to enhance real-time interactions between digital and physical entities and improve their predictive capabilities. Addressing integration requirements and achieving interoperability within the context of Industry 4.0 require a platform approach to handle the heterogeneity of system-of-systems, storage capabilities, devices, and the overall ecosystem involved. This work proposed a model-driven Digital Thread Platform for CPS that provides a foundational architecture and systematizes the integration methodology using a layered DSL approach. In this work, we also discussed the principles, architecture, and specific aspects of our growing Digital Thread platform, which adheres to best practices in coordination languages. The presented approach covers integrations domain by domain, technology by technology, and platform by platform, ultimately offering a generalizable and highly reusable solution. By adopting state-of-the-art LCDEs like DIME and Pyrus, the platform supports a high level of reusability and analysis at the coordination layer, surpassing what is currently achieved with glue code.

Through numerous case studies, it is been demonstrated how complex systems of systems can come together in quasi-realistic settings. My initial research has paved the path for research collaborations within the CONFIRM Hub and the research groups in and across universities. The baseline architecture serves as a benchmark for colleagues to further extend the platform into domains such

as business organizational management [114], risk assessments [115], grammatical evolution [116], and more. In the follow-up, new projects have successfully acquired funding to further extend the platform in the domain of biomedical and cancer analytics. The rapid development capabilities along with built-in checks and one-click deployment have also proven effective in teaching agile development to undergraduate and postgraduate students, who have further extended its capabilities and conducted numerous academic case studies. The platform also generated opportunities for student bursaries within Ireland and across Europe, as students contributed to integrations in the domains of robotics and Matlab frameworks. Additionally, outreach sessions were conducted within the community for the general public and schools in Limerick, aimed at motivating and raising awareness of ICT education.

The platform's effectiveness has been demonstrated through various use cases, examples, and applications across multiple domains. It enhances the cohesiveness of heterogeneous systems and platforms, making them easier to understand, develop, and maintain, while also enabling domain experts to participate in the development cycle. This innovative platform approach has the potential to simplify application and systems development in the CPS domain, democratizing the development cycle and reducing the steep learning curve associated with managing multiple layers of knowledge. The belief is that this shift can significantly impact the ability of manufacturing experts to utilize advanced analytics and integration and interoperability platforms without requiring extensive coding and architectural expertise. The quality and completeness of the research outcomes are validated through empirical research within the open-source community and among adopters in academia and industry. This work with the ability to carry out agile modifications stems from an iterative and prototype-driven approach, where each cycle leads to improved releases of the previous content.

While the DTP platform still requires refinement, such as testing in distributed production environments, support for complex data types, addressing propagation delays, enhancing the user interface, and the aspects of commercialization. Similarly, the automatic interoperability aspects between the two platforms, as discussed in section 4.3, need to be addressed in future work. Although a fairly complete set of SIBs has been incorporated to encapsulate each

step's behaviour and provide a high-level abstraction for users, there are still many domains and technologies that need to be integrated on a case-by-case basis.

The project, in its current state, has been presented to the *Accelerating Research to Commercialisation Hub* for further work and is also shortlisted for funding for two years to extend its TRL from 4 to 7/8. The overall concept of the platform, along with its application validations, was included in the Research at Immersive Software Engineering (R@ISE) proposal at the University of Limerick, securing funding in collaboration with Analog Devices. The R@ISE project serves as a catalyst for the development and will lead research in both core (Low Code / No Code development platforms) and applied Software Engineering (Digital Thread). This paradigm is likely to appeal to sectors and industries that require close collaboration among diverse stakeholders, where skilled developers are scarce, and where faster turnaround times make code generation an attractive form of automation.

5. CONCLUSION AND FUTURE WORK

Part II

The Papers

6

Paper I (AP1)

Integration of Micro-Services as Components in Modeling Environments for Low Code Development

Hafiz Ahmad Awais Chaudhary and Tiziana Margaria

Publication Report

- **Publication Date:** 08/2021
- **Journal:** Proceedings of the Institute for System Programming of the Russian Academy of Sciences
- **Publisher:** Federal State Budgetary Institution of Science Institute of System Programming of the Russian Academy of Sciences
- **DOI:** https://doi.org/10.15514/ispras-2021-33(4)-2

Contributions Summary: *Conceptualization, all authors; underlying platform selection, all authors; literature review, H.A.A.Chaudhary; architecture design, H.A.A.Chaudhary; implementation, H.A.A.Chaudhary; testing, H.A.A.Chaudhary; writing—original draft preparation, H.A.A.Chaudhary; writing—review and editing, T.Margaria, H.A.A.Chaudhary; supervision, T.Margaria; project administration, H.A.A. Chaudhary and T.Margaria; funding acquisition, T.Margaria*

Abstract

Low code development environments are gaining attention due to their potential as a development paradigm for very large scale adoption in the future IT. In this paper, we propose a method to extend the (application) Domain Specific Languages supported by two low code development environments based on formal models, namely DIME (native Java) and Pyro (native Python), to include functionalities hosted on heterogeneous technologies and platforms. For this, we follow the analogy of microservices. After this integration, both environments can leverage the communication with pre-existing remote RESTful and enterprise systems' services, in our case Amazon Web Services (AWS) (but this can be easily generalized to other cloud platforms). Developers can this way utilize within DIME and Pyro the potential of sophisticated services, potentially the entire Python and AWS ecosystems, as libraries of drag and drop components in their model driven, low-code style. The new DSLs are made available in DIME and Pyro as collections of implemented SIBs and blocks. Due to the specific capabilities and checks underlying the DIME and Pyro platforms, the individual DSL functionalities are automatically validated for semantic and syntactical errors in both environments.

6.1 Introduction

Low code development platforms enable their users to design and develop applications with minimal coding knowledge [117], with the support of drag-and-drop visual interfaces that operate on representations of code as encapsulated code wrappers. The main aim [23] of these platforms is to produce flexible, cost effective and rapid applications in a model driven way. Ideally, they are adaptive to enhancements and less complex in terms of maintenance. Model-driven development (MDD) is an approach to develop such systems using models and model refinement from the conceptual modelling phase to the automated model-to-code transformation of these models to executable code [22]. The main challenges with traditional software development approaches are the complexity in development at large scale, the maintenance over time, and the adaptation to dynamic requirements and upgrades [117]. Doing this on source code is costly, and it systematically excludes the application domain experts. who are the main knowledge and responsibility carriers. At the same time, the cost of quality documentation and training of new human resources for code-based development are other concerns in companies and organizations that depend on code.

Domain Specific Languages (DSLs) conveniently encapsulate most complexities of the underlying application domain. Encapsulation of code and abstraction to semantically faithful representations in models empowers domain experts to take advantage of these platforms. They can develop products in an efficient manner and also meet the growing demands of application development without having deep expertise in software development. Based on a study [118] from 451 researches, the maintenance effort with low code platforms proved to be 50-90% more efficient as compared to changes with classical coding languages.

Software systems in general, and especially web apps in internet-centered ecosystems and digital threads in an Industry 4.0 context, are not isolated in nature: they demand interaction with various external systems, libraries and services. Frequent needs are (but not limited to)

- acquire sensors data from external systems,
- feed data to external dashboards for analytics and publishing,

- utilize the compute power of cloud systems,
- reuse sophisticated enterprise services.

In this context, microservices [119] play an important role at the enterprise level. The microservices paradigm (SOA done right) defines certain methods to design software services as a suite of independently deployable components with the purpose of modularity, reusability and autonomy [119]. Different versions of these services may coexist in a system as a set of loosely coupled collaborative components and must be independently replaceable without impacting the operations of heterogeneous systems.

This paper proposes the integration of microservices as components in two graphical modelling development environments based on formal models: the general purpose, desktop DIME [2] Integrated Modelling Environment and the special purpose, web based Pyrus(Pyro) [26]. Their extension and integration with external systems through services extend the capabilities of these platforms to meet wider communication needs (e.g. in the cloud), and also to take advantage of existing sophisticated enterprise services (e.g. AWS).

Low-code programming both at the API and the platform level is considered to be a game changer for the economy of application development. Gartner Inc., for example, predicts [120] that the size of the low-code development tools market will increase by nearly 30% year on year from 2020 to 2021, reaching a $5.8 billion value in 2021. They state that so far, this is the fastest and probably the simplest and most economical method of developing applications.

In this paper, Sect. 6.2 discusses the state of art, Sect. 6.3 states the problem, Sect. 6.4 gives an overview of the platforms used to extend the low-code DSLs. Sect. 6.5 explains the integration, architecture and implementation of SIBs in DIME (the desktop IME) and blocks in Pyro/Pyrus (the web IME). Finally, in Sect. 6.6 we conclude and discuss.

6.2 State of the Art

Most domain specific languages today are at the coding level and do not leverage a model driven approach at the platform level. The rise in re-usability and

maintainability demands paved the path to low code development environments and gained the attention of the developer's community [24]. The construction of meta-models behind these DSLs is challenging since they must capture all the domain knowledge, i.e. provide both semantic and syntactic rules. Ktrain [68] is a popular coding level DSL: a Python wrapper that encapsulates Tensor Flow functionalities and facilitates developers to augment machine learning tasks with fewer lines of Python code. Xatkit [121], still in the early stages of development, increases the reusability of chatbots by evolving NLP/NLU engine for text analytics. At the language level, they support several versions of bots, but the generation of chatbots from existing data sources at the framework level is in future plans. jABC [21] is a general purpose XMDD framework for the development of desktop and enterprise applications in model driven fashion. It enables its users to compose models by drag and drop of reusable blocks into hierarchical graph structures that are executable (interpreted) and compilable. Aurera [117] is a standalone desktop system for business modelling and addresses the challenges of frequent changes to IT solutions. The system is in the early stages of development and does not support communication with external systems. DIME [2] is a general purpose MDD platform-level tool, suitable for agile development due to its rapid prototyping for web application development. It follows the One Thing Approach based on XMDD [99], in a lineage of development environments that traces back to the METAFrame'95 [100, 122]. DIME supports both control flow and data flow modelling in its process diagrams. Control flow models admit a single start node but may have multiple end nodes, and nodes (called SIBs)representing single functionalities or sub-models are graphs, i.e. formal models. The SIBs are connected via directed edges depending on the business logic, with distinct edge types for dataflow and control-flow. Agent-based modelling paradigm [123] is another popular approach to increase the development productivity in simulation environments. CaaSSET [124] is a Context-as-a-Service based framework to ease the development of context services. The transformation into executable services is semi-automatic.

The market segment of web based development environments is still relatively young. Not having many established environments, there is a huge potential for

research and collaboration in this area. Theia [125], is a textual DSL tool supporting both desktop and web based IDEs. Pyro [26] is a web based graphical modelling environment for the collaborative development of web applications based on DSLs. Pyro, like DIME, is itself a product modelled with the Cinco [25] Meta Tooling Framework, which provides a suite of textual DSLs in which to specify the models for which to generate editors. The MGL ("Meta Graph Language") defines the structural information on the tool's model; the "Meta Style Language" (MSL) file specifies the visual characteristics (e.g. shapes and colors) of this model. The "Cinco Product Definition" (CPD) file specifies the details of the tool generation. Both DIME and Pyro are advanced graph model editors generated in this way from Cinco specifications. In this sense, they share a common philosophy, the semantic and syntactic characteristics of their respective models and edit/check/manipulate capabilities are described formally in their MGL, MSL and CPD files.

To interact with external entities, Micro service [126] is a popular way to develop modular, reusable and autonomous service components. We adopt this approach to extend the functionalities of two of the platforms in a model driven way. Following the same principles of graphical microservices architectures, AjiL [127] is a good effort in this direction, but due to performance delays in complex applications, they shifted their focus from graphical to textual notations.

6.3 Problem Statement

We consider the DIME [2] and Pyro [26] Cinco-products as our case study. Both are graphical Integrated Modelling Environments for low-code/no-code application development, used to develop research [13, 128] as well as industrial applications. We will use DSLs to virtualize the technological heterogeneity of the services, delivering a simple, coherent and efficient extension to both low-code modelling platforms.

Concretely, we show how to extend the capabilities of the DSLs through new, heterogeneous services. We

1. extend DIME, an offline eclipse-based general-purpose MDD environment for Web applications, by integrating a generic RESTful service as a new component, technically adding a new executable SIB that a) represents and b) executes this REST service;

2. extend Pyrus/Pyro, a collaborative, web based special-purpose MDD environment for data analytics and AI/ML, by integrating cloud-based enterprise services in a similar fashion. Here we chose Amazon Web Services.

The models in the 2 IMEs are different: DIME has rich models that cover processes, data, GUI, roles and security, and supports both dataflow and control flow models. Pyrus is simpler, and supports only dataflow modelling, which is popular and sufficient in the analytics pipelines it addresses. As the specific integration depends on the characteristic and expressive power of the models, there are differences.

The extension by integration adds to the tools the capability to communicate with sophisticated enterprise ecosystems, without sacrificing the flexible yet intuitive modelling style for the no-code users, who just use the DSLs that are available.

6.4 Overview of the IMEs

Domain-specific languages aim at minimizing the domain/IT knowledge gap between domain experts and software developers by lifting the vocabulary, granularity and structure of the application domain into the modelling language, so that the modelling entities stay familiar to the domain experts and their intuition is indeed correct. Domain experts prefer graphical languages because of the haptic functionality of drag-and-drop from a collection of functionalities is an apt metaphor for the construction of complex behaviours from an appropriate network of identifiable, well understood building blocks along intuitive control flow and data flow patterns.

The effort to develop these tools from scratch is enormous. Consequently, the specialization and evolution of such tools is hindered by the sheer cost and complexity of managing their code and its quality and support. Cinco [25] was

a game changer: a meta-level platform that wipes out this cost and complexity by providing the above described domain specific graphical modelling and code generation capabilities. Most Cinco products are based on Eclipse, enhanced with graphical modelling tools and various plug-ins. Suddenly, one can create a new Integrated Modelling Environment by specifying properties in three files and availing of the Cinco code generation capability for the target execution environment (e.g. eclipse or web). Modifications are not anymore at the code level: to change DIME or Pyro, one edits the specifying files and re-generates the tool with the appropriate generator.

In this paper[1], we will discuss the extension and integration of external systems as microservices in two of the Cinco's products DIME and Pyro (particularly Pyrus).

6.4.1 DIME

DIME is an Integrated Modeling Environment based on J2EE Eclipse, to design, develop and deploy web applications in an agile paradigm. Its model types help users to graphically model and develop different aspects of ordinary web application: (i) data model, (ii) GUI model, (iii) business logic in terms of processes and persistence, and (iv) roles and security model. The specific functional capabilities are provided to the users as a family of Graphical DSLs. The GUI DSL and a DSL providing a collection of generic blocks (called SIBs, for Service Independent Building Blocks) come with DIME, and other, domain specific DSLs can be added at need. Modelling in DIME happens mainly by the mechanism of drag-and-drop of DSL components on a canvas, and components comprise a node and a predefined set of outgoing edges. DIME supports both data and control flow to implement different aspects of business logic. Consistency checks are built-in in the DIME MGL and MSL, so that errors are either prevented (e.g., an output cannot connect to another output) or detected (e.g. the model is incomplete because some edges are dangling, not connected). DIME follows the One Thing Approach philosophy [98] by enforcing the SIBs to be generic and encapsulate

[1]The complete project code is available in Github: https://github.com/ahmadch1991/syrcose21

only the required functionality. This way, SIBs are easily understandable and reusable, and application experts that are not coders can develop complex applications by using the SIBs in the provided DSLs. GUI models represent single pages of the web application and links to the underlying functionalities. Process models can be hierarchical, i.e. contain other process models, this way easing the organization and comprehension of the structure and behaviour of complex applications. Once the models are ready, the product generation step feeds the models collection to successive model-to-model and model-to-code transformers, resulting in a complete code generation for a standard web application runtime.

The setup for the development environment for DIME requires Java version 1.8 and Eclipse dependencies to be installed on the development machine.

6.4.2 Pyro

In contrast to DIME, Pyro is a web based Cinco-product that runs in a web browser and turns it into a collaborative domain-specific graphical modelling environment for data-flow applications. Pyro stores objects and data types in a loosely coupled manner [101]. To incorporate the rich features of typical web application, like the built-in support of cross-platform and a reusable components focused architecture, its front-end is built upon the Angular Dart [129] framework. To meet the needs of uninterruptible user interaction with the modelling environment, data exchange is implemented via non-blocking REST-based asynchronous communication. As a more recent development, Pyro is being enhanced with performance optimization and integration of external systems.

Pyrus is a specific Pyro derivative specialized for dataflow models executing within the popular Jupyter Notebook environment. It is therefore particularly attractive for data analytics and AI applications, that are frequently coded in Python.

Working with Pyro/Pyrus requires the platform deployment on a local or remote server accessible via browser.

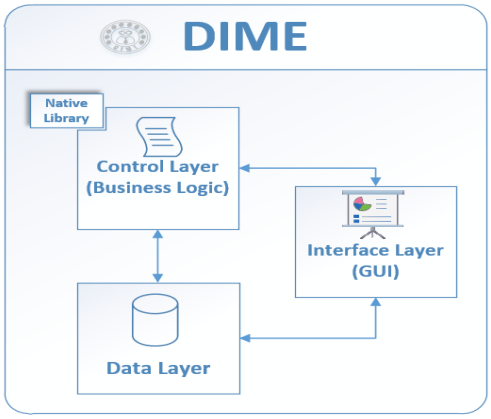


Figure 6.1: DIME: modelling architecture and native library support

6.5 Extending the IMEs

We show now how to extend DIME and Pyrus with RESTful services and cloud-based AWS services, respectively. This happens by implementing a new DSL consisting of a collection of capabilities that run on an external platform in a different technology. Effectively, these are akin to microservices. We show here exemplarily how to implement one such microservice for each case. The extension to other RESTful services or other AWS or similar services is then easy to achieve following these blueprints.

6.5.1 RESTful Extension of DIME

We show now how to develop a generic Service Independent Building Block (SIB) in DIME that communicates with any external RESTful system.

Extending the DIME functionality happens by using the support of the native library it provides. In DIME's multi-model type architecture, the business/logic model type is where the new SIBs will be utilized. As shown in Fig. 6.1, the existing model architecture is extended with the addition of a native library as a new block belonging to the process/business logic model type. The native block will be merged to the process/business logic models during the automated code generation phase for the web application. Concretely, the extended functionality will be integrated as Java code with the remainder of the application during the

compilation, and this way it will not add any additional performance penalty.

```
package app.demo
sib rest_read_str_list : file_path#Java_fn
    url : text
    input_var : text
    input : text
    output : text
    -> success
        output: [text]
    -> noresult
    -> failure
```

Listing 6.1: SIB declaration for the "REST Read" SIB

For the SIB implementation, we consider here a REST service that acts as a server and returns a list of country names on the basis of a country code input, e.g. United Kingdom for input 'uk', and the name of all countries from the database for input 'all'. The service is implemented in PHP in a conventional fashion, and deployed on an external public server. It will respond to the requests generated by client SIBs

Now, we need to create a new client SIB with appropriate characteristics to communicate with RESTful service. This encompasses the SIB declaration and the SIB implementation.

The SIB declaration is shown in Listing 1.

- Firstly, in the project explorer we add a new, empty file with the extension ".sib" and the name of the proposed SIB.
- This SIB declaration file contains the signature of the new SIB. It starts with the keyword "sib", followed by the new SIB name, that in our case is REST_read_str_list, followed by a colon and the path to the attached Java function. This is the function be invoked when the SIB is used in the process modelling.
- The next section contains the proper signature: the list of inputs and outputs, with name and data types. In our case, the SIB accepts the following I/O:

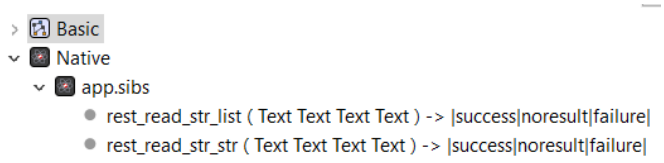


Figure 6.2: SIBs explorer with the new native SIBs

  - URL of an external server
  - input variable name and data to create a valid URL at run time.
  - the output variable name is also added in the signature, to extract the requested data from server response for further JSON parsing.

- finally the list of different control branches based on outcomes. In our case the three branches are "success", which returns a text output provided by the external service, "noresult" of the external services returns no result, and "failure" in case of error in the communication with the external service.

For the SIB implementation, The RESTful "Rest Read" service is implemented in PHP in a conventional fashion, and deployed on an external public server. It will respond to the requests generated by these SIBs.

Once the declared SIB, its signatures and the attached Java function are validated by the platform, the SIB will be visible in the explorer as a Native SIB, with the other default SIBs as shown in Fig. 6.2. At this point, it is ready to be used, and available to the DIME users as a drag and drop item, ready to be inserted in any process model.

Fig. 6.3 shows the visual representation of the newly developed SIB, as it appears when it is used in a process model. The required four inputs are being fed to this block using data flow (dotted) arrows. We see the three outgoing branches, labelled as defined. On success, the result will be conveyed as a string (or list of strings) to the successive SIB.

DIME automatically validates semantic and syntactic errors after the insertion and data connectivity of SIBs, ensuring this way the correctness of intended behaviour (automatic quality assurance [106] of models).

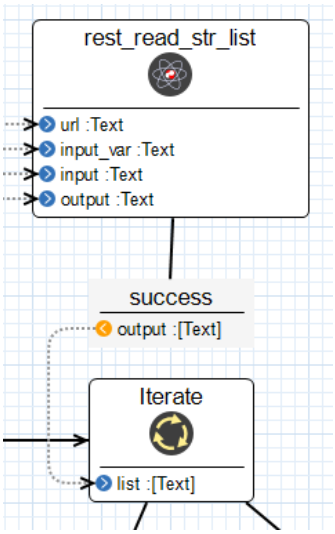


Figure 6.3: The REST read SIB in use: visual representation in a model

6.5.2 Cloud Extension of Pyrus

We extend now the Pyrus is an online data analytics platform built using Pyro with Amazon Web Services (AWS), choosing the Amazon Translate service[130]. Pyrus communicates with the Jupyter hub at the backend. It uses the RESTful protocol to read function signatures and execute the attached Python code. As shown in Fig. 6.4, Jupyter and Pyrus communicate in an asynchronous manner.

The mechanism for the new AWS Translate block definition and implementation is similar to the DIME SIB declaration, but it only contains the signature, and no outgoing branches. As Pyrus supports a dataflow modelling style, there are no control elements (the branches). The signature declaration starts with the # keyword, it is followed by meta data and the implementation of the functions in a python file. It has an extension ".py" and is located in the Jupyterhub space. Pyrus automatically reads these annotated signatures and shows them as drag-able blocks in its explorer.

Fig. 6.5 shows the working pipeline of AWS_translate: in reality, we have defined 2 blocks, AWS init_session and AWS translate_string. The workflow is in fact logically divided in two phases: initialization and implementation.

The initialization block must meet the preconditions of the external server in order to use its services. Communication with the AWS server/services requires

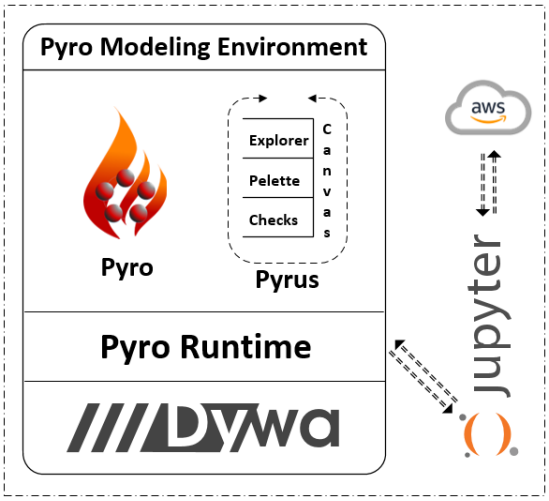


Figure 6.4: The Pyrus/Pyro architecture extended with AWS

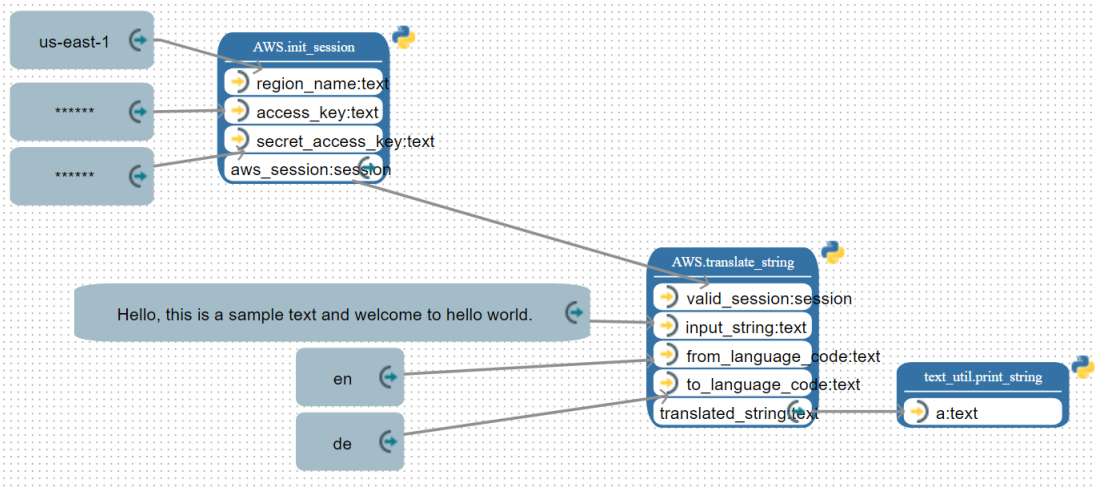


Figure 6.5: Pyrus pipeline using AWS_translate

a valid session, validated with credentials, i.e. access key, secret key, and server information. The "AWS.init_session" initiates the communication transaction with the AWS server. It accepts the required inputs/tokens from connected grey blocks, which are constants.

Once successfully authenticated by AWS, a session token is provided for further communication with AWS. This token (output) is fed to the "AWS.translate_string" block along with the other required inputs: the text string to be translated and the code of the from and to languages.

Finally, the (translated text) result is passed to the next block, "text_util.print_string", that prints it on screen.

The pipelines are automatically validated by the underlying modelling platform to check for connectivity errors of the blocks on the canvas.

6.5.3 Tool and Technologies

The tools and technologies used for these implementations and extensions are Eclipse, Java, JSON library, PHP, Python, DIME, Pyrus, Jupyter Hub and Amazon Web Services.

As the methodology is generic, it can be followed like a blueprint to implement communication and integrate a large variety of external services and platforms. The resulting drag and drop components enrich the DSL domain and expressive features of low code development in the mentioned platforms.

6.6 Conclusion and Discussion

We presented a generic extension mechanism to two low code development environments along a microservice philosophy. We showed it by integrating preexisting remote RESTful services and cloud-based enterprise system services as new drag and drop components in the respective DSLs. In DIME, an offline low-code IME, we used the native library mechanism, with signature declaration, linked Java backend code, and the code is merged with the logic layer at compile time. Pyrus, an online no code graphical data analytics tool, is linked with Jupiter Hub for functions discovery and code execution. To display new Python functions as components in Pyrus, custom signatures are added to the Python files defined in the Jupyter hub, and the data flow pipeline of the service is modelled in the Pyrus frontend.

The simplicity and generality of the integration are an important feature of the chosen platforms. We envisage in fact a systematic integration of DSLs for various application domains stemming for our research collaborations. The simpler this is, the easier is the adoption of the approach across diverse application

domains, research groups, and industrial partners. The (hand)code based extension approach of most popular low-code environments, that do not use formal models, nor generate "intelligent" modelling domains that have built-in checks for the model conformance are in fact inferior and sources again of complexity in the management of heterogeneity, code maintenance and evolution. The next application domains will be data visualization and data streaming platforms. We will support more AI/ML and data analytics functionality both in DIME and in Pyrus, adding also cross-platform integration, in order to use the analytics capabilities of Pyrus pipelines in the DIME Digital Twin applications for Industry 4.0 as well as in the Digital Humanities [131].

Acknowledgment

This work was supported by the Science Foundation Ireland grants 13/RC/2094 (Lero, the Irish Software Research Centre) and 16/RC/3918 (Confirm, the Smart Manufacturing Research Centre).

7

Paper II (AP2)

The Interoperability Challenge: Building a Model-driven Digital Thread Platform for CPS

Tiziana Margaria, Hafiz Ahmad Awais Chaudhary, Ivan Guevara, Steve Ryan and Alexander Schieweck

Publication Report

- **Publication Date:** 10/2021
- **Conference:** International Symposium on Leveraging Applications of Formal Methods
- **Venue:** Rhodes, Greece
- **Book:** Leveraging Applications of Formal Methods, Verification and Validation. Practice
- **Publisher:** Springer International Publishing
- **DOI:** https://doi.org/10.1007/978-3-030-89159-6_25

Contributions Summary: *Conceptualization, All authors; technical Support, H.A.A.Chaudhary, A.Schieweck; architectural inputs, H.A.A.Chaudhary, A.Schieweck; architecture design and implementation, H.A.A.Chaudhary; domain specific integrations, H.A.A.Chaudhary, I.Guevara, A.Schieweck and S.Ryan; case studies, H.A.A.Chaudhary, I.Guevara, A.Schieweck and S.Ryan; writing—original draft preparation, H.A.A.Chaudhary; writing—review and editing, T.Margaria and H.A.A.Chaudhary; supervision, T.Margaria; project administration, H.A.A. Chaudhary and T.Margaria; funding acquisition, T.Margaria.*

Abstract

With the heterogeneity of the industry 4.0 world, and more generally of the Cyberphysical Systems realm, the quest towards a platform approach to solve the interoperability problem is front and centre to any system and system-of-systems project. Traditional approaches cover individual aspects, like data exchange formats and published interfaces. They may adhere to some standards, however, they hardly cover the production of the integration layer, which is implemented as bespoke glue code that is hard to produce and even harder to maintain. Therefore, the traditional integration approach often leads to poor code quality, further increasing the time and cost and reducing agility, and a high reliance on individual development skills. We are instead tackling the interoperability challenge by building a model driven / low-code Digital Thread platform that 1) systematizes the integration methodology, 2) provides methods and techniques for the individual integrations based on a layered Domain Specific Languages (DSL) approach, 3) through the DSLs it covers the integration space domain by domain, technology by technology, and is thus highly generalizable and reusable, 4) showcases a first collection of examples from the domains of robotics, IoT, data analytics, AI/ML and web applications, 5) brings cohesiveness to the aforementioned heterogeneous platform, and 6) is easier to understand and maintain, even by not specialized programmers. We showcase the power, versatility and the potential of the Digital Thread platform on four interoperability case studies: the generic extension to REST services, to robotics through the UR family of robots, to the integration of various external databases (for data integration) and to the provision of data analytics capabilities in R.

7.1 Introduction

The manufacturing industry in the context of Industry 4.0 demands automated and optimized production lines and is moving towards connected and smarter supply chains processes [35, 36]. Cyber Physical Systems (CPSs) are core building blocks of future factories [9] and researchers believe that, with the emergence of systematic industrial integrations of ICTs and external information systems, CPSs will contribute towards "smart anything everywhere" in particular also smart cities and smart factories [132]. This medium-to-large scale industrial integration implies interoperation of interconnected, heterogeneous virtual and physical entities and devices towards a shared goal [10]. Interoperation includes real time data from machines, production lines, IoT devices, networks, programmable logic controllers and external systems into smarter, connected manufacturing systems [133].

In this context, the integration and interoperability among all these entities is a key challenge for the success of Industry 4.0. Due to architectural convergence, the holistic integration challenge can be organized in three levels [134]:

1. **Physical Integration**, handling the connectivity and communication among devices.

2. **Application Integration**, dealing with the coordination and cooperation among different software applications and data stores.

3. **Business Integration**, covering the collaboration between different functions, processes and stakeholders.

In this context, considering the "reprogrammable factory" vision brought forward within the CPS Hub of the Confirm research centre [13] and the high-level depiction in Fig. 7.1, we find a broad correspondence between the three layers above and the three layers implicit in the picture. The Digital Twin is a "sosia" of any individual component, software or process, and the Digital Thread is a fitting analogy for the role played by any integration and interoperability layer delivering that ability to communicate and cooperate. Ideally, the digital thread should not be provided through a myriad of scripting quick fixes, nor through a

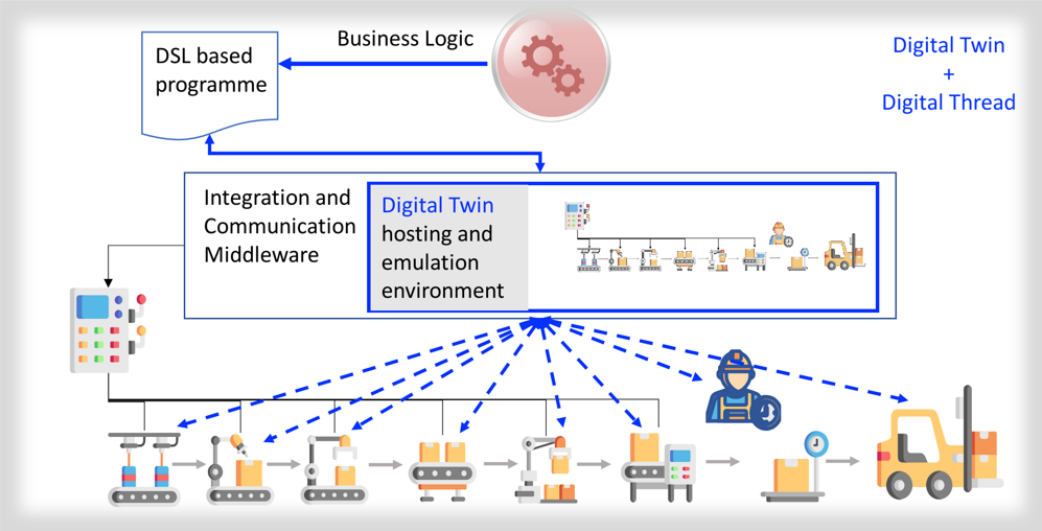


Figure 7.1: Confirm HUB CPS – the reprogrammable factory vision (source: Confirm HUB2)

vast patchwork of bespoke technologies, that may be adequate to serve individual point-to-point interfacing needs, but become a nightmare to understand, test, validate, manage, and evolve.

In Fig. 7.1, the Digital Thread is the collection of blue lines (solid or dotted), that manage the communication and interoperation between the **Business layer** (at the top), the integration and communication middleware and their platforms (like e.g., EdgeX foundry) as well as the Digital Twins, both at the **Application layer** (in the middle), and the myriad devices, machines, sensors, dashboards, and more at the **Physical layer**, which may also include software for SDNs, SCADA, analytics, AI and ML, and more.

If properly provided and managed, these many heterogeneous vertical and horizontal integrations can enable CPSs to leverage the many advances in industrial systems, big data, AI/ML and cloud computing systems. This way, the seamless integration needs advocated by leading technology providers, vendors, and end-users [135] can be fulfilled.

The IEEE defines interoperability as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged" [60]. The author highlighted this need and introduced best prac-

tices to develop smarter applications rather than fragmented applications [136]. Literature shows that five categories of interoperability can have quite different arrangements [137]:

1. Device Interoperability
2. Network Interoperability
3. Syntactical Interoperability
4. Semantic Interoperability
5. Platform Interoperability

The effort of building interoperable systems is an outstanding challenge in the adoption of new technology. The integration layer is too frequently neglected and left for developers to solve as a side issue. This means that a number of experts are required over and over again to reprogram these complex systems in accordance with evolving needs and standards. This is a time-consuming and expensive task [14], and such systems are hard to produce and difficult to maintain. The author [15] also concluded that manual integrations between APIs reduce agility and that inaccuracies in the integration may also lead to financial losses and unexpected delays in production. CPSs are typically embedded into a more complex system via interfaces, so modularity (plug and play) and autonomy are important enablers to adapt the upgrades and reconfigurations effectively, in accordance with rapidly changing customer needs [16]. Trustworthy interoperability both at the vertical and horizontal level is critical for setting up Industry 4.0 operations [135].

Model-driven development (MDD) is an approach to develop complex systems using models and model refinement from the conceptual modelling phase to the automated model-to-code transformation of these models to executable code [22]. The main aim [23] of MDD is to produce flexible, cost effective and rapid applications, that are adaptive to enhancements and less complex is terms of maintenance. Achieving this on the basis of direct source code editing is costly, and it systematically excludes the application domain experts, who are the main holders of domain knowledge and carriers of responsibility. At the same time, the

cost of quality documentation and training of new human resources for code-based development are other urgent concerns today in companies and organizations that depend on code.

For an adequate, scalable and possibly general and evolvable solution to the interoperability challenge, we propose instead to use adequate, modern software platforms based on model driven development concepts, paying attention to choose those that best support a) high assurance software and systems, b) a fast turnaround time through agility and a DevOps approach, and c) an inclusive understanding of the stakeholders, where few are professional coders. Therefore we adopt a low-code application development paradigm, combined with code generation and service orientation.

The paper is organized as follows: Sect. 2 introduces the digital thread concept and its relation with interoperability. Sect. 3 discusses the low-code environment platform we use to develop the digital thread platform itself. Sect. 4 describes the current status of the platform: latest integrations, ideas and enhancements that benefit the bootstrapping of components in the smart manufacturing domain. Sect. 5 addresses the specific questions posed by the Special track organizers. Finally, Sect. 6 concludes and sketches the planned future work.

7.2 Digital Thread In The Middle – Interoperability.

Digital Twin and Digital Thread are two transformational technological elements in digitalization of the Industry 4.0 [61]. The Digital Twin covers individual aspects of physical assets, i.e., their virtual representation, their environments and the data integrations required for their seamless operations. Digital Twins and AI models are the two kinds of models that the manufacturing Industry has meanwhile accepted as useful. However, they are not the only ones. A Digital Thread connects the data and processes for smarter products, smarter production, and smarter integrated ecosystems. In the modern era, the Digital Thread provides a robust reference architecture to drive innovation, efficiency and traceability of

any data, process and communication along the entire system (or system of systems') lifecycle. This is a new, much more structured and organized way to look at integration and interoperability. It is unfamiliar to the manufacturing world, and it is also still unfamiliar to many in some software engineering communities.

For this new paradigm to enter the mainstream, systems and their models need to be connected through **an integrated platform** for automatic data and process transformation, analysis, generation and deployment that should be able to take systematic advantage of the formalized knowledge about the many immaterial and material entities involved. Referring to Fig. 7.1 again, data and operations from and to any of the heterogeneous elements (component, subsystem) in the picture, should be mediated (i.e., adapted, connected, transformed) through the Digital Thread platform, which becomes both the nervous and circulatory system of the overall system:

- The nerves, as whatever is sensed needs to be sent to the decision systems and the commands are then relayed to the actuators.

- The circulatory system, as plenty of data is moved in order to "nourish" the information-hungry services that store, aggregate, understand, and visualize what happens in the system, increasingly in real time or near-real time.

The choice of which concrete IT system to adopt for this central role is not an easy one, and it is not a choice that can be amended or reversed easily later on. The properties of the Digital Thread will depend very intimately on the characteristics and features of the IT platform on which it is based: whatever the IT platform does not support will be difficult to overlay a posteriori, and whatever is easy in that platform will likely be adopted and become mainstream for the community of users.

Bearing in mind all the desired characteristics, we chose DIME [2] as the IT platform of choice underlying the Digital Thread solution.

7.3 The Underlying Low-Code Development Environment

DIME is an Eclipse based graphical modeling environment developed with the Cinco SCCE Meta Tooling Suite [25] It is a low-code application development environment that follows the philosophy of OTA (One Thing Approach) [98] and the eXtreme Model Driven Development paradigm [99, 138] to support the design, development, and deployment of (originally web) applications in an agile way. DIME empowers application domain experts who are not proficient coders/programmers to fully participate in the entire design, development and evolution process because it supports easy modelling, done by means of drag and drop of pre-existing components. For the separation of concerns, DIME supports several model types that express distinct perspectives on the same comprehensive model. This "write once" rule is the essence of the coherence by construction principle central to the One Thing Approach. The DIME model types encompass:

- A **Data model**, which covers the persistence layer (both types and relations) of the application in a form similar to a UML class diagram.
- A collection of **Process models**, that define the business logic on the basis of internal and external libraries of basic functionalities provided by means of the Native DSL mechanism. Each DSL exposes a collection of SIBs (for Service Independent Building blocks), that are reusable, instantiable and executable modeling components with either an associated code implementation or an associated hierarchical process model.
- A collection of **GUI models**, defining the elements (look and feel, actions and navigation) of the pages of the web application, and
- **Security and Access control models**, mainly handling the security and access permission aspects of the application.

This is different, for example, from the standard UML models [139]: UML and related tools support a variety of different model types (static, like UML class diagrams and DIME's Data model, and dynamic like UML's activity diagrams

and DIME's process models) serving different purposes, but those models/model types are not connected among each other. Therefore, it is very easy in UML to breach consistency of the overall model collection, because changes do not propagate from one model to the other.

We value DIME's characteristics of open source, flexibility, ease of extension, support of high-assurance software quality, agility, service-oriented approach, and containerization. For the specific low-code support, its model-driven approach is based on DSLs at two levels:

1. **Language DSLs**, as a mechanism to design and implement the application design environment itself, i.e., the Integrated Modeling Environment (IME), and

2. **Application domain DSLs**, at application design time. We want to use **Native DSLs** as the means to integrate and expose collections of capabilities offered by end devices and other sources of functionalities to the application designers, and **Process DSLs** as the means to foster the reuse of medium and large grained business logic across applications.

As different models cover different aspects of the target application, to ensure the intended behavior each model of the application is validated at compile time both at DSL and platform level for syntactic and semantic errors. After validation, these models act as input for subsequent model-to-code transformation phases. The key design principles of DIME are simplicity [104], agility [105] and quality assurance [106], hence, DIME is a promising "game changer" low code development environment (LCDE) for the realization of sophisticated web applications in tremendously shorter development cycles.

7.4 Digital Thread Platform: The Current Status

We target the application domain of advanced manufacturing including manufacturing analytics. Accordingly, we intend to support the conception, design and

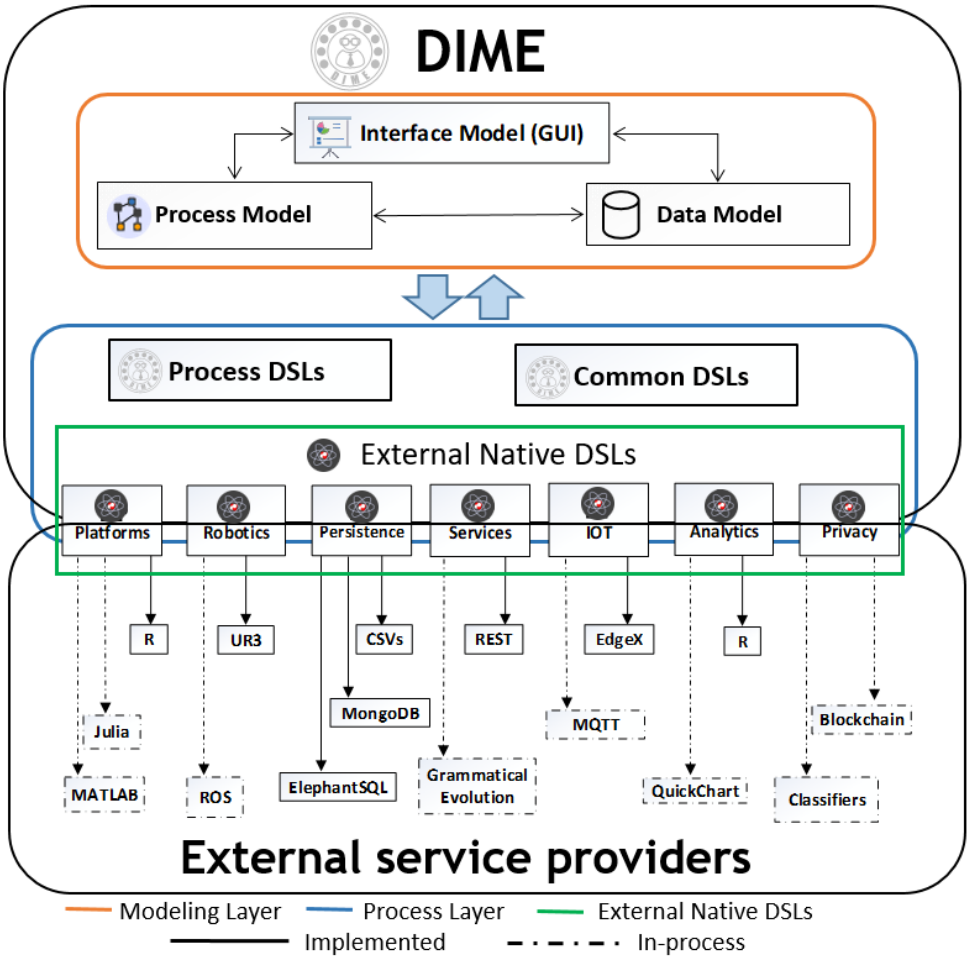


Figure 7.2: Architecture overview of DIME and custom DSLs

implementation of a set of applications, like, for example, robotics navigation and control, proactive maintenance, MES monitoring, but also analytics dashboards that analyse or summarise in real time or near-real time data provenient from various systems and subsystems of a complex, possibly distributed production plant. In this context, data, processing and communications are expected to concern a large variety of devices, data sources, data storage technologies, communication protocols, analytics or AI technologies and tools, visualization tools, and more. This is where the integration of external native DSLs plays a key role. The current architecture of the Digital Thread platform is depicted in Fig. 7.2.

We see that DIME's **Language DSL**, used to design the applications, en-

compasses for the moment in the advanced manufacturing setting primarily the Data, Process and GUI models.

We also see that already a significant variety of external platforms (like EdgeX for IoT), technologies (like REST services, or R for analytics) and tools (like the UR family of robots) have been integrated. All these are part of the **Application DSL** layer mentioned in Section 3, including quite a number of **Native DSLs** external to DIME.

The central property of simplicity here is that, once integrated, the Native DSLs all look "alike" within DIME: the collection of individual functionalities has its own, but uniform representation, and their use within DIME is uniform as well. This means that once a DIME user has learned how to work with the three model types and with the basic functionalities, this user can produce high quality applications that span a variety of technologies and application domains without the need to be able to master any of their underlying technologies, programming languages, or communication protocols, as these are part of the encapsulation of this heterogeneity within the DSLs, and its virtualization by means of the uniform representation and handling. Note that this approach is not completely unusual: with more or less success, generations of platforms have pursued this goal. Some platforms are domain specific and special purposed like for example, EdgeX [4] for the provision of an extensible, uniform service-oriented middleware for (any of the) supported IoT devices and their management. EdgeX defines itself as "the preferred Edge IoT plug and play ecosystem-enabled open software platform" [4], a "highly flexible and scalable open-source framework that facilitates interoperability between devices at the IoT edge". Its data model is YAML profiles, its exposed services are implemented as REST microservices, it supports the C and Go programming languages for users to write their own orchestrations (instead of DIME's process models). It does not support GUI models as these user interfaces are not an aspect in their focus. Other platforms have a broader scope. For example, GAIA-X [140] aspires to become "a federated data infrastructure for Europe". Among the platforms that have meanwhile over a decade of history, FIWARE [141] describes itself as "the Open-Source Platform for Our Smart Digital Future" and offers a wide collection of services and service components that can be reused by application developers.

They all require programming ability, none of them offer a low-code approach, they all provide collections of reusable components, and do not envisage support for the orchestration on top. Their view is the bottom-up approach of component provision, that an expert will then somehow orchestrate.

In this respect, our value proposition sits clearly at the upper, application development layer, where we see the interoperability challenge truly resides.

We also see ourselves as systematic users of such pre-existing platforms, who are for us indeed welcome providers of Native DSLs. In this context, a number of integrations in DIME relevant to the advanced manufacturing domain have already been addressed.

Seen from an **Application domain** point of view, for example, the following have already been integrated:

- A IoT through (some parts of) EdgeX [111, 112]
- Robotics through the UR command language [13]
- Persistency layer through various data storage alternatives, from CSV files to relational (PostgreSQL) and no-SQL (MongoDB) databases (own work)
- Cloud services [28]
- Data analytics with R libraries (own work)

and own work is ongoing on

- some forms of AI and Machine Learning (classifiers, Grammatical Evolution [142], and more)
- Robotics through ROS, additional to [13, 128, 143]
- Distributed Ledger Technologies through blockchain
- Visualization tools with, e.g., Quickchart

Seen from a **Technology portfolio** point of view,

- REST services [28]

Table 7.1: SIBs information for REST services integration.

SIB Name	Input(s)	Output(s)	Explanation
Rest_read_str_str	Url Input_var Input_val Output_var	Output_val (single var)	This SIB accepts inputs required to initiate the communication with an external REST service and receives a single variable as a result.
Rest_read_str_list	Url Input_var Input_val Output_var	Output_val (array/list)	This SIB accepts inputs required to initiate the communication with an external REST service, receives the result as an array or a list and feeds it to the subsequent SIB for iterative processing of the elements.

- R, seen as a programming language (own work)

are already supported, and the next months will see own work on

- Matlab and Julia, as programming languages/tools for simulations
- MQTT and other native IoT protocols, as in some cases it is impractical to have to use EdgeX.

In the following, we will provide some details on a few selected examples of these integrations.

7.4.1 REST Services

A case study [28] details a generic extension mechanism, where two LCDE platforms based on formal models were extended following the analogy of microservices. This extended the capabilities of DIME by integrating cloud and web services through REST. RESTful APIs are a standardized way how applications can communicate, firstly described by Roy in his PhD Thesis [144], and have become one of the most used APIs schemas. DIME uses REST to share information

between the front and the back end. While the commands are encoded via the widely supported HTTP standard, data can be exchanged in many formats. The most common data format is the JavaScript Object Notation (JSON), but also Extended Markdown Language (XML) and others can be used.

In this context, this new DIME DSL allows to act as a client for those APIs, i.e., to send requests to external applications and to decode JSON responses into the data domain of DIME. Table 1 shows a list of sample SIBs with relevant IOs and explanations.

7.4.2 Robotics With the UR Language

UR3 is a well-known lightweight collaborative robotic arm designed to work on assembly lines and production environments in the smart manufacturing context. The robotic arm is not only easy to install but has a simple command language to program all the tasks required, with a tethered tablet. The paper [13] showed how to build a remote controller through a DIME Web application that manages the remote communication with UR cobots and the commands through a UR-Family native DSL. Fig. 7.3 shows the hierarchical process model in DIME for the outer working of the controller: the robot is initialized (started and ready to respond), it is sent to an initial position to test the correct functioning of the command channel, then the program with the real task is uploaded (this is itself a DIME a process) and the communication is then closed upon execution completion. Table 2 shows a list of sample SIBs with relevant IOs and explanations.

7.4.3 Data Management via Files and External Databases

DIME supports basic file handling operations, sufficient for text and Comma Separated Value (CSV) files. However, handling large datasets requires coordination with dedicated structured or non-structured databases. Recent work integrates MongoDB Atlas and Elephant SQL, two fully managed NoSQL cloud databases, and the PostgreSQL database service. The integrations use the MDD approach to provide functionalities to import and export data from/to these storage alternatives - an essential capability for data interoperability and data migration

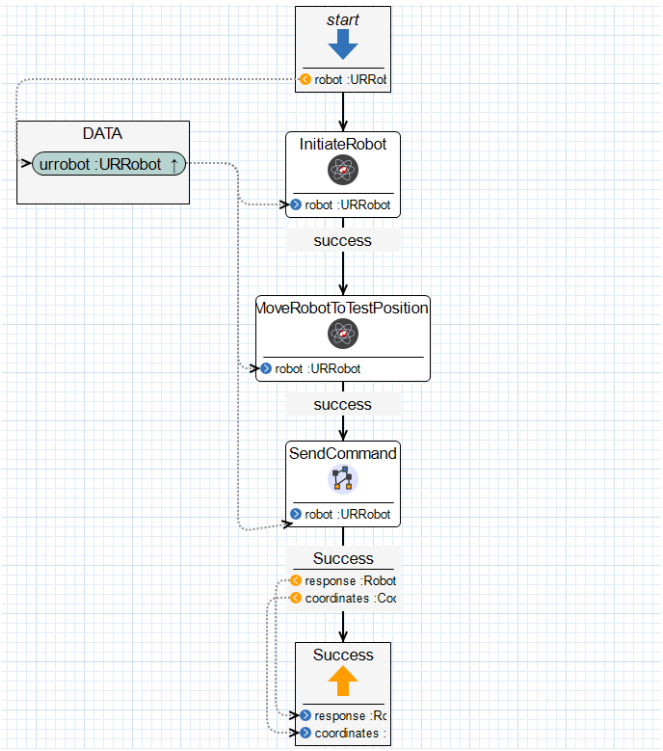


Figure 7.3: DIME process for the UR robot position control

in the Digital Thread platform. Table 3 shows a list of sample SIBs from the MongoDB integration with the relevant IOs and explanations.

7.4.4 Analytics With R

DIME is built upon J2EE and supports all its functionalities and capabilities. However, specialized languages and platforms like MATLAB for simulations and R for data analytics are optimized for those tasks and need to be supported in a proper Digital Thread platform. We recently extended DIME with the R environment by encapsulation through a Native DSL shown in Table 4. Fig. 7.4 shows the runtime architecture: the application and the R environment are deployed in two different docker containers. The Rserve library is the entry point of the R environment, it handles all the external communication using TCP/IP. DIME uses this mechanism to provide the R data analytics capabilities.

The impact of having a platform mindset is that the functionality needs to be

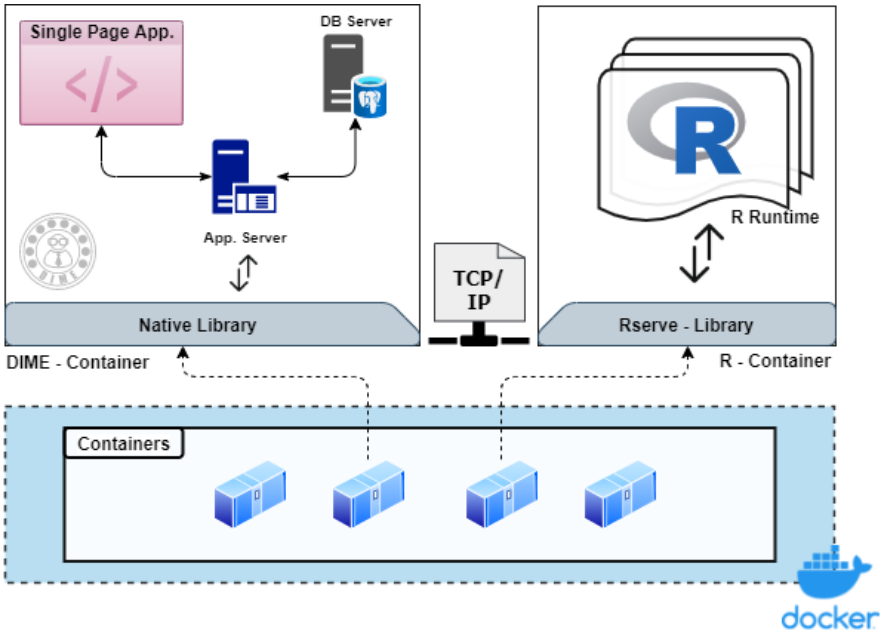


Figure 7.4: Runtime infrastructure of DIME and R - environment

implemented only once and is reusable across multiple domains by very different domain experts, as illustrated in Fig. 7.5 and Fig. 7.6. The same plot_R_histogram SIB is used in fact in Fig. 7.5 (left) with a manufacturing domain dataset to draw the histogram of manufacturing fitting failures per installation year (left), and in Fig. 7.6 on the Irish census data of 1901: in this history/humanities domain the same SIB is used to visualize the breakdown of the 1901 population by age.

7.5 Programming: What's Next?

Considering the questions posed to the authors in this Special Track, we answer them briefly from the point of view of the technologies described in this paper, considering also our experience in projects and education.

1. *What are the trends in classical programming language development? both wrt. applications programming and systems/embedded programming?*
   While the state of the art in these domains is still dominated by traditional, hand-coded software, the low-code development wave is reaching industry adoption and a certain degree of maturity. So far it is more prominent in

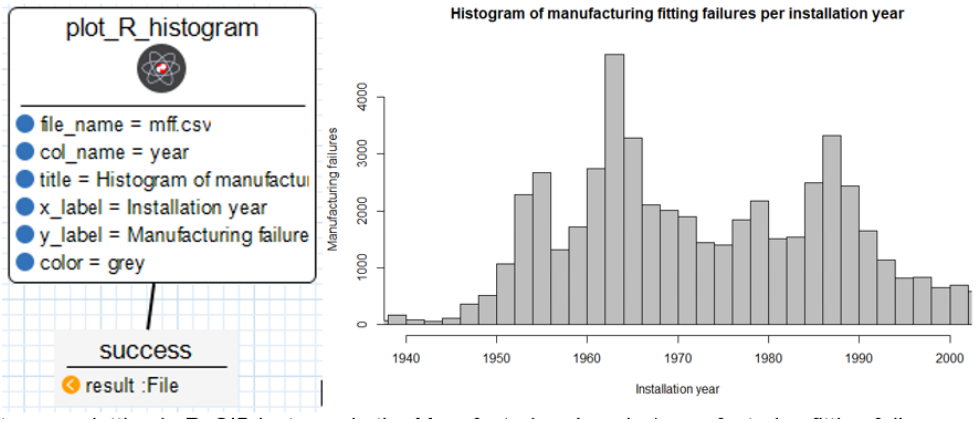


Figure 7.5: Histogram plotting in R: SIB instance in the manufacturing domain (manufacturing fitting failures per year)

general application programming and not yet in the CPS/embedded systems domain, but that is in our opinion a matter of diffusing across communities. We are surely working to reach the embedded systems, CPS and Industry 4.0 industrial adopters for our methods.

2. *What are the trends in more experimental programming language development, where the focus is on research rather than adoption? This includes topics such as e.g. program verification, meta-programming and program synthesis.*

   In this context, we see the evolution of meta-programming from the classic and traditional UML-driven community and mentality, that we see still prevail in recent surveys [24], towards the more radical approach promoted by Steffen et al. via Language Driven Engineering [18] and purpose driven collaboration using purpose specific languages (PSLs) [145]. This is a powerful, yet still niche, area of research and adoption. In this line of thought, also [146] advocates intent-based approaches and platforms as a way of channelling complexity by focusing on what matters. As adopters of the LDE and DSLs paradigms through the use of the Cinco-products DIME and Pyro/Pyrus [3, 26], we see the advantages and the power of these new paradigms and tools. The need to understand the platforms, the vari-

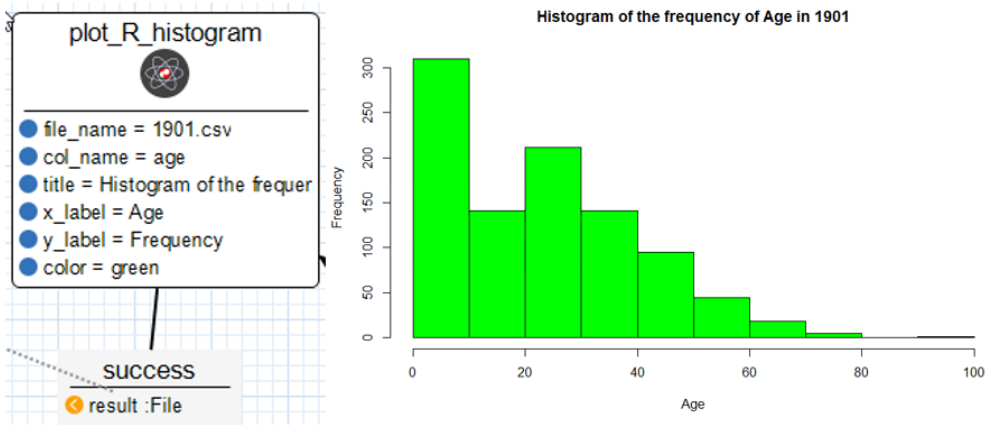


Figure 7.6: Histogram plotting in R: SIB instance in the humanities domain (1901 census population breakdown by age)

ous levels of "meta" and their interplay, which needs to be respected and embraced, requires more understanding of the interna of these paradigms, their implementations, and also the limitations imposed by the languages and platforms they at the end based upon (like Eclipse, E-core, and more). This is also underlined by Lethbridge [147], who provides also recommendations for the next generation of Low-code platforms. Core advantages of model-driven and low-code taken together are in the rapidity of evolution, and the precision of the generated artefacts. Taking out the human factor from a number of steps in the software implementation process may eliminate some genial solutions, but it also eliminates a wealth of errors, misunderstandings, and subjective local decisions that may be incoherent with other local decisions elsewhere. This enforced "uniformity by generation" has the advantage of enforcing a standard across the generated code base, and a generation standard is less unpredictable and easier to maintain and evolve.

In terms of program synthesis, we have a long experience in synthesis of workflows [148], of mashups and web services [80, 149], of applications in robotics and bioinformatics [75, 128] and of benchmark programs with well defined semantic profiles [150]. The potential for application to low-code and in particular no-code development environments that support a formal

methods-underpinned semantics is certainly enticing. The fact is, so far the popular platforms of that kind do not have formal semantics, and in this sense, the Cinco-DIME-Pyro family of platforms is indeed quite unique.

3. *What role will domain-specific languages play and what is the right balance between textual and graphical languages?*
   Concerning DSLs, we are keen adopters of them both at the language design level (as in DIME) and at the application domain level, with the External native DSLs. In our experience, they are useful to address the knowledge, the terminology and the concerns of both programmers and non-programmer stakeholders in a collaborative application development team. They are a key element of the bridge building [151] so necessary to get the right things right.
   Currently, most domain specific languages are at the coding level and do not leverage a model driven approach at the platform level. On the DSL side, the internal DSLs built in Scala of [152] address specific aspects in the design of embedded systems. They are an attractive step towards the preparation of abstractions that can connect well with the modelling level. The construction of meta-models behind these DSLs are challenging since they must capture all the domain knowledge, i.e. provide both semantic and syntactic rules. For example, Ktrain [68] is a popular coding level DSL: a Python wrapper that encapsulates Tensor Flow functionalities and facilitates developers to augment machine learning tasks with fewer lines of Python code.
   We see the graphical presentation of, specifically, coordination languages as an advantage for those tasks that privilege evidence and intuition. In this sense, "seeing" a workflow and a dataflow in a native representation as in DIME and Pyrus exposes some errors in a more self-evident way than if this representation had to be first derived from the linear syntax of customary code. Extracting again the Control Flow Graph and the Dataflow Graph, e.g., is common practice to then analyze dependencies or do the meanwhile well established program analysis and verification. We see an advantage to

use them as the explicit, mathematically correct, representation facing the designers rather than to extract them from the traditional program code where they are only implicitly present.

4. *What is the connection between modeling and programming?*
   In light of the above, the connection is tight between, e.g., the program models used in DIME and the code they represent. We are here concentrating on the software that enables the operation, in particular the interoperation and control, of applications and systems, and therefore we do not delve into the kind of cyberphysical systems modelling that concerns the physics, mechanics, and general simulation models.
   In terms of our own experience, being able to cover a variety of models in a single IME is a great advantage. The METAFrame [153] and jABC platforms [21] supported only process models, and even in DyWA [101] the integration between the data model and process models happened through import/export across two tools. In comparison, the current integration of language DSLs in DIME provides a level of comfort, ease of development and built-in checks that make DIME a success in our teaching of agile development to undergraduates and postgraduates.

5. *Will system development continue to be dominated by programming or will there be radical changes in certain application areas/generally? E.g. driven by breakthroughs in AI and machine learning techniques and applications.*
   Next to the traditional hand-coded programming and the full AI/ML based approach, we see a significant and growing role for the XMDD style of modelling [99, 138], that we see as an intermediate paradigm, more controllable, analyzable and explainable than those based on AI/ML. In our opinion, it covers the sweet spot between these two schools of thought and practice.
   Several other approaches seem to inhabit this middle too: CaaSSET [124] is a Context-as-a-Service based framework to ease the development of context services. The transformation into executable services is semi-automatic.

Agent-based modelling paradigm [123] is another popular approach to increase the development productivity in simulation environments.
In terms of AI support, for example, Xatkit [121], still in the early stages of development, increases the reusability of chatbots by evolving NLP/NLU engine for text analytics. At the language level they support several versions of bots, but the generation of chatbots from existing data sources at the framework level is in future plans.
In terms of trends that have an influence on the programming and modelling philosophy, service orientation and more recently microservices play a significant role. This architectural style that tries to focus on building single-function modules with well-defined interfaces and operations can be seen in part as an evolution of web services [126], in a trend towards the production of fine-grained systems [119] that seems to conceptually align with the growing attention to limiting scope in order to tame complexity. There are graphical approaches [127], but mostly they use standard programming languages. Dedicated programming languages like Jolie [154] offer native abstractions for the creation and composition of services, but add to the layers of infrastructure needed to develop and then execute microservices. Here, we see our abstraction as one level higher, so that we integrate microservices as simply just one additional flavour of decentralized execution [28], building on previous experience with Webservices and WSDL.

6. *Is teaching classical programming as a third discipline sensible/required?*
We would advocate that an XMDD approach based on DSLs as we have presented is easier to understand, largely (programming) language and application domain independent. In our approach, the largest part of these technical, infrastructural and knowledge layers are dealt with by IT and programming professionals who integrate the domains and this way encapsulate them. What users do see, in terms of Native DSLs and the coordination layer, has a domain specific meaning but a language and domain independent general syntax and semantics. Accordingly, we would consider

it a better choice of abstraction level to bring to the masses of professionals as a third discipline than the traditional programming in one paradigm/language, which is necessarily a very specialized choice.
There are also other frameworks in the making: for example, Aurera [117] is a low-code platform for automating business processes in manufacturing. It is a standalone desktop system that addresses the challenges of frequent changes to IT solutions. It is however still in the early stages of development and does not support communication with external systems.

7. *Can we imagine something like programming for everybody?*
Yes, we can! And the XMDD paradigm for Low-code and no-code application development is in our experience a strong candidate toward that aim.

7.6 Conclusion and Outlook

We addressed the principles, the architecture and the individual aspects of the growing Digital Thread platform we are building, which conforms to the best practices of coordination languages. Through the adoption of the Low-Code Development Environment DIME it supports a level of reuse, refactoring and analysis at the coordination layer that goes beyond what is achieved today with the current practice of glue code. We illustrated the current status, and described various extensions through generic REST services, to robotics through the UR family of robots, to the integration of various external databases (for data integration) and to the provision of data analytics capabilities in R.

We are currently working in various collaborative contexts to enrich the set of supported DSLs, as shown in Fig. 7.2. The choice of what to address next depends on the needs arising in various contexts, and it is limited by the time and staff available. The snowball effect of the impact has however already kicked in: in more than one case, a new application, sometimes in a completely different domain and collaboration, has already been able to avail of existing native DSLs, or even processes, developed in a totally different context.

Over time, we expect reuse to be increasingly the case, reducing the new integration and new development effort to a progressively smaller portion of the models and code needed for at least the most standard applications. We also expect this kind of paradigm to attract the attention of those sectors and industries that require a tighter cooperation between stakeholders with different expertise and knowledge, where there is a lack of skilled developers, and where the need for a faster turn around time can make code generation attractive as a form of automation.

Acknowledgement

This work was supported by the Science Foundation Ireland grants 16/RC/3918 (Confirm, the Smart Manufacturing Research Centre) and 13/RC/2094_2 (Lero, the Science Foundation Ireland Research Centre for Software).

Table 7.2: SIBs information for robotic arm integration.

SIB Name	Input(s)	Output(s)	Explanation
InitiateRobot	No input	result (boolean)	The SIB does not need any input. It reads the robotic configurations from a predefined location, initiates a connection with the robotic arm and returns a boolean status code according to the execution outcome.
MoveRobotToTestPosition	No input	result (boolean)	The SIB reads predefined coordinates from the application code and sends them to the robotic arm over a network connection. After the mechanical movement, it returns the according boolean status code.
MoveCoordinatesRobot	x y z ax ay az	result (boolean)	The SIB accepts a set of coordinates (3D position, 3 angles) as input, sends them to the robotic arm requesting it to move to this position, and then returns a boolean status code.
StopRobot	No input	result (boolean)	The SIB sends to the robotic arm the stop command, with instructions to shut down and stop the communication, and returns a boolean status code.

Table 7.3: SIBs information for external databases (MongoDB) integration

SIB Name	Input(s)	Output(s)	Explanation
ConnectMongoDB	ConnectionString	result (boolean)	The SIB accepts as input a ConnectionString containing the address and credentials of the server. The boolean result tells whether or not the connection is successful.
ReadfromDB	projectName databaseName attributeID	result	The SIB accepts the relevant project, database and attribute names and returns the result of the query as a single or multiple tuples.

Table 7.4: SIBs information for the R – Environment integration.

SIB Name	Input(s)	Output(s)	Explanation
execute_R_instructios	instruction	result	The SIB accepts a single R – language instruction as input and returns the execution result (if there is any) from the R – environment.
execute_R_file	file_name	result	The SIB accepts a filename as input, executes the batch of R instructions contained in that file in the R environment and returns the execution result.
plot_R_histogram	file_name col_name title x_label y_label color	image	The SIB accepts the dataset, parameters and formatting details as input, generates a histogram in the R environment and returns the resulting image.
plot_R_wordcloud	file_name col_nam min_frequency max_words	image	The SIB accepts the dataset, parameters and specification details as input, generates a word cloud in the R environment and returns the resulting image.

8

Paper III (AP3)

DSL-based Interoperability and Integration in the Smart Manufacturing Digital Thread

Hafiz Ahmad Awais Chaudhary and Tiziana Margaria

Publication Report

- **Publication Date:** 11/2022
- **Journal:** Electronic Communications of the EASST
- **DOI:** https://doi.org/10.14279/tuj.eceasst.81.1198

Contributions Summary: *Conceptualization, all authors; underlying platform selection, all authors; literature review, H.A.A.Chaudhary; architecture design, H.A.A.Chaudhary; implementation, H.A.A.Chaudhary; testing, H.A.A.Chaudhary; writing—original draft preparation, H.A.A.Chaudhary; writing—review and editing, T.Margaria, H.A.A.Chaudhary; supervision, T.Margaria; project administration, H.A.A. Chaudhary and T.Margaria; funding acquisition, T.Margaria*

Abstract

In the Industry 4.0 ecosystem, a Digital Thread connects the data and processes for smarter manufacturing. It provides an end to end integration of the various digital entities thus fostering interoperability, with the aim to design and deliver complex and heterogeneous interconnected systems. We develop a service oriented domain specific Digital Thread platform in a Smart Manufacturing research and prototyping context. We address the principles, architecture and individual aspects of a growing Digital Thread platform. It conforms to the best practices of coordination languages, integration and interoperability of external services from various platforms, and provides orchestration in a formal methods based, low-code and graphical model driven fashion. We chose the Cinco products DIME and Pyrus as the underlying IT platforms for our Digital Thread solution to serve the needs of the applications addressed: manufacturing analytics and predictive maintenance are in fact core capabilities for the success of smart manufacturing operations. In this regard, we extend the capabilities of these two platforms in the vertical domains of data persistence, IoT connectivity and analytics, to support the basic operations of smart manufacturing. External native DSLs provide the data and capability integrations through families of SIBs. The small examples constitute blueprints for the methodology, addressing the knowledge, terminology and concerns of domain stakeholders. Over time, we expect reuse to increase, reducing the new integration and development effort to a progressively smaller portion of the models and code needed for at least the most standard applications.

8.1 Introduction

The digital transformation in Industry 4.0 integrates manufacturing processes with computing devices, platforms, control systems, communication protocols, and data stores [155]. The progressive vertical and horizontal integration and connectivity in manufacturing ecosystem contribute towards improved sustainability and better resource utilization, so that companies can achieve higher industrial performance, shorter production cycles and customization on demand, ideally reaching a batch size of one [59]. Monitored and optimized manufacturing processes in smart factories demand an increased interoperability between IoT devices and systems with advanced capabilities, like autonomous and smart configurations based on historical settings and architectures. These needs are reshaping the way industries traditionally operate and communicate. In addition, the software systems, particularly in internet-centered ecosystems, demand a high level of interaction with various external services, networks, technologies and platforms.

Digital Thread [62, 63] is a data-driven architecture analogy for an integration and interoperability layer with the ability to communicate and manage interoperation between business, application and physical layer. It provides a robust reference architecture to drive innovation, efficiency and traceability of any data, process and communication along the entire system as well as the Digital Twins and the devices, machines, sensors and dashboards, which may also include analytics, AI and ML, and more. In industry 4.0 based ecosystems, a *Digital Thread* [13] connects the processes and data in order to enable smarter manufacturing and smarter factories through increased data exchange and process integration. Hence, the Digital Thread requires an end to end integration of the data sources, processes and dashboards that cooperate with each other to deliver such complex and heterogeneous interconnected systems. The costs and timeline for a Digital Thread delivery through traditional software development would be prohibitive because of the associated challenges of maintenance over time, quality of documentation, lack of modularity and reusability, and the complexity in development for the adaptation to any new dynamic requirements and upgrades [117].

Low-code development environments (LCDEs), on the contrary, promise to fulfil the robust enterprise requirements, largely automate the software development process, and address the core challenges associated with conventional software development [23]. LCDEs enable domain experts to take advantage of graphical abstractions and automatic code generation, and develop production-ready applications through model driven engineering principles [156], ideally requiring minimal or no coding experience [117, 157]. The sweet spot seems here to be the abstraction from unfamiliar programming syntax to models, combined with an abstraction from coding to the domain specific knowledge. However, there are many flavours of LCDE and many flavours of MDD. Instead of computer programs, the primary focus in the MDD paradigm is to use models that are (programming and execution) platform independent (PIM), so that the design language is less bound to the chosen underlying technology. If the chosen models have an adequate syntax and semantics, it is possible to provide automatic model-to-code generation, and also a more or less elaborate formal verification of the models. These capabilities can make it easier to model a system that is more widely understandable, easily maintainable, and possibly also scalable and flexible.

If the modeling language is much closer to the application domain, it may even empower domain experts to participate in the development process. This direct co-design approach is by far the most effective method for boosting productivity and reliability [27]. *Domain specific languages (DSLs)* are tailored for a particular domain, that encapsulates or resort domain specific constructs familiar to the domain experts, and accordingly provide abstractions where the domain experts find themselves at ease. DSLs, both textual and graphical pave the way for new possibilities for reusability, optimization and transformation, and even formal verification that would be much harder to achieve otherwise [17]. To a certain extent, the combination of LCDEs, MDD and DSLs seems therefore a winning strategy, but there is not yet a solution that supports the heterogeneity needed for the Digital Thread, the formality needed for the MDD-based rigour for high assurance software, and the specificity embodied by DSLs.

We address here the task of supporting all three aspects for the Digital Thread by means of a platform concept. Specifically, we support the integration of exter-

nal services from various (domain specific) platforms and various programming languages in order to deliver their much simpler and better controlled interoperability. Our goal is to develop a service oriented domain specific Digital Thread Platform in the context of Smart Manufacturing, for which we provide these integrations and orchestrations in a formal methods based, low-code and graphical, model driven fashion. We chose the Cinco products DIME [2] and Pyrus [3] as the underlying IT platforms for our Digital Thread solution to serve the needs of the applications addressed: manufacturing analytics and predictive maintenance are in fact core capabilities for the success of smart manufacturing operations. They use a collection of tools and techniques to detect anomalies and potential defects in the processes and the equipment before these reach a point of failure. The use of data-driven proactive maintenance methods also helps operators to plan the maintenance schedules for the equipment.

In this regard, the contribution of the paper is that we extend the capabilities of these two platforms with new external integrations in the domain of data persistence, IoT connectivity and analytics, and provide an overview of services linking together in the platforms to support the basic operations of smart manufacturing. DSLs with a family of SIBs [21] are added with small examples that are sufficient to constitute blueprints, i.e. reference examples that the adopters can use as a starting point, and adapt and enrich as needed. The SIBs (Service Independent Building blocks) are executable modeling components for process models. We are also moving from individual reusable SIBs to features [158, 159] intended here as ready-made workflows, where a collection of SIBs are packaged into workflows that are ready to use as a hierarchical SIB.

The challenge to maintain consistency between continuously evolving system level requirements, component specifications, and evolving underlying implementations [160] is addressed by resorting to hierarchical structuring, the introduction of a feature level abstraction [161], and a systematic integration of externally provided services that goes beyond the manual, ad-hoc integration of the services one by one. This has been done in the past with automatic integration of externally provided WebServices through generation of the corresponding classes and wrappers [162]. Similarly, prior to DIME, to make the data type definitions in DyWA accessible to the process models, a code generator generated the domain

specific classes and the corresponding CRUD services [101]. With DIME, the accessibility of data models to process models is built-in, so there is no export step anymore. This approach of encapsulation and publishing has been successful in many respects, in particular it supports the definition of formal service and type semantics, which enables automatic process analysis and generation techniques [163], and hence also large scale automatic composition and reuse[164]. The main research question we face now concerns the feasibility of these many and diverse integrations in a more abstract, systematic, and guided way than just through the customary case-by-case and service-by-service ad hoc approach.

In this paper, Sect. 8.2 discusses the related work, Sect. 8.3 gives an overview of the principles and architecture of the Digital Thread Platform, Sect. 8.4 discusses two representative case studies, with integrations in DIME and in Pyrus, and finally in Sect. 8.5 we conclude and discuss next steps.

8.2 Related Work

Over the years, domain specific languages, both textual and graphical in a low-code and model driven paradigm, have become one of the most popular approaches for the design and development of heterogeneous systems [25, 64, 65]. According to Gartner [120], Low-Code Application Platforms (LCAP) are being increasingly adopted by industry, specifically by software-as-a-service (SaaS) vendors and are expected to grow significantly in adoption and economic impact over the years. These platforms address the challenges associated with conventional development paradigms, and democratize the process of software development by facilitating the participation of domain experts in the development cycle who have little or no coding knowledge. This is opposed to traditional hard-coded programming techniques [117], that are beyond the competence of most domain experts. The first meta-level framework of this category was proposed in 1995 [100] with immediate industrial applications in the INXpress product by Siemens-Nixdorf [165]. Framework thinking combined with a low-code approach offers support for the systematic and rapid generation of application specific complex objects from collections of reusable components. Considering the costs associated with skilled human resources and the high maintenance costs of software

and IT systems, automation is by far the most effective way to stay competitive yet deliver high quality solutions.

Model driven development (MDD) with adequate models is an automated approach to the rapid design of flexible and cost effective applications by means of drag & drop visual interfaces. Holistic MDD covers from the conceptual modeling phase to the model-to-code transformation phase [22]. In this line of thought, the jABC platform [21] based the design and development of applications on formal models and its Lightweight Process Coordination. It accelerated the development process of applications through the concept of reusable building blocks, orchestrated into analyzable control structures called Service Logic Graphs. Following similar practices, several model driven platforms were proposed for model checking [74], early bioinformatics applications [75, 166], fostering collaboration through tool interoperability within the FMICS Working Group on Formal Methods for Industrial Critical Systems [77, 78], to compose and combine heterogeneous planning algorithms in Plan-jETI [79, 80] and also mobile apps [81] and cross platform [82] applications. Enhancing the model driven paradigm with domain specificity tailors the modeling environment towards a specific application domain, with the purpose of enhanced productivity, reduced complexity and increased domain expert participation [25]. In general, the development of DSLs may follow any of the available implementation approaches, and may require a collection of guidelines, design patterns, model checking and common language design and implementation challenges for reusability [72]. A comparison of different implementation approaches of 40 DSLs from very different domains [73] concluded that the compiler based approach (42.1%) is the most popular. One of the case study shows a model driven DSL for the personalisation of travellers' information that designs information systems by assembly of pre-existing models using the PERCOMOM method [167]. Similarly, model driven modeling has been applied to the development of distributed control systems for the automatic transformation of block-based design models to a component-model implementation [85].

The Industry 4.0 revolution [34] covers a broad range of key enabling technologies like cyber physical systems, digital twins, IoT, AI, AR/VR and big data analytics. The adoption of these Industry 4.0 technologies pushed manufacturing

companies and their suppliers to adopt and adapt data-driven strategies, with the focus to meet the dynamic needs of "smart" factories and contributing towards sustainable manufacturing environments. Meanwhile, [168] and [23] discussed the emerging research and practices on sustainable smart manufacturing and Industry 4.0 with focus on existing low-code development platforms. The paradigm shift in manufacturing from traditional approaches to smart manufacturing requires in fact an end to end integration and interoperability at different stages of different business entities, processes and systems [58]. In this context, [13] discussed the key role of the Digital Thread in Industry 4.0 to couple the data and processes and provide traceability in the entire lifecycle, to deliver an integrated smarter ecosystems.

The Language-Driven Engineering approach of [18] addresses the issue of adapting the needs of domain specific languages with the need to contain the cost, expertise and complexities in the development of corresponding baseline tools from scratch. The Cinco [25] meta-level platform by the same authors supports a meta-model driven development of graphical domain specific modeling tools, and it also provides automatic code generation capabilities from high level (meta-model) specifications. In the LDE, two high level meta-modeling languages describe the language elements, syntax and semantics (through the Meta Graph Language, MGL) and the rendering style in the graphical editor (through the Meta Style Language, MSL). Several modeling tools and frameworks developed using Cinco have so far addressed multiple industrial and academic settings [2, 26, 169, 170]. In our Digital Thread platform we chose to work with two specific products generated from Cinco meta-models: DIME [2] and Pyrus [3].

DIME is an Eclipse based, low-code MDD environment for modeling and deploying complete web applications. Its one-click generation and deployment in a service oriented manner [171] is an example of the One Thing Approach (OTA) defined in [98]. Being an Integrated Modeling Environment, DIME supports hierarchical modeling, useful for scalability and real life system design. Its several model types cover the different aspects of an application: the data model for the data type definitions and persistency, the process models for the business logic (control flow and data flow), and the GUI models for the Web application's user interface. The models pertaining to the application under development are

validated dynamically wrt. syntactic and static semantics both at the model and the project level. This built-in support facilitates the domain experts in debugging their designs. Warnings and error messages help localize the issues at design time, even before implementation and deployment, therefore cutting down significantly the testing effort and costs.

Pyrus is a web-based collaborative graphical modeling environment for no-code data analytics. Pyrus enables users to comfortably orchestrate and reuse (analytics) functionalities implemented in Python and available in Jupyter notebooks. Users access Pyrus as an online graphical IDE, discover pre-implemented functionalities from an established online IDEs as DSLs, and orchestrate them graphically to produce collaboratively data analytics pipelines. Like DIME, Pyrus follows the core principles of LDE [18], OTA [98] and eXtreme Model Driven Development (XMDD) [99]. Hence its users should be able to take design decisions for their pipelines, i.e. decide "what" the pipeline does, without needing to know how to implement the individual functionalities (i.e. the "how" to do it at the programming level). In this sense, Pyrus promotes virtualization, encapsulation and reusability in a low-code/no-code fashion.

8.3 Extending the Digital Thread Platform

We work towards the creation of a smart manufacturing Digital Thread ecosystem by extending the domain specific functional capabilities offered by the base platforms DIME and Pyrus through the systematic integration of various externally provided capabilities. They are integrated in a service-based fashion, as *external native DSLs*, in order to provide to the Digital Thread platform a collection of ready to use application domain specific functionalities that may run on a technologically heterogeneous landscape.

As we are part of the Confirm Smart Manufacturing research centre[1], we target here specifically the application domain of advanced manufacturing, in order to offer a comfortable low-code/no-code web application development in

[1]Confirm (www.confirm.ie) is the SFI national research centre in Smart Manufacturing headquartered at the University of Limerick, with a mission to transform industry to become leaders in Smart Manufacturing

DIME and a dedicated manufacturing analytics development in Pyrus. In this context, projects routinely face data, processing, workflows and communications from and to different sources, like tools, machines, data and knowledge bases, with various service providers, and interfacing with a number of different vendor specific and (mostly data) abstraction platforms. The case studies are expected to concern a large variety of devices, data sources, data storage technologies, communication protocols, technologies and tools for analytics or AI, visualization tools, and more. This level of diversity and heterogeneity is where the integration of external native DSLs plays a key role.

We are incrementally building a significant ecosystem [11] of collaborations spanning various application domains to use all those services in the application design, in a possibly seamless way. In our first proof of concept [28], we implemented a generic extension mechanism, in analogy to micro-service architectures [119], to the two low-code development environments DIME and Pyrus. The PoCs integrated a generic RESTful service as a DIME Service Independent Blocks (SIBs) and a cloud-based AWS enterprise service as a Pyrus SIB, respectively. This was useful to demonstrate the feasibility to an audience of engineers not accustomed to model driven development, service oriented computing, nor low-code/no-code. The quality and completeness of the research outcomes are validated via empirical research, both within the open source community and with the adopters in academia and industry. The ability to carry out *agile modifications* on a use case basis is due to the iterative and prototype driven approach, where at each cycle new improved releases of the previous content are planned and evaluated, and the content grows from one release to the next. Within the research group, we follow the three cycle view of design science research [113] methodology, hence reconnecting to both the design and the relevance cycle.

In general, we see here that the Digital Thread platform successfully connects various elements of the advanced manufacturing landscape as they are found in large *Manufacturing Testbeds*, like those in the Confirm research centre headquarters in Limerick and others at specific industry partner sites. Such facilities include on purpose diverse and redundant categories of equipment like robots, working tools and machines, building and factory floor infrastructure, as well

as a selection of communication networks. Their aim is in fact to facilitate experimentation, either between researchers and industrial project partners (as in Confirm) or as a solution co-design infrastructure between commercial providers of specific equipment and their customers, as in ADI's Catalyst collaboration hub[1]. In such equipped spaces, the goal is to try out established and new technologies and their combination, experiment with feasibility and fact-finding, with relatively short turn around cycles of individual experiments and testbeds (a few weeks to a few months), in order to distil use cases, configurations and semi-ready reference applications that can be easily turned to projects, products and new state of the art. The abstraction from the "real" cyber physical systems in the Manufacturing Testbeds to a set of services provided by **External Service providers** as **External Native DSLs** is the key to enabling the kind of modeling and implementation we envisage.

8.3.1 The DIME Platform Architecture

The current architecture of the Digital Thread platform is based on DIME as its underlying Integrated Modeling Environment and is shown in Fig. 8.1. The DIME **Modeling Layer** provides the essential modeling capabilities, like the specific modelling languages for the user interfaces, the data model and the process models. This is the standard DIME distribution that is accessible online at *https://scce.gitlab.io/dime/content/introduction*. We adopt the DIME modeling languages as they are, with no changes nor extensions. Our goal here is to keep the modeling language as stable as possible, not creating any branch that may give rise to incompatibilities across syntactic and semantic expressive power and capabilities. This modeling language layer constitutes the *modelling grammar* of the Digital Thread Platform.

Seen from our users' perspective, the core of the Digital Thread platform concerns the **Process Layer** and it is going to be used as a no-code platform

[1]ADI's Catalyst is an R&D collaboration hub for customers looking to get to market faster, generate revenue more efficiently, and strengthen and evolve their ecosystem. https://www.analog.com/en/about-adi/incubators/catalyst.html

for application development. Given an application under development, its process model consists of the set of possibly hierarchical DIME process models that collectively describe the control flow and data flow of the business logic for this application. The main process is usually an orchestration of a number of subprocesses, plus some individual atomic functionalities that are provided as SIBs. The overwhelming majority of these processes and SIBs reside in the Process Layer, where we distinguish Common, External Native and Process DSLs. **Common DSLs** come with the default DIME distribution. These built-in DSLs are application independent and cover the basic helper operations in building business logic, like e.g. string and arithmetic operations. **External native DSLs**

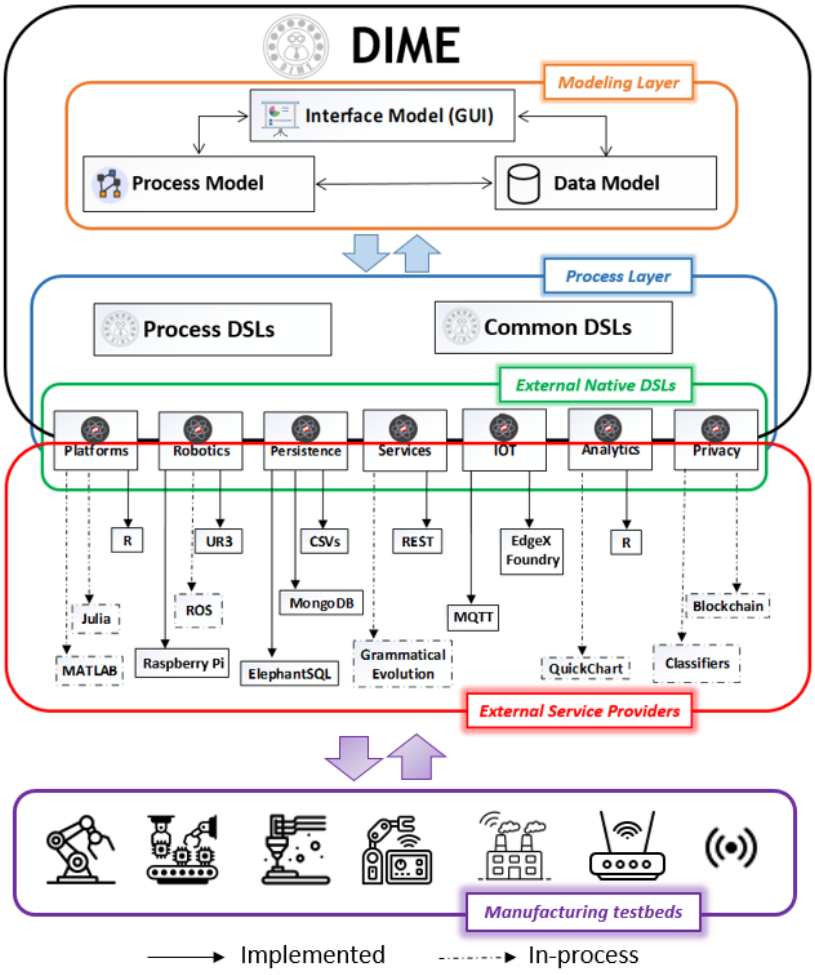


Figure 8.1: Architecture overview of the Digital Thread Platform - DIME

are the set of services, technologies and platforms that reside outside the standard DIME distribution. Their capabilities are accessed at runtime via network protocols in the context of the Digital Thread platform. **Process DSLs** implement the domain specific generic and hierarchical workflows as orchestrations of common and external native DSLs. Our extension of DIME for the Digital Thread platform concerns a) the External Native DSLs: this is where the domain specific *vocabulary* extension takes place by adding devices, external programs, and AI/reasoning, and b) the *Process Layer*, which collects over time a number of domain specific Process DSLs, including the (few) Common ones provided by DIME itself.

Accordingly, each domain specific DSL consists of

- a number of atomic SIBs that provide a set of individual capabilities as the smallest units usable in the models, and
- a number of predefined processes that offer coarser grained capabilities, that are used in the hierarchical models.

The **External Service Providers** layer shows a list of technologies and frameworks that are either already integrated (solid arrows) with the platform or work in progress (dotted arrows). External Native DSLs organize the integration and publishing of these external technologies as individual graphical DSLs. A possible organization principle is according to the different verticals, e.g. IoT, persistence, analytics etc. REST services [28], Data analytics with R libraries [11] and the MongoDB no-SQL database were integrated as services, while R, seen as a programming language, and the EdgeX Foundry middleware integration and interoperability platform had to be integrated as platforms [29], which is the more general but also more complex integration form. EdgeX Foundry [4] serves as an edge middleware for the IoT - abstracting and mediating between physical sensing and actuating "things" and our information technology (IT) systems. It is therefore particularly interesting for our platform, as it eliminates the cumbersome work of integrating each device individually into the Digital Thread Platform. MongoDB is also very useful as it handles the flexible and semi-structured nature of data schemas.

8.3.2 The Pyrus Platform Architecture

The current architecture of the Analytics platform is based on Pyrus as the underlying web-based collaborative graphical modeling environment and is shown in Fig. 8.2. In it, the **Ecore** component displays to the users a collection of default DSLs as well as external DSLs. This way, the users can access the SIBs of those DSLs and orchestrate them graphically in a data flow fashion. The default SIBs come with the standard distribution of Pyrus and cover the basic data analytics needs. The other DSLs are added as a result of functions discovery from the Jupyter Hub instance that serves as the execution server. The Pyrus SIBs are usually in Python. However, the Python functionality of advances and domain specific DSLs, like the Smart manufacturing analytics in our case, are implemented in the JupyterHub space as normal Python code. From the perspective of developing a DSLs, annotations are added at the top of the Python functions in a simple SIB declaration language. Pyrus reads them and with that information it presents them to the users as a SIBs in the corresponding DSL. From an integration point of view, the **External Service Providers** layer shows the external technological stack under integration in this fashion. The many popular libraries and frameworks that are currently supported in Pyrus range from AI/ML libraries, cloud services [28] and cloud databases like many AWS services, MongoDB etc.

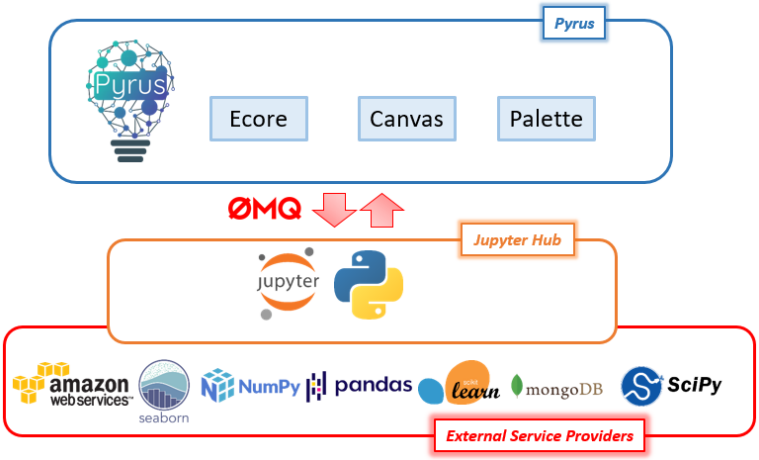


Figure 8.2: Architecture overview of the Analytics Platform - Pyrus

For functions discovery and pipeline executions, Pyrus communicates with Jupyter over the ZeroMQ protocol. The **Palette** component of Pyrus provides a list of standard input/output holders to attach variable values at runtime. The **Canvas** is the drawing board, shared by one or more users, used to develop pipelines with the help of SIBs and I/Os from the Ecore and Palette components. Indeed Pyrus supports a collaborative, synchronous co-design online, in the browser.

8.3.3 Overcoming Heterogeneity

The **External Native DSLs Layer** handles the integration and interoperability with the outer world. It abstracts its inherent heterogeneity from the users by providing them with a nice, easy and uniform presentation in terms of individual SIBs, palettes with DSLs, and collections of processes. The design and implementation of the External Native DSLs is a difficult task that we as experts have to accomplish, in order to provide uniformity and simplicity to users who work within the process and modeling layers.

The External Native DSLs in fact provide a series of different technology verticals ranging from external IT platforms, domains like robotics, vendor specific technologies, execution environments, and overarching services like privacy, authentication, and more. The central property of simplicity here hinges on autonomy and reusability: once integrated, all these disparate functionalities will be presented as a collection of SIBs. These SIBs are families of built-in modeling classes that reference executable implementations running in the appropriate platforms/containers. Based on prior integration experience and the appropriate extension mechanisms, we have discussed in [29] two distinct integration alternatives: services and platforms.

The expectation with service integrations is that the services are already deployed on servers, thus accessible and ready to use over a public/private network. The integration mechanism then follows the principles of native library services interfaces: the required functionality is embedded as a family of SIBs that are ready to be used as a drag and drop modeling components. In DIME, the signature of each new functionality with a correct syntax is added to a file with

the extension .sib and the SIB definition consists of a SIB name, the list of its inputs and outputs and the fully qualified path of the Java implementation to be invoked when used in a model. In Pyrus, instead, the annotations for SIBs are added on top of each of the Python functions in a file with .py extension.

The additional challenge associated with platform integration is the need to prepare the external execution platform in such a way that it serves the needs of the underlying application. Such platforms are in different runtime environments, with their own technological infrastructure that may vary across the technologies and the deployments. Mostly the infrastructure is prepared and deployed in separated docker containers, so that the different environments stay isolated, are flexible, scalable and accessible over standard protocols, and deliver the required functionalities. The case studies presented next in Sect. 8.4 discuss small workflows that provide examples of end to end implementation of various stages of manufacturing analytics and predictive maintenance.

8.4 Smart Manufacturing Case Studies

Manufacturing analytics and predictive maintenance are two important capabilities for the success of smart manufacturing operations. They use a collection of tools and techniques to detect anomalies and potential defects in the processes and the equipment before these reach a point of failure. The use of data-driven proactive maintenance methods also helps operators to plan the maintenance schedules for the equipment. In the following, we discuss how we implemented the core of these applications as DSLs in DIME and Pyrus. While these are small examples, they are sufficient to constitute *blueprints*, i.e., reference examples that the adopters can then use as a starting point, and adapt and enrich as needed. We searched on purpose for simple examples that are complete, in order to showcase the essence of the respective applications with minimal complexity.

8.4.1 Manufacturing Analytics: a Process DSL in DIME

The DSL for manufacturing analytics in DIME aims to show Confirm users how to analyse datasets and visualize them in web applications. The dataset used in

this specific example is taken from the Kaggle repository [172] and consists of historical data from a hundred different manufacturing units. It contains a list of failures, voltage consumption, and rotation cycles across different sensors and components, together with the unit type.

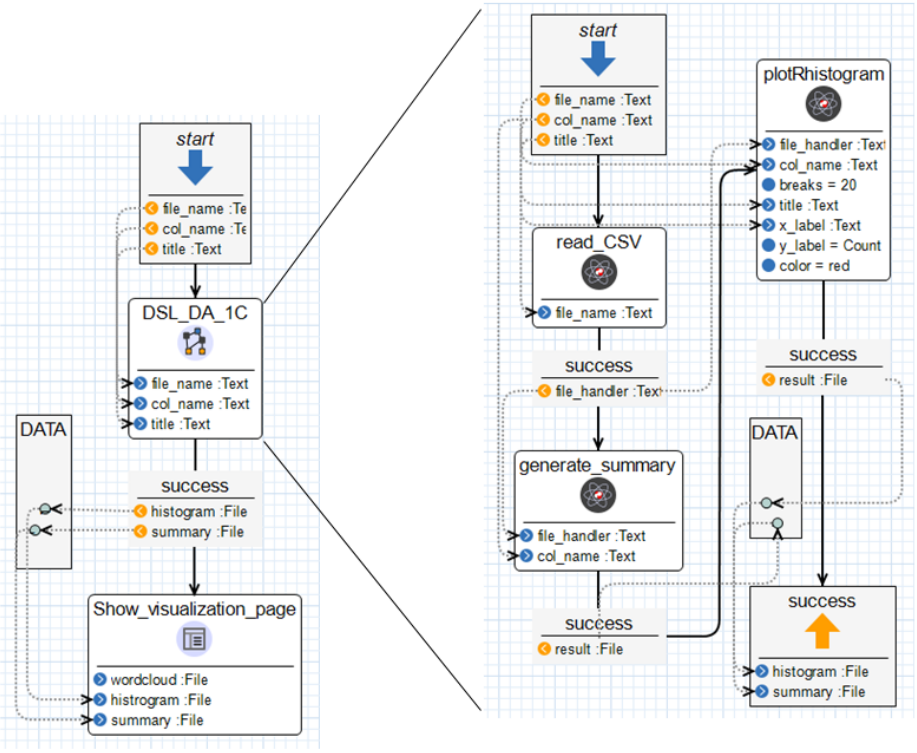


Figure 8.3: Hierarchical process model for manufacturing analytics

Fig. 8.3 (left) shows an excerpt of the Manufacturing Analytics process model, consisting of two hierarchical SIBs: the process SIB (DSL_DA_1C) and the GUI SIB (Show_visualization_page). The body of the process SIB is shown in Fig. 8.3 (right): its control flow proceeds from the Start with a list of inputs and reaches the SIB read_CSV, that reads the content of the data file and passes it to the generate_summary SIB with a parameter indicating the attribute for which a summary is desired, in this case the age column. Its execution computes various model fitting functions, providing a result file. The subsequent plotRhistogram SIB, which has a list of other required parameters for content and layout, computes the histogram of the given column. We see here the control flow displayed as solid arrows, and

the data flow displayed as dotted arrows. This explicit representation of both flows enables a simple visual validation. As the models are also formal models (specifically, Kripke Transition Systems, KTS) [173], they are also amenable to data flow analysis and formal verification, e.g. by means of model checking for temporal logics [122, 174].

The SIBs in the workflow expose the input and output ports for communication with peer SIBs and other (nested or encapsulating) processes. All the analytical computations are executed internally on an R infrastructure reached over the TCP/IP protocol. We reuse here our previous platform integration of R. After successful execution, the control is transferred back to the parent SIB and the results are passed to the application's GUI through the GUI SIB Show_visualization_page that displays them in the web application's interface model.

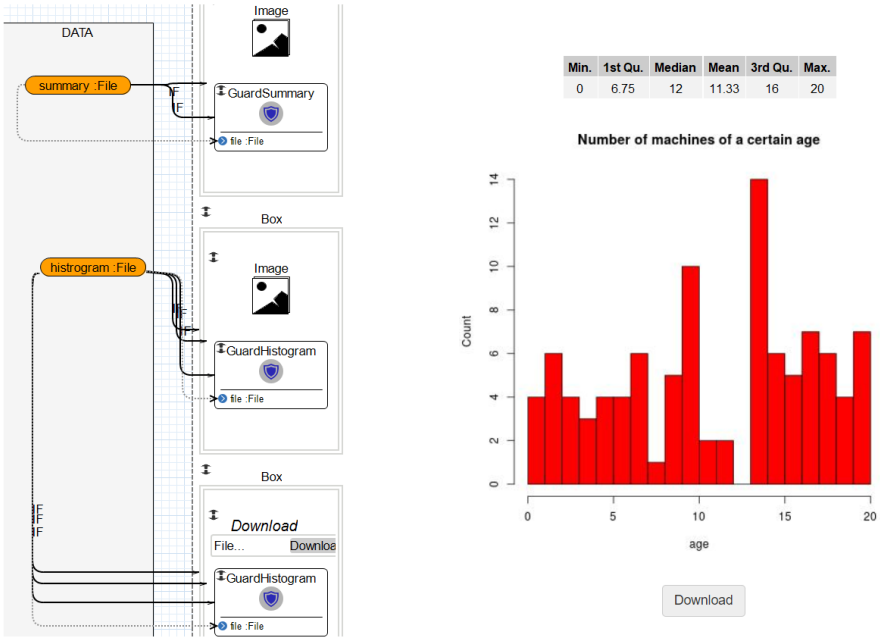


Figure 8.4: Manufacturing analytics DSL: interface model (left) and web page (right) for a manufacturing analytics page

Fig. 8.4 (left) shows a segment of the GUI model for this page and Fig. 8.4 (right) the corresponding rendered webpage. On the left, we see how the model treats the display data received from process models. There are three interface

blocks: an image display for the summary, an image display for the plot, and an action button to enable the download. While the atomic SIBs in Fig. 8.3 are our (DSL) extensions to DIME, the GUI models are implemented fully within the GUI DSL provided by the basic DIME distribution to all its users, as we explained in Sect. 8.3.1. A uniform GUI language across all the DIME applications is a great advantage for the human comprehension of the Web applications and their ease of maintenance, even across application domains. In Fig. 8.4 (left) we also see that the image placeholders are protected with Guard SIBs, to ensure only properly authorised access to the data.

Users of the Web application see only the webpage shown in Fig. 8.4 (right): this is the generated web interface, displaying a summary of different model fitting functions concerning the age of the machines in the considered machine pool (between 0 and 20 years old, with a mean age slightly above 11 years), and a histogram detailing the failure count for machines in dependence of their age. For example, in this case, older machines (>13 years) fail more often than younger ones, the worst performing machines are 10 and 13 years old, and it may be the case that the machines aged 11, 12 and 13 (if there are any), which have very low failure counts, may have undergone recent successful maintenance/ reconditioning.

The hierarchical workflow presented in Fig. 8.3 and Fig. 8.4 is easily encapsulated as a single process SIB for maintenance analytics: the Feature_Maintenance_Analytics shown in Fig. 8.5. It is a complex, self-contained building block that is reusable inside other processes that require this kind of data analysis and display. The corresponding code is generated at web application deployment time with DIME's fully automated model-to-code transformation facility.

The workflows we developed for this descriptive analysis are actually reusable across different domains: the manufacturing content is fully determined by the dataset under consideration. If we used a healthcare dataset, or an education related dataset, the same workflow would produce a dashboard for those other application domains. In this sense, we succeed in creating a library of transdisciplinary DSLs, that can be easily reused even across communities.

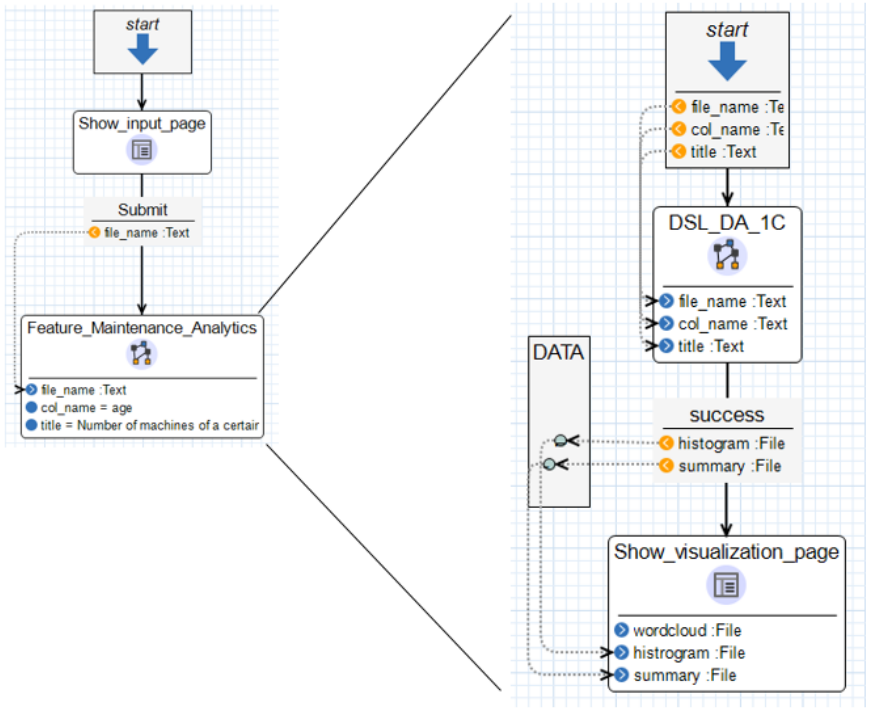


Figure 8.5: Hierarchical process SIB for maintenance analytics

```
# Method: correlation_matrix
# Inputs: table:Table
def correlation_matrix(bev):
    cor = bev.corr()
    Fig, ax = plt.subplots(figsize=(12,6))
    sns.heatmap(cor, annot = True)
```

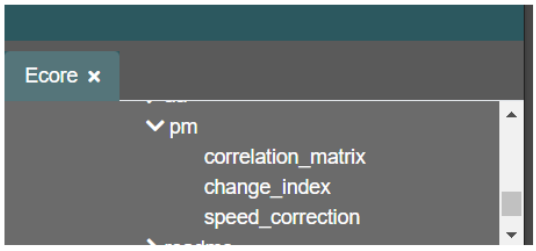


Figure 8.6: Correlation matrix: Python function annotation in Jupyter and its visualization in Pyrus

8.4.2 Proactive Maintenance Planning: a DSL in Pyrus

The second case study concerns a DSL for predictive maintenance of machines and it is implemented in Pyrus. This time we do not create the functions from scratch,

but instead encapsulate and reuse pre-existing, externally developed back-end Python scripts for the creation of graphical DSLs. The dataset used in this example stems from the beverages manufacturing industry: it contains a list of equipment and instruments used to develop certain products. The dataset and Python scripts are taken from the GitHub repository [175]. Fig. 8.6 (left) shows a snippet of the Python code for the correlation_matrix in Jupyter, the added annotations for Pyrus integration on top, and on the right how it appears listed as one of the drag and drop components once imported in Pyrus. This illustrates the case of integration of external Python libraries, as is the case for example for scikit [176] and many other popular libraries like Matplotlib for numerical plotting, Numpy, Pandas, SciPy etc.

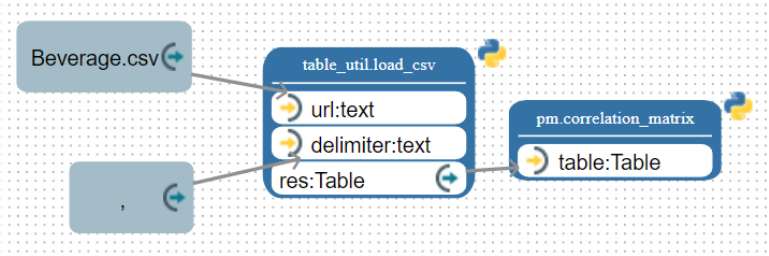


Figure 8.7: Pyrus data pipeline for correlation matrix display within a predictive maintenance DSL

The Pyrus pipeline shown in Fig. 8.7 is very basic: it calculates the correlation matrix expressing the association and dependency of various industrial components, like the Blower, the Labeller, the Palletizer, etc., on each other. In the Pyrus modeling canvas, the required constant nodes on the left concern two strings: they define the dataset path (beverage.csv) and the delimiter (a comma) in the CSV. The first connected computational SIB is table_load_csv: it reads the dataset and passes it to the pm.correlation_matrix SIB. This SIB computes the correlation matrix and displays the result as the heat map shown in Fig. 8.8.

Similarly, Fig. 8.9 shows a second data pipeline, useful to monitor and correct some of the equipment parameters (i.e., reduce the blower speed if it exceeds 22000 rps (radians per second)) and plot them before and after implementing the corrective measures. The first SIB, table_util.load_csv loads the dataset like in the previous example. The second and third SIBs, table_util.sub_table and

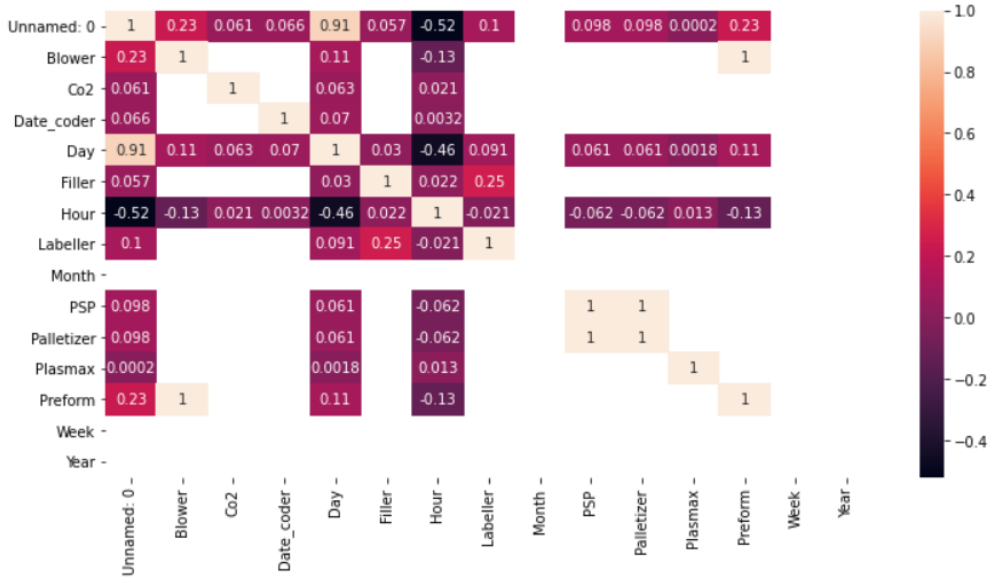


Figure 8.8: Heat map for the beverage industry

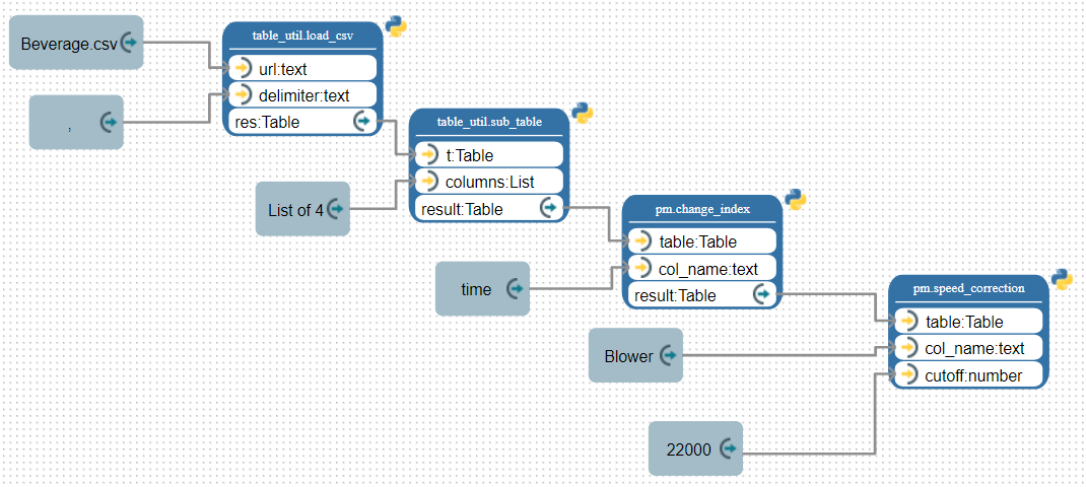


Figure 8.9: Pyrus data pipeline for predictive maintenance DSL

pm.change_index, filter the required columns, provided as inputs as a list, and apply database indexing on the selected column (here, the time column) for performance optimization in further data accesses. Finally, the pm.speed_correction SIB plots the behaviour of the Blower (or any other monitored machine, as this is a parameter) over time vs. the tuning parameter subject to the corrective measure. Here, for example, Fig. 8.10 shows the Blower speed before (blue plot

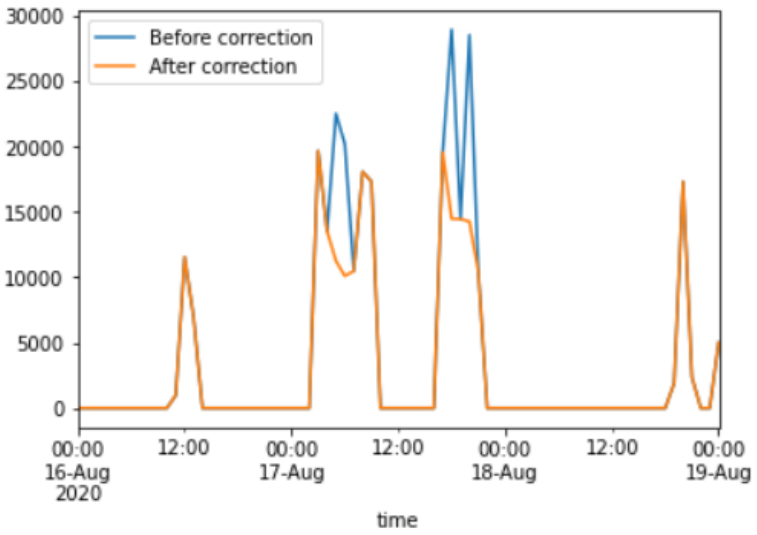


Figure 8.10: Predictive Maintenance: plot of the corrected vs. original behaviours

line) and after the correction (yellow plot line).

All the implemented functionalities can be used within Pyrus as no-code drag and drop components. After connecting the correct data flow, users can this way execute the pipelines for their specific scenario, and obtain the resulting plots like shown in Fig. 8.10 without any need of programming. This capability gives domain experts the opportunity to concentrate solely on the composition and parameter optimization of the individual functions in no-code fashion, without having any deep expertise in Python coding. Finally, the creation of a hierarchical workflow with prior connectivity of data flow is in progress.

8.5 Conclusions and Next Steps

We addressed here the principles, the architecture and the individual aspects of the growing Digital Thread platform we are incrementally building, which conforms to the best practices of coordination languages. Manufacturing analytics and predictive maintenance are both important capabilities for the success of smart manufacturing operations. In this regard, we have extended the capabilities of platforms in the vertical domains of persistence, IoT and analytics to support core operations of smart manufacturing. DSLs with a family of SIBs are

added, with small examples that address the individual knowledge, terminology and concerns of individual stakeholders or small groups of stakeholders, yet are sufficient to constitute blueprints for easy adoption and modification. Following the philosophy of thinking in units of functionality for simplicity [177] and reusability, and using hierarchy for scalability to larger and more complex systems [158, 178], we are also moving from individual reusable SIBs to *features*, intended here as ready-made workflows, where a collection of SIBs are packaged into task-specific sub-workflows that are ready to use as a hierarchical SIBs.

The quality and completeness of the research outcomes are validated via empirical research, both within the open source community and with the adopters in academia and industry. The ability to carry out agile modifications is due to the iterative and prototype driven approach, where at each cycle new improved releases of the previous content are planned and evaluated, and the content grows from one release to the next. The capability of rapid development with built-in checks makes these tools a success in our teaching of agile development to undergraduates and postgraduates. In addition, we used these topics as a teaching case study for Masters students, who further extended the collection of Pyrus capabilities and carried out a number of academic case studies.

The next steps include performance improvements of the baseline architecture, refactoring some of the implemented integrations and enriching the list of services and technological integrations. Over time, we expect reuse to be increasingly the case, reducing the new integration and new development effort to a progressively smaller portion of the models and code needed for at least the most standard applications. We also expect more categories and best practices of integration to emerge, in order to produce highly reusable External native DSLs.

Acknowledgment

This work was supported by the Science Foundation Ireland grants 16/RC/3918 (Confirm, the Smart Manufacturing Research Centre) and 13/RC/2094 and 2094-1 (Lero, the Software Research Centre).

9

Paper IV (AP4)

Efficient Model-Driven Prototyping for Edge Analytics

Hafiz Ahmad Awais Chaudhary, Ivan Guevara, Jobish John, Amandeep Singh, Alexander Schieweck, Tiziana Margaria and Dirk Pesch

Publication Report

- **Publication Date:** 09/2023
- **Journal:** Electronics
- **DOI:** https://doi.org/10.3390/electronics12183881

Contributions Summary: *Conceptualization, All authors; underlying platform selection, T.Margaria and D.Pesch; architecture design, H.A.A.Chaudhary, I.Guevara, A.Singh, A.Schieweck and J.John; implementation, H.A.A.Chaudhary, I.Guevara, A.Singh and J.John; resources, H.A.A.Chaudhary, I.Guevara, A.Singh, A.Schieweck and J.John; hardware setup, D.Pesch, J.John, H.A.A.Chaudhary, I.Guevara and A.Singh; writing—original draft preparation, I.Guevara, A.Singh, H.A.A.Chaudhary, A.Schieweck and J.John; writing—review and editing, T.Margaria, D.Pesch, H.A.A.Chaudhary, I.Guevara and A.Schieweck; supervision, T.Margaria and D.Pesch; project administration, H.A.A. Chaudhary, I.Guevara and T.Margaria; funding acquisition, T.Margaria and D.Pesch*

Abstract

Software development cycles in the context of IoT applications require the orchestration of different technological layers and involve complex technical challenges. The engineering team needs to become experts in these technologies and time delays are inherent due to the cross-integration process because they face steep learning curves in several technologies, which leads to cost issues, and often to a resulting product that is prone to bugs. We propose a more straightforward approach to the construction of high-quality IoT applications by adopting model-driven technologies (DIME and Pyrus), that may be used jointly or in isolation. The presented use case connects various technologies: the application interacts through the EdgeX middleware platform with several sensors and data analytics pipelines. This web-based control application collects, processes and displays key information about the state of the edge data capture and computing that enables quick strategic decision-making. In the presented case study of a Stable Storage Facility (SSF), we use DIME to design the application for IoT connectivity and the edge aspects, MongoDB for storage and Pyrus to implement no-code data analytics in Python. We have integrated nine independent technologies in two distinct Low-code development environments with the production of seven processes and pipelines, and the definition of 25 SIBs in nine distinct DSLs. The presented case study is benchmarked with the platform to showcase the role of code generation and the reusability of components across applications. We demonstrate that the approach embraces a high level of reusability and facilitates domain engineers to create IoT applications in a low-code fashion.

9.1 Introduction

Modern Internet of Things(IoT) architectures enable efficient communication among interconnected infrastructures of sensors, devices, and systems, with the intention of offering innovative solutions for diverse applications [179]. In recent years, there has been a significant increase in the adoption of various IoT platforms and there is a growing demand for these platforms, devices and solutions from different vendors to seamlessly work together. This need not only leads to interoperability challenges in the ecosystem, but also makes the software development process more complex. The complexity across the different technological layers that compose a system requires deep and cross-functional technical expertise and integration knowledge for each of the involved heterogeneous systems. In this context, IoT middleware platforms become an integral part of the IoT ecosystem that provides a common interface between the different sensors, computing devices, and actuators. These platforms address different kinds of requirements and often push project managers to perform an exhaustive analysis before choosing and implementing a specific type of architecture. From the point of view of application developers, the lack of compatibility between IoT platforms introduces an extra step towards tailoring their applications to support platform-specific APIs. Considering the specifications of the application under development, the analysis may concentrate on various aspects such as interoperability, scalability, security and privacy, spontaneous interaction, or an unfixed structure.

In this context, we are developing a low-code Digital Thread Platform (DTP) [11] for Cyber-Physical Systems (CPS), which is based on the principles of Model Driven Development (MDD) and supports the capabilities of automatic code generation from the formal semantics underlying the models. It also supports one-click deployment of heterogeneous applications that require the interoperability across various systems, technologies and programming paradigms. The platform also enables the development of personalised solutions by seamlessly integrating applications and solutions into the wider landscape of enterprise applications, harnessing the potential of digital threads and digital twins [180, 181]. We opted

for model types that have well-defined semantics for formal analysis and verification. This decision is based on our belief in the advantages of early validation and verification at the model level, as well as enabling automated support for syntactic and semantic correctness.

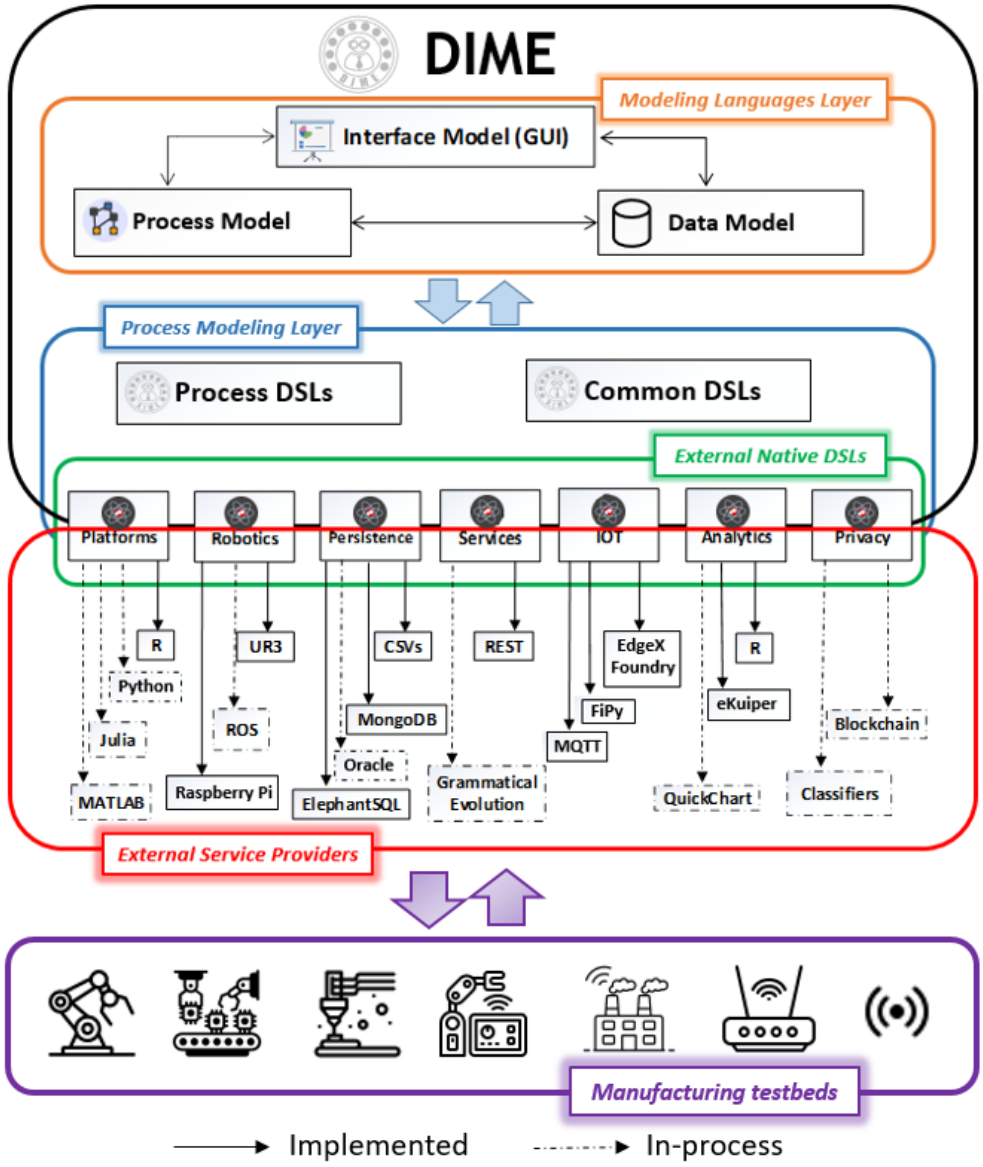


Figure 9.1: Architecture overview of the low-code Digital Thread Platform.

We discuss as a case study the development of an IoT application including Edge Analytics by using two low-code platforms. We consider here the scenario

of controlling and maintaining the environmental conditions of a storage facility as an example of a Stable Storage Facility (SSF), where the temperature, humidity and light intensity levels need to be maintained at the required levels in the facility. We leverage the usage of two low/no-code development environments, DIME [2] and Pyrus [3] to extend the horizon of services, platforms and the domain-specific functionalities that they support. Figure 9.1 shows the architecture of the Digital Thread platform, where the modeling Languages layer provides the essential modeling capabilities, e.g., for the GUI, data persistence, etc. The Process modeling layer models the business logic for the application in combination with already implemented Common and External Native Domain Specific Languages. External Native DSLs are the set of services, technologies and platforms that reside on top of the standard DIME distribution. The different categories of functionalities that the platform supports are either already implemented/integrated (solid arrows among the External Service Providers) or in the testing phase/in progress (dotted arrows).

The rest of the paper is organised as follows: Section 9.2 summarises related work, Section 9.3 gives an overview of the software platforms adopted for the use case, Section 9.4 discusses the use case of a stable storage facility, Section 9.5 discusses the results, and Section 9.6 concludes and outlines future work.

9.2 Related Work

The widespread availability of technologies such as IoT, Cloud Computing, and communication protocols (e.g., Bluetooth, WiFi, LoRaWAN, etc.) has revolutionised the IT industry by enabling the decentralisation of not only data locality but also computing resources. This allows improvement in large-scale data management in real-time and leads towards a high degree of innovation and simplicity in development workflows. Low-code development platforms are increasingly being adopted by the Information technology (IT) industry and enabling rapid development of complex workflows and software solutions by following a model-like paradigm, which focuses on composition behaviours rather than boilerplate code. In a worldwide survey, 96% of the IT professionals agreed that low-code development in comparison with traditional development is much faster [182] and

expected to generate approximately 65 billion U.S. dollars revenue globally by 2027 within the low-code platforms market [183], which also demonstrates the potential and capabilities of this paradigm.

9.2.1 General Purpose Low-Code Platforms

Many industrial solutions support the visual composition of the orchestrations, which is more intuitive than manual coding. Several tools are available in the industry to work in a low-code fashion, e.g., Tines [107], an Irish-based company providing automatised solutions for security prevention; PTC ThingWorx [184], an all-in-one IIoT platform developed by PTC providing a pre-built solution for several requirements in the context of manufacturing; AWS IoT [185], a popular and scalable solution to handle up to billions of devices in the IoT context, allowing a cross-integration of capabilities with different services such as ML, analytics for Data-Driven decisions, etc; Microsoft Azure IoT Suite, another solution from Microsoft delivering integration with several services, not only with ML and analytics, but also adding security layers and connecting, monitoring and controlling in the cloud up to billions of devices as well [186] or H2o.ai [187], a complete no-code platform to deliver AI/ML solutions in the cloud targeting non-technical people to be able to complete working pipelines. Many of these tools either support control-flow or dataflow models, but not both, and do not have formal models incorporated.

Several use cases also have appeared in academia, to work in different domains to rapidly fulfil these requirements in a low-code way. For instance, [188] proposes an AI engineering platform for smart IoT services, enabling data analytics and ML parts to run partially or completely on the IoT edge devices. In [189], the authors establish an IoT-enabled application with early analysis of performance characteristics, such as timing and performance properties. This approach has a direct application in identifying issues with performance attributes, decreasing development time/effort, and delivering better quality without budget overshooting. The application itself is being developed using a lightweight interface tool framework, developed with the Eclipse Modeling Framework (EMF) [190].

9.2.2 Model-Driven Development

Model-driven development is a type of low-code development approach that focuses on the use of models to automate the software development process. In model-driven approaches, code generators are typically utilised to generate code automatically from higher-level specifications [191]. An interesting case for MDD IoT simulators can be found in [83], where new projects require a highly specialised setup (devices, fog/cloud/analytic nodes, hardware and software) and, together with the development cost, could be quite expensive. The AToMPM tool [192] is a web-based open-source framework to design Domain Specific Language (DSL) environments, perform model transformations and manipulate and manage models [193]. It also has an API for binding Python language. Component-based development [86, 87] is another similar paradigm where independently deployable components are developed and then integrated into the bigger ecosystem. The modeling in component-based development, referred to as component modeling, defines the standards for the composition, modeling and deployment mechanism for the implementations of components. The authors of [194, 195] used UML annotations for component models while [89] developed their own modeling languages. The PROGRESS project [89] defines the component models in the domain of embedded systems and consists of two modeling layers to handle different kinds of operation, i.e., ProSys models the systems and subsystems while ProSave captures the data transfer and control flow between the components. The functional behaviours of components are defined using REMES [196], a hierarchical modeling language.

Domain-Specific Languages in the context of low-code platforms typically include pre-built components, templates, visual interfaces for designing and arranging application components and tools for integrating with external data sources and systems. Some prominent platforms that use DSLs are OutSystems [197] for enterprise applications, Mendix [198] for mobile applications, Appian [199] for business processes, Cinco [25] for metamodeling and [200] for big data workflows.

To make this more cost-efficient, simulation environments can help to model and obtain more useful insights about the architecture and focus on high-level concepts. Moreover, these types of architectures require programming skills to

create the simulation environments. In order to make it accessible to everyone and enable the design of complex IoT simulation environments without writing code, a graphical and model-to-text approach was developed. Furthermore, these model-based languages often have a lower barrier to entry since they utilise graphical symbols and diagrams instead of code, which reduces the overall learning curve for domain experts. The study [1] compared various language workbenches with Cinco-family products and concluded that the LDE approach through DSLs overcomes the challenge of an extremely steep learning curve associated with development with traditional approaches.

9.2.3 DIME

DIME [2] is an Eclipse-based Integrated Modeling Environment that provides graphical DSLs for the development and deployment of prototype-driven web applications. DIME follows the One Thing Approach (OTA) [98] and the eXtreme Model Driven Design (XMDD) paradigm [99] for modeling and development. It provides a family of model types that are programming Language DSLs in order to tailor the environment to specific application domains. The different models are the following:

- Data Model: To describe the database layer.
- Process Model: This model type defines the business logic of the application. It uses control and data flow together to give a detailed view of the described logic.
- UI Model: A WYSIWYG editor for the web user interface.

The collections of components that can appear in the process models can be extended with Java code through a Native Service Independent Building Blocks (SIBs) mechanism, enabling the definition of new SIB palettes. SIB palettes are typically domain-specific and grow over time due to incremental extensions. These native DSLs are collections of ready-to-use functionalities, as described in Section 9.3.3 for our SSF use case.

9.2.4 Pyrus

Pyrus [3] is a web-based graphical modeling platform specialised for Data Analytics that supports pipelines in a data-flow-driven fashion. The Python functions are implemented, stored and executed in Jupyter and displayed as a family of reusable DSLs in Pyrus. On execution, Pyrus performs automated model-to-code transformation on designed pipelines and executes them on Jupyter instance.

9.2.5 Node-RED

Node-RED [201] is a JavaScript-based, low-code platform, widely used for IoT projects. Originally developed by IBM, it is now also fully open source. The Node-RED modeling style is based on Flows, which are simple control workflows that can contain multiple operations (see Figure 9.2). Each operation is started by a trigger node, and successor nodes are linked via flow edges. If a node has multiple successors, they are executed in parallel. Node-RED flows are directly deployable from the flow editor in a single-click fashion. The dataflow is not visually presented and is implemented through JavaScript Object Notation (JSON) messages passed along the control flow edges.

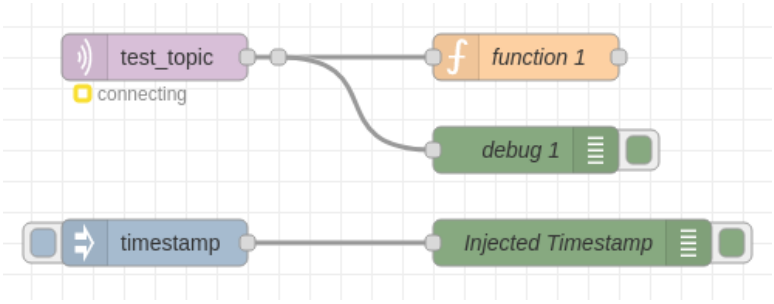
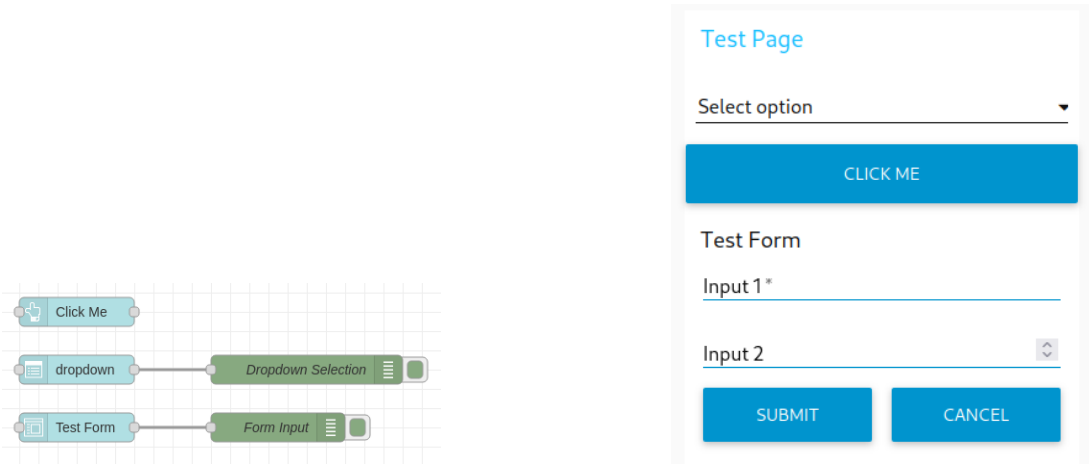


Figure 9.2: Example of a flow in Node Red.

Node-RED comes with a default set of available blocks and functions, but can easily be extended by adding modules to the Node-RED instances. As Node-RED has a very active community, there are many modules available, which can be installed directly through the UI interface. There is no notion of a data layer or database for the created flows, but there are modules to interact with files and databases.



(a) Simple UI model in Node-RED

(b) The Node-RED dashboard created by the UI Model (** indicates a required field)*

Figure 9.3: Example of a UI definition in Node-RED.

There is no out-of-the-box UI element for the flows, but several UI modules are available. The Node-RED Dashboard is the de facto standard tool to create simple interactive UIs (see Figure 9.3). In the Node-RED Dashboard, the UI elements are introduced via node inflows and arranged via different node properties. There is no WYSIWYG editor for the UI, and the flow developer must specify the position of the element within a fixed page, tab, or row-like structure. This limits the available ways to position elements, but it creates very consistent UIs.

9.2.6 Embedded Hardware Platforms

A plethora of low-power embedded computing and wireless sensor platforms have been developed over the past two decades by academic researchers, commercial enterprises and the Maker community. These platforms range from single board computers such as RaspberryPi, Beaglebone, and BananaPi to low-power, small-scale sensor nodes such as Rene, Mica, MicaZ, TelosB, IRIS, SunSPOT, the latter developed to study wireless sensor networks and their applications. The Maker community is served by several companies that develop and supply low-power embedded systems boards and platforms under labels such as Arduino,

Adafruit, Sparkfun, and PyCom. The Pycom IoT device consists of a wireless module (Fipy) mounted on a Pysense expansion board equipped with several sensors: a temperature, humidity (SI7006A20) and light sensor (LTR329ALS01). The light sensor typically provides two channels labelled as ctrl_ch1 and ctrl_ch2, etc. to measure light intensity in different parts of the spectrum: one for visible light and one for infrared light. The outputs of these two channels are then combined to determine the overall light intensity in the environment. These computing platforms use microcontroller technologies from ARM, NXP, ST Microelectronics, Nordic, TI, or Espressif among others. They include wireless chipsets based on IEEE802.11 (Wi-Fi), Bluetooth (standard and BLE), IEEE802.15.4 and wide-area wireless technologies such as LoRa, Sigfox, NB-IoT and LTE-M. A wide range of commercial low-power embedded systems platforms are also available [202, 203], but they tend to be designed for specific applications and use cases. A broad range of companies from system solutions providers to chip manufacturers also provide similar low-power embedded hardware and prototyping boards. The space is particularly vibrant with small companies and start-ups. A comprehensive survey of academic prototypes and commercially available wireless sensor platforms for Internet of Things applications can be found in [204].

9.3 The Software Platforms

In this section, we describe our technology stack for this case study and why we chose those technologies. Further, we show how the Native SIB concept leads to the production of reusable SIB palettes.

9.3.1 The Two LC/NC Integrated Modeling Environments

For this work, DIME and Pyrus are used as the modeling environments of choice. The combined view of control and data flow in DIME makes it easier to formulate complex workflows. Additionally, the interaction between DIME's data and process layer allows checking the data types used by the individual SIBs, which helps to ensure the correctness of the operations. However, the complexity of DIME can also be too much when one just wants to analyse the collected data. So,

Pyrus will be used for this step in the case study. The pure data flow approach of Pyrus makes it easy to clean, analyse and visualise the outcomes.

9.3.2 The External Software Platforms and Applications

EdgeX Foundry [4] is a leading open-source Industrial IoT middleware platform. It is widely used by industry and research teams for industrial automation. It helps exploit the benefits of the edge in terms of computing and intelligence. Its internal architecture has four service layers: (1) device services (2) core services, (3) supporting services, and (4) application services. A detailed description of all the layers and associated microservices can be found in [111, 112]. The EdgeX services needed for our SSF use case are provided in DIME as an EdgeX-specific native DSL.

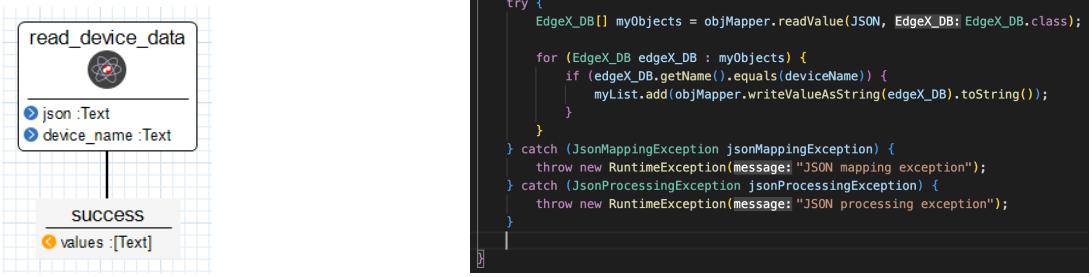
The **Atlas** NoSQL cloud database is a Database-as-a-Service product by MongoDB [205] that is optimised for complex data objects in real-time, which is one of the core needs of IoT applications. Hence, we use MongoDB to store/access the data collected from the Pycom FiPy sensors (light, temperature and humidity) via MongoDB-specific native DSL.

R [206] is an open-source programming language specifically for statistical computations and data visualisation. We use an R language native DSLs to analyse and visualise the datasets collected from different sensors.

Docker [108] is an open-source cutting-edge containerisation platform revolutionizing the way software applications are developed, deployed, and managed. With its lightweight, portable containers, Docker empowers developers to build, ship, and run applications seamlessly across diverse environments, ensuring consistency and scalability in the modern era of software development. We use Docker to securely deploy frameworks, services and platforms in separate containers.

9.3.3 The Native SIBs and SIBs Palettes

Figure 9.4(a) shows the visual representation of a SIB when used in a process model. The SIB requires two text inputs, a JSON string and a device_name, either from another SIB via data flow (dotted) edges or a static value.



(a) The SIB in DIME

(b) The SIB body implementation in Java

Figure 9.4: Native EdgeX palette: the read_device_data SIB in DIME [5].

The backend implementation of this SIB in Java is shown in Figure 9.4(b) and is invoked at runtime, it reads all the values of the database from the different devices connected to the middleware. If the execution is successful, the corresponding data output is made available and the outgoing control branch labelled success leads to the next SIB in the workflow.

The low-code part of the development process consists of extending the capabilities of the DIME platform through this integration of further native SIBs when developing a new application. Every SIB is developed once and integrated into the platform for further reuse. This way the development of further applications that require similar functionalities in systems becomes increasingly a no-code task that also leads to much more efficient and waste-free production of software, based on large-scale reuse of quality-assessed components.

9.4 The Use Case: Stable Storage Facility

We illustrate the application of our low code development platforms through a specific IoT use case: controlling and maintaining the environmental conditions of a storage facility, where the temperature, humidity and light intensity need to be maintained at the required levels SSF.

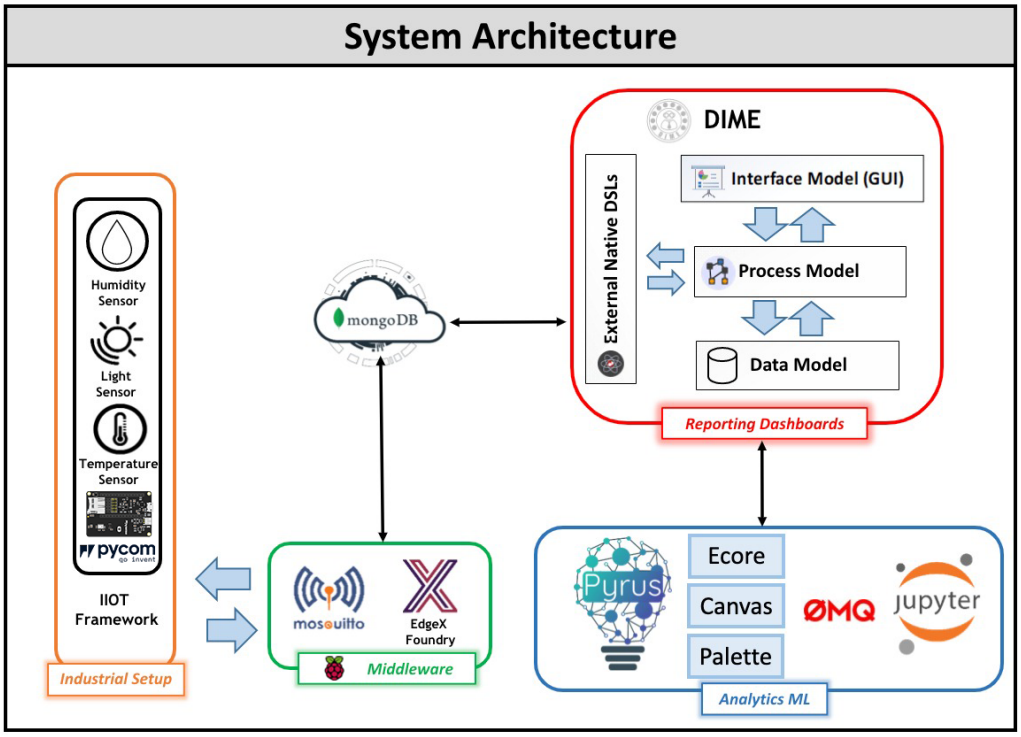


Figure 9.5: Stable Storage Facility (SSF): runtime architecture, infrastructure and communications

9.4.1 The SSF IoT Architecture

Figure 9.5 shows the run-time architecture of the SSF application. We use Pycom IoT devices to monitor the environment. In the experimental setting, Pycom FiPy devices are installed in the test environment (Lero building at the University of Limerick) to record the observations at fixed intervals. Each device periodically reports the data, i.e., light, humidity and temperature values, to an on-premise edge device. These devices send the data to the middleware platform (i.e., edge device with EdgeX Foundry listener) via the MQTT protocol that stores it in a no-SQL database in a MongoDB Atlas instance. Based on the received sensor data, the corresponding controlling actions can be enabled to manage the stability of conditions in the SSF. On the right side of Figure 9.5, we find the controller application. It is a web application developed in a low-code fashion using DIME and deployed into a dedicated Docker container. The web application manages

all the devices and maintains stable environmental conditions. The web controller initiates the communication with the local MongoDB Atlas database and manages the data mobility with EdgeX Foundry over REST protocol and finally communicates internally with the R docker container for data-related computations and visualisations. We then implement the same data analytics pipelines in the Pyrus dataflow environment and then extend the analytics with more advanced features.

This use case requires the integration and interoperation of a variety of systems, devices, software platforms, communication protocols and runtime environments. They are successfully orchestrated to produce a visualisation, an understanding and an interpretation of the data, in two different low-code platforms.

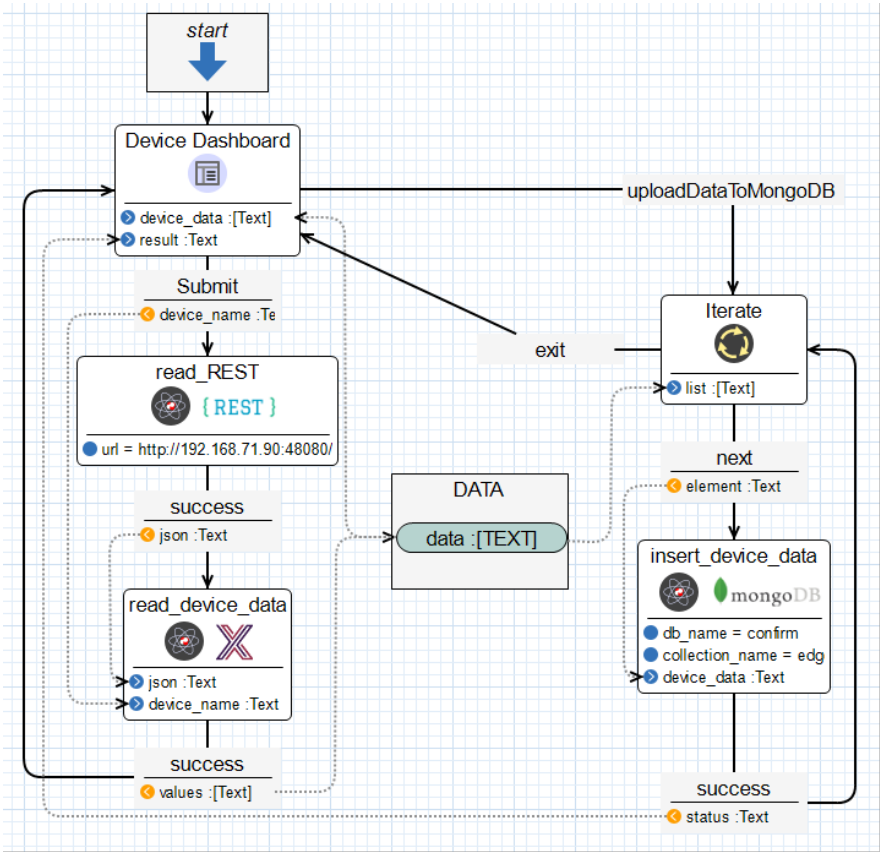


Figure 9.6: Data acquisition from EdgeX Foundry and data ingestion into MongoDB process in DIME [5].

9.4.2 Data Acquisition Process Model

Figure 9.6 shows the process model in DIME for the data acquisition from the IoT devices and its ingestion into MongoDB. In this model, the workflow starts with start SIB followed by a hierarchical GUI SIB for user interactions concerning device selection. The control flow then moves to the next SIB read_REST, reads the device data from the EdgeX Foundry instance and retrieves the corresponding raw data. For data collection, the SIB read_device_data, previously discussed in Section 9.3.3, parses the raw data, i.e., sensors data in JSON format from the middleware platform and returns the parsed data back to the GUI, which is then displayed on the web page. In the case of the storage requirements by the user, the control flow in Figure 9.6 proceeds to the right side of the workflow and performs the data ingestion into the MongoDB cloud database.

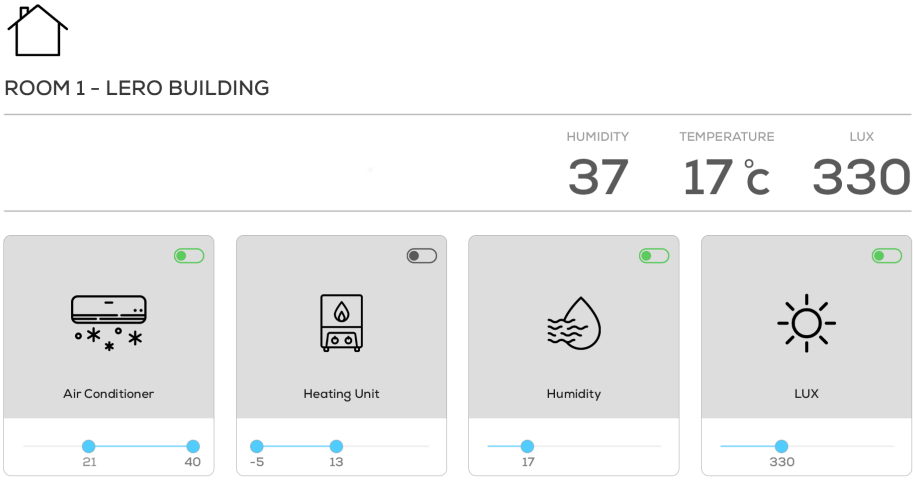


Figure 9.7: The control application: the GUI

9.4.3 The SSF Control Application in DIME

Figure 9.7 shows the web page for the control application, where the header section displays the data collected from the different sensors: the temperature, humidity and light sensors. In the grid section, each "card" controls one of the actuators depending on the respective value/range that is selected using the range

setter. Accordingly, all the sensors are continuously monitored. Whenever a sensor reaches one of its set threshold values, the controller will send the appropriate command to turn on/off the respective controlling unit: the cooling or heating unit in case of a temperature threshold, the humidifier in case of a humidity threshold and the lights based on the lux values. Additionally, the user also has the option to enable/disable selected devices using the toggle button visible at the top right of each card. For example, in this view, the Heating Unit is switched off while the other units are active. In this way, the application helps the users to optimise the environmental conditions of the Stable Storage Facility automatically with maximum efficiency.

9.4.4 The Reporting Dashboards in DIME and Pyrus

DIME: Figure 9.8 shows the process model for Analytics Dashboard in the DIME application, where connection_mongoDB SIB fetches the required data from storage and passes it to the split_data_frame SIB along with the column names that require splitting, e.g., in this case, a total of four channels from two devices, i.e., light_ch1, light_ch2, light_ch3 and light_ch4. The next SIBs plot the graphs against these data frames, as shown in Figure 9.9. The plots are then displayed on the web application using the GUI SIB analytics_dashboard.

Pyrus: Now we discuss how we carry out essentially the same computations in Python using another low-code/no-code platform: Pyrus. Figure 9.10 shows an alternative data processing pipeline implemented in the Pyrus. The pipeline starts by establishing a connection to the MongoDB with the connection_to_mongoDB block with the required inputs. It again fetches the required data in JSON format and the data frame is passed to the next block, convert_to_datetime for preprocessing. The data frame and the names of the channels are then passed to the block dataframe_split to create a separate data frame for each light channel. Finally, the plot_all_data block plots the same graphs in Pyrus as shown in Figure 9.9.

Further, we extended the analytics capabilities of the pipeline from Figure 9.10 with additional parameters, as shown in Figure 9.11. This new pipeline analyses the data collected from light, humidity and temperature sensors. The data are

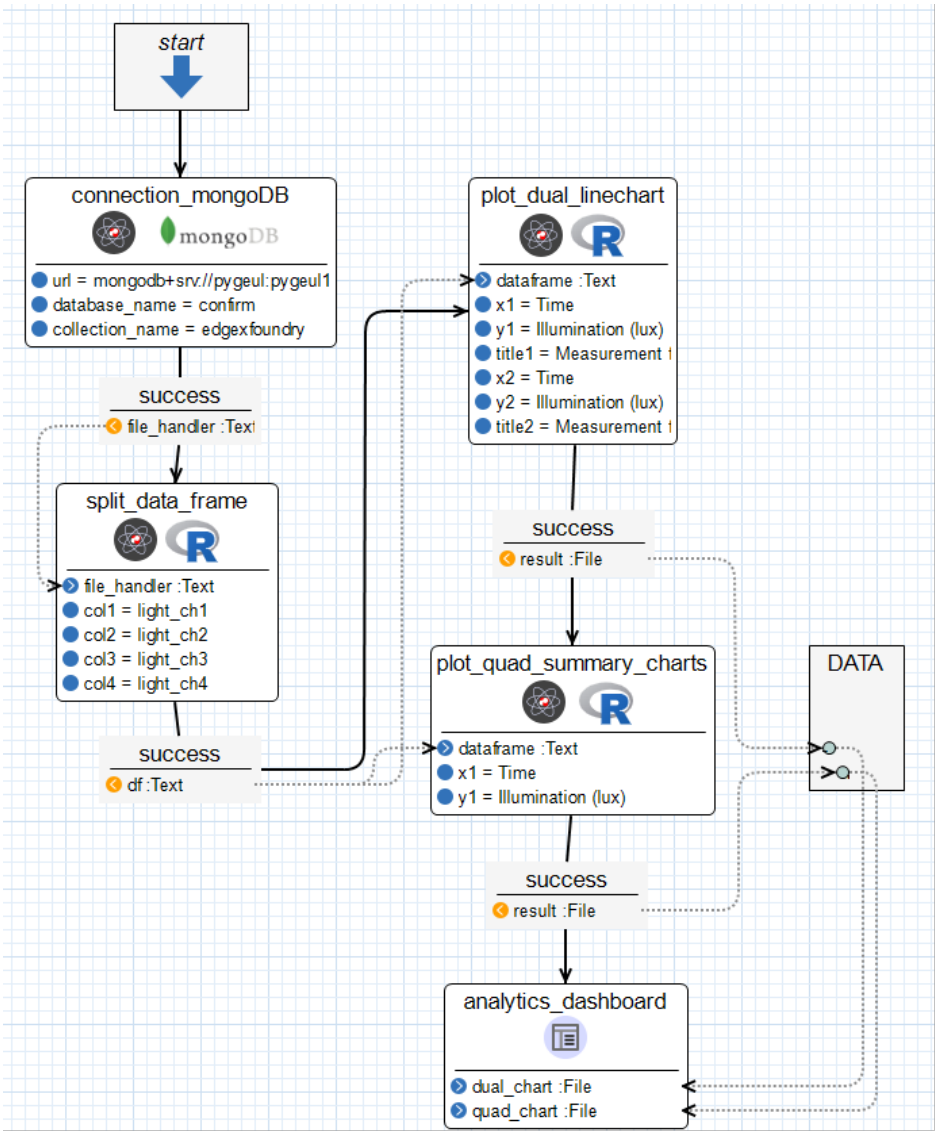


Figure 9.8: Analytics dashboard in R: the process in DIME [5].

directly acquired from the MongoDB database using the connection_to_mongoDB block. The data are then pre-processed by dropping the irrelevant columns, renaming existing columns and converting the time column to Pandas datetime format using the clean_df block. Since the data are collected from the devices over the span of several days, a range of dates can be selected for analysis. This happens using the date_mask block, where the sample date of 18 November 2022 is input. The final plot_all_data block plots all the processed data frames in the

Pyrus dashboard, as shown in Figure 9.12.

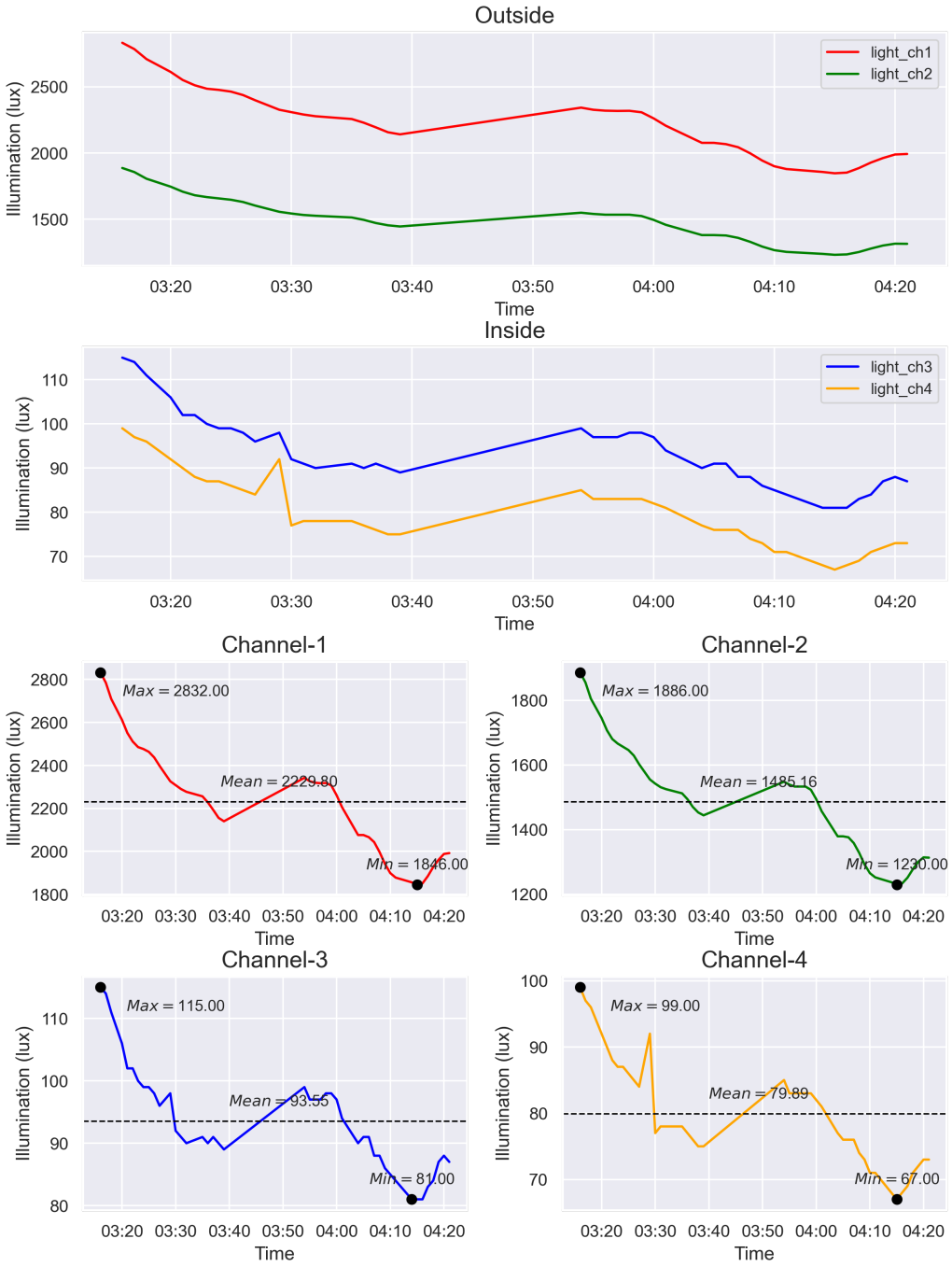


Figure 9.9: Dual and quad summary line plots generated using R and the DIME DSLs and processes [5].

9. PAPER IV (AP4)

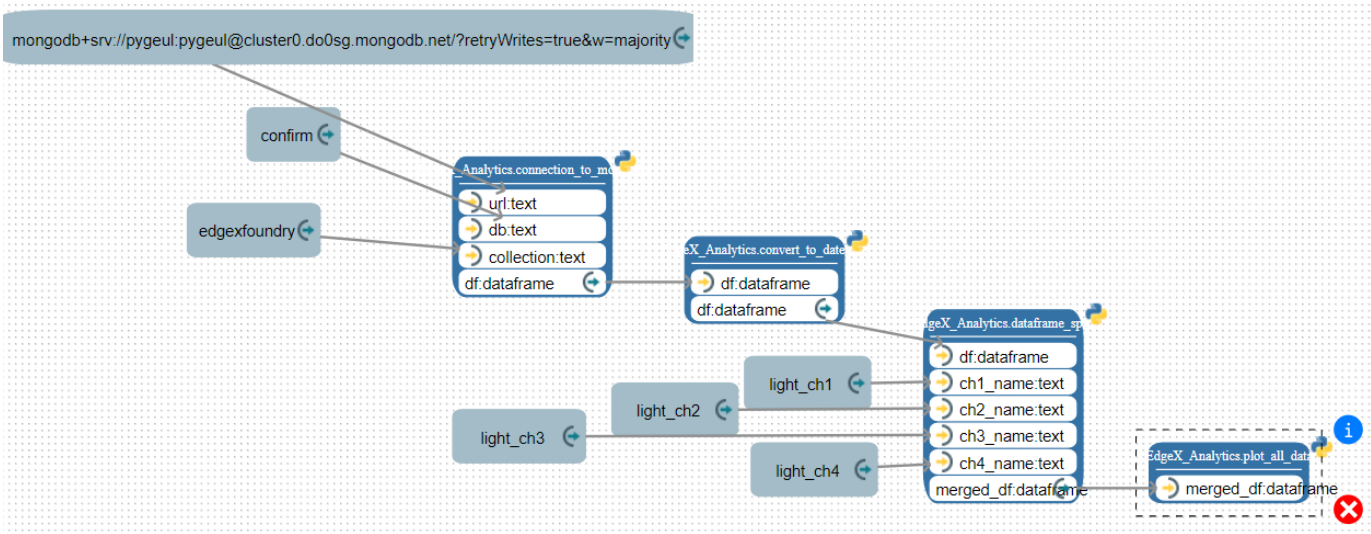


Figure 9.10: Analytics dashboard in Python: the Pyrus pipeline [5].

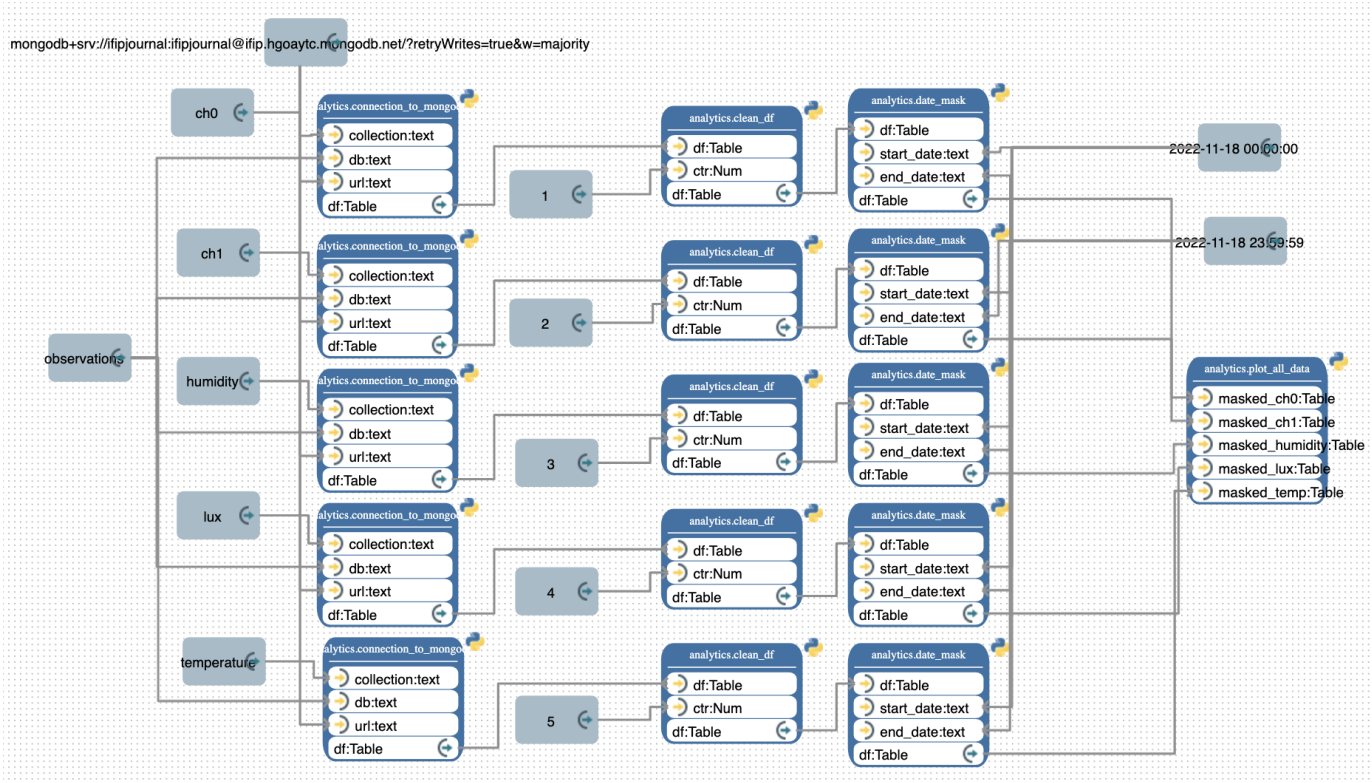


Figure 9.11: Extended Pyrus pipeline.

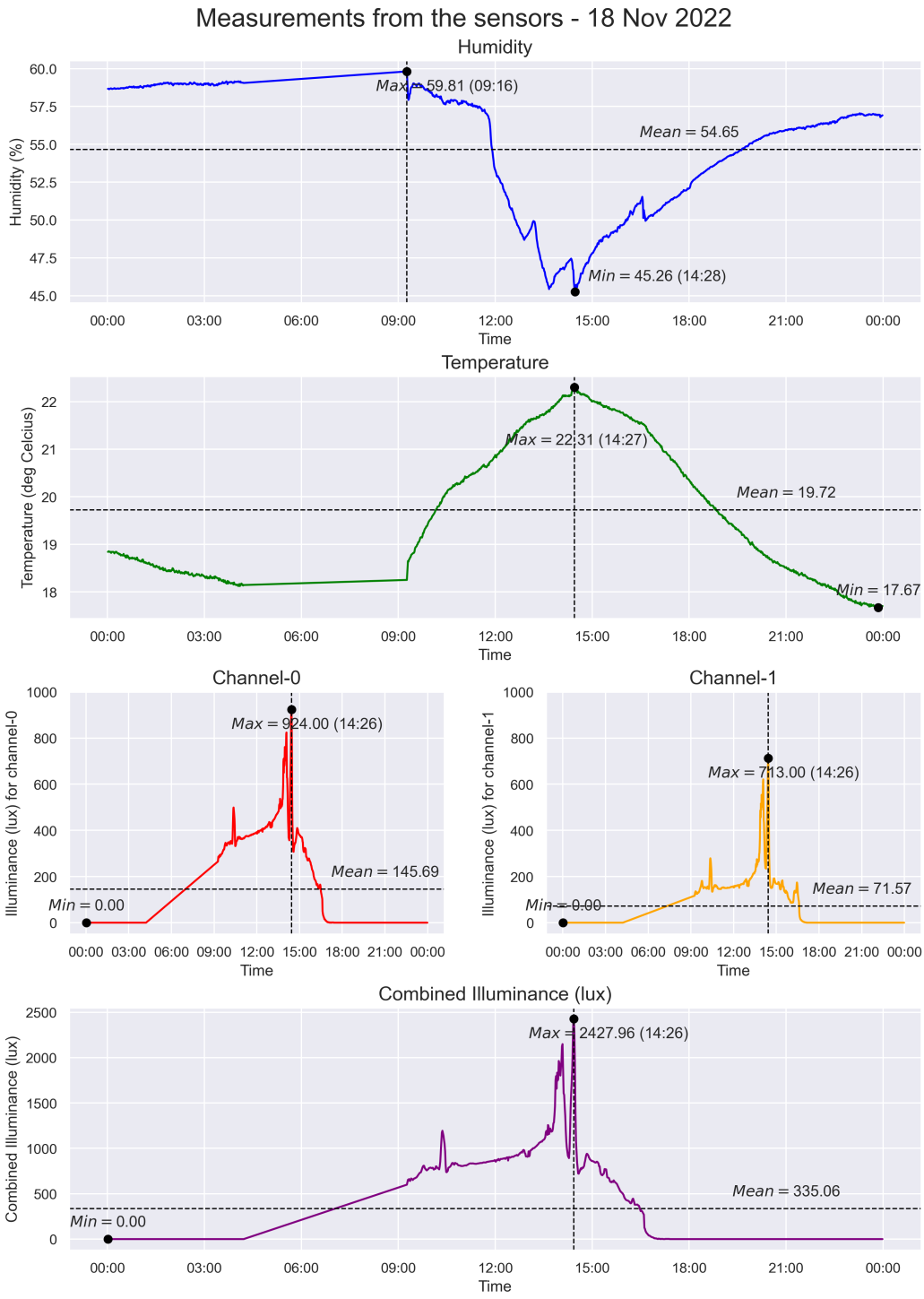


Figure 9.12: Advanced analytics visualisations generated using Python and Pyrus.

9.5 Results and Discussion

The presented case study concretises a heterogeneous architecture for the development of low-code IoT applications, where sensor devices interact with an orchestrator and an analytics server to automatically control the environmental parameters via a web controller based on the sensors' input, and eventually facilitate the decision-making process with the help of data analytics, which enables the storage facility supervisors to take future decisions. In the concrete experimental setting, we consider the scenario of controlling and maintaining the environmental conditions of a SSF where the temperature, humidity and light intensity levels need to be maintained at the required levels.

9.5.1 Platform Extension Through New DSLs

We have extended the Native DSLs for EdgeX Foundry, REST, R, and MongoDB, while also addressing MQTT and Raspberry Pi. From the DIME processes, we communicate with the EdgeX Foundry services via REST protocol without any manual integrations. For the sensor devices, we use standard APIs to capture and send data to the EdgeX Foundry server using the MQTT protocol over WiFi. Altogether, we have integrated nine independent technologies in two distinct Low-code development environments with the production of seven processes and pipelines, and the definition of 25 SIBs in 9 distinct DSLs. The significance of this approach lies in its contribution to the extent of the supported functionalities in the Digital Thread Platform of Figure 9.1.

9.5.2 Benchmarking Code Generation

Embracing a platform mindset results in a single implementation of functionality with high reusability across multiple domains by very different domain experts. Additionally, a large part of the code is automatically generated from the models. To showcase these effects, we benchmark two of the implemented case studies: the SSF presented in this work and the predictive maintenance case study presented in [30].

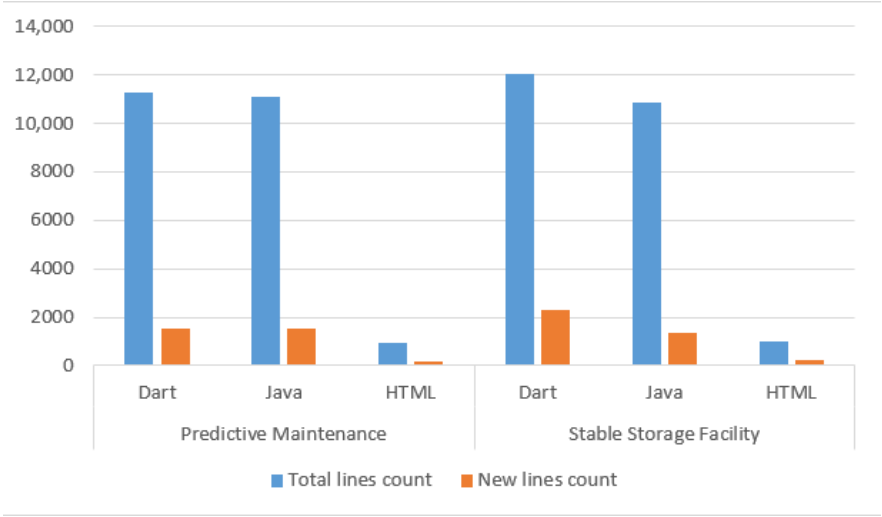


Figure 9.13: Total lines of code vs. newly added functionality code in both DTP applications

Figure 9.13 shows the count of total lines of code generated (blue bars) for the implemented pipelines vs. the lines of code manually implemented (orange bars) as new SIBs. In Table 9.1, we see that for all three languages (Dart for the frontend, Java for the business logic and HTML for the UI), the new code written manually is a fraction of the total code. For Java in our use case, we have 1326 new lines of code in comparison with 10,889 resulting from the generation, which links also preexisting SIBs. This means that we wrote only 12.18% of the new code needed for this application, and the data show that this proportion is

	Predictive Maintenance			**Stable Storage Facility**		
	Dart	Java	HTML	Dart	Java	HTML
Total lines of code generated	11250	11096	943	12037	10889	1006
Total lines of code implemented	1517	1533	190	2304	1326	253
Percentage of code implemented manually	13.48%	13.82%	20.15%	19.14%	12.18%	25.15%

Table 9.1: Code stats for DTP applications

similar across languages and case studies. This comparison demonstrates that the dependency on software developers for new code is greatly reduced because we leverage the generation from models and the availability of the platform and prior DSLs.

9.5.3 Effects on Reuse

The encapsulation of this heterogeneity within the DSLs via uniform graphical representation in DIME makes it possible for domain experts to correctly identify and reuse large portions of functionality across different applications. Identifying features in code with the goal of encapsulation and reuse is a well-known nightmare at the code level, as it requires reverse engineering of code and dependencies. This is a core reason why even very common functionality is re-implemented over and over again. With our low-code, graphical and generation-based approach, we avoid the problem by learning to work with only three DIME model types. In addition, domain experts can create high-quality applications spanning across various application domains and technologies without requiring the mastery of the underlying technologies, programming languages, or communication protocols.

In terms of relevance to the IoT user community, this is a transformative contribution where, rather than requiring expertise in a multitude of diverse technologies, users only need training on these low-code platforms with small learning curves because of the simpler abstractions compared to traditional programming languages. After initial training, IoT application developers can effortlessly design, deploy, maintain, and evolve their applications in a uniform environment through platforms like DIME and Pyrus. The proposed approach can effectively support the development and deployment of small to large-scale IoT applications using the multi-node Edge-X deployment.

In terms of the choice of models, we chose to combine simplicity [177] with formality [122, 207], to enable ease of comprehension by a wide range of specialists and at the same time pave the way to automated verification and validation along the lines of [208] for model checking, [209] for automated synthesis, [122, 210] for automated testing and validation, and [103] for the automated support of CI/CD.

Altogether, the long-term goal is to support the full lifecycle of continuous model-driven engineering [211].

9.6 Conclusions and Future Work

Domain-specific languages in a low-code and underlying model-driven paradigm have become a popular approach to design, develop and quickly ship heterogeneous systems. In this use case, we developed a heterogeneous system that connects various technologies: the web application interacts through the EdgeX middleware platform with several sensors and data analytics pipelines. We presented a specific IoT use case: controlling and maintaining the environmental conditions of a storage facility, where the temperature, humidity and light intensity need to be maintained at the required levels SSF. The edge computing and analytics-related SIBs are developed and added to the suite of External Native DSLs library of the Digital Thread Platform. The developed application collects, processes and displays key information about the state of the edge data capture and computing. We use DIME to design the complete application and to implement the IoT and edge aspects, and Pyrus to implement no-code data analytics in Python. The developed case study is benchmarked and compared with prior case studies to showcase the aspects of reusability. We demonstrate and conclude that both are straightforward approaches in which domain experts can create IoT applications in a low-code fashion. Our next steps include the extension of the supported devices, protocols and services within the capabilities of the Digital Thread Platform.

Funding

This project received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Smart 4.0 Co-Fund, grant agreement No. 847577; and research grants from Science Foundation Ireland (SFI) under Grant Number 16/RC/3918 (CONFIRM Centre for Smart Manufacturing), 13/RC/2094-1 (Lero, the Software Research Centre) and 18/CRT/6223 (SFI Centre of Research Training in AI).

9. PAPER IV (AP4)

10

Paper V (AP5)

Model-Driven Engineering in Digital Thread Platforms: A Practical Use Case and Future Challenges

Hafiz Ahmad Awais Chaudhary, Ivan Guevara, Jobish John, Amandeep Singh, Amrita Ghosal, Dirk Pesch and Tiziana Margaria

Publication Report

- **Publication Date:** 10/2022
- **Conference:** International Symposium on Leveraging Applications of Formal Methods
- **Venue:** Rhodes, Greece
- **Book:** Leveraging Applications of Formal Methods, Verification and Validation. Practice
- **Publisher:** Springer Nature Switzerland
- **DOI:** https://doi.org/10.1007/978-3-031-19762-8_14

Contributions Summary: *Conceptualization, All authors; architectur design, H.A.A.Chaudhary, I.Guevara, A.Singh, and J.John; pipelines design and imple-*

mentation, H.A.A.Chaudhary, I.Guevara and A.Singh; case-study implementation, H.A.A.Chaudhary, I.Guevara, A.Singh and J.John; resources, H.A.A.Chaudhary, I.Guevara, A.Singh and J.John; hardware setup, D.Pesch, J.John, H.A.A.Chaudhary, I.Guevara and A.Singh; Access control policies, A. Ghosal; writing—original draft preparation, I.Guevara, A.Singh, H.A.A.Chaudhary, J.John and A. Ghosal; writing—review and editing, T.Margaria and D.Pesch; supervision, T.Margaria and D.Pesch; project administration, H.A.A. Chaudhary, I.Guevara and T.Margaria; funding acquisition, T.Margaria and D.Pesch.

Abstract

The increasing complexity delivered by the heterogeneity of the cyber-physical systems is being addressed and decoded by edge technologies, IoT development, robotics, digital twin engineering, and AI. Nevertheless, tackling the orchestration of these complex ecosystems has become a challenging problem. Specially the inherent entanglement of the different emerging technologies makes it hard to maintain and scale such ecosystems. In this context, the usage of model-driven engineering as a more abstract form of glue-code, replacing the boilerplate fashion, has improved the software development lifecycle, democratising the access to and use of the aforementioned technologies. In this paper, we present a practical use case in the context of Smart Manufacturing, where we use several platforms as providers of a high-level abstraction layer, as well as security measures, allowing a more efficient system construction and interoperability.

10.1 Introduction

In the actual data-centric era, where the shift towards distributed and ubiquitous architectures demands increasing computing capabilities, cloud-based processing represents a bottleneck. This is due to the increasing amount of devices taking part in the systems, which are often systems of systems. According to the Cisco Annual Report 2018-2023 [212], IoT devices will account for 50 percent (14.7 billion) of all global networked devices by 2023, having an impact on the workload processing and forcing companies and organizations to look for more cost-effective and efficient alternatives. In this context, since the data is generated at the edge of the network, i.e., by the IoT devices, it would be more efficient if the processing of the data also happens at the edge. This is called Edge computing [53]. Edge computing plays a significant role in the Industrial IoT (IIoT) sector, lowering the cost of data transport, decreasing latency and improving the overall efficiency of the architecture [213]. This does not mean that cloud and edge computing are incompatible, rather they complement each other, allowing us to have more tools to confront these issues and be able to improve the overall performance in a balanced and customized way.

Based on physical configurations and functional requirements in the context of smart manufacturing, there are many standard architectures for the implementation of IoT based systems. ISA-95[IEC 62264-1:2013] [214]is one of the international standards for enterprise control system integration that defines both physical arrangements and functional hierarchies from device to device communication to functional management at different granular levels. FIWARE [215] platform is an EU initiative towards the development of smart applications in manufacturing with a set of standardized APIs. Similarly, EdgeX Foundry [4] is an open source standardized interoperability framework for IIoT Edge computing and Gaia-X [216] is a European standard for the development of next generation of data oriented infrastructures.

Shifting massive amounts of data towards the edge has also a downside: it requires to reconfigure the computation architecture in order to leverage computations on the edge components, and, as a consequence, also the ability to orchestrate the different heterogeneous responses from each SDK brought in by

devices of the ecosystem. This is a massive ask to the ability to integrate not just data, but processes whose bits and pieces are often buried in the SDKs. Another challenging task is the integration of edge intelligence[31], where the cost of delivering such solutions could be high if the advantages and limitations of this technology are not taken into account.

Model driven development (MDD) is one of the key approaches to develop heterogeneous systems from the conceptual modelling design to automated model-to-code transformations [22]. The main goal [23] of MDD is to develop rapid applications, that are flexible and adaptive to continuously changing requirements. The goal of this case study is to rely on model-driven capabilities as far as it is possible and convenient, i.e., without forcing the entire ecosystem to be integrated into a single platform. We use here the EdgeX Foundry platform as a middleware for IIoT components for data acquisition from heterogeneous sensors. Additionally, we use three low-code platforms for the development of functional pipelines: Tines for notifications among systems, Pyrus for data analytics and the DIME platform for process modeling and data reporting, empowering prototype-driven application development. All three follow to different extents the XMDD paradigm [138], where the technical details of the communication with a component or subsystem are encapsulated in high-level models, and this abstraction is useful in order to rapidly bootstrap a workflow and enable non-expert programmers to responsibly and directly participate in the software development cycle.

In the following, Sect. 10.2, discusses the industrial use-case with its system architecture, Sect. 10.3 covers the secure access policies and encryption techniques, followed by our conclusions and reflections in Sect. 10.4.

10.2 Industrial Use-Case: Safe Operation of Machines

In this section, we detail an industrial use case associated with the safe operation of a machine. Most manufacturing industries are equipped with complex

machines on their shop floor, such as computer numerical control (CNC), coordinate measuring machines (CMM), 3D printers, etc. They are monitored and controlled through an industrial automation network in addition to the human-machine interface. A safe, healthy operating environment is essential in such factory and laboratory areas. The machine operators are expected to comply with several health and safety measures, such as wearing gloves, glasses, boots, aprons and hats where opportune. For example, in certain areas on the production floor protection measures such as boots and glasses are mandatory.

The machine operators however may not always comply and follow all the safety measures, incurring higher safety risks. This is a recognized occupational hazard, and companies and organizations have a high interest in minimizing such risks and hazards. In this direction, an automatic system [217] consists of wearable devices and an NVIDIA Jetson board is proposed in an industrial scenario for monitoring the activities of personals and the behaviours of robotic systems.

One way to address non-compliance is by monitoring the production floor with cameras that continuously monitor these areas for proper behaviour along with health and safety guidance, and also for assistance in case of need. Several of these machines have inbuilt sensors that measure various parameters such as vibrations, temperature, pressure etc. In many cases, these parameters can be used as a marker to identify the health of machines and tools. Several industries follow additional digitization strategies by employing additional IIoT sensing modules. The machine/tool health status inferred from these sensor data, combined with the camera-based monitoring, can then ensure that proper health and safety measures for both machines and operators are followed in an industrial workyard. Security measures have to be taken as well, here we concentrate on attribute-based cryptography as a means to ensure that there are no leaks of business or privacy sensitive data.

10.2.1 Architecture of the Use Case

Fig. 10.1 shows the system architecture of the case study. Several different sensors installed in critical locations across the factory workyard monitor the working conditions on the floor of the Industrial Setup, on the left. Cameras are also installed

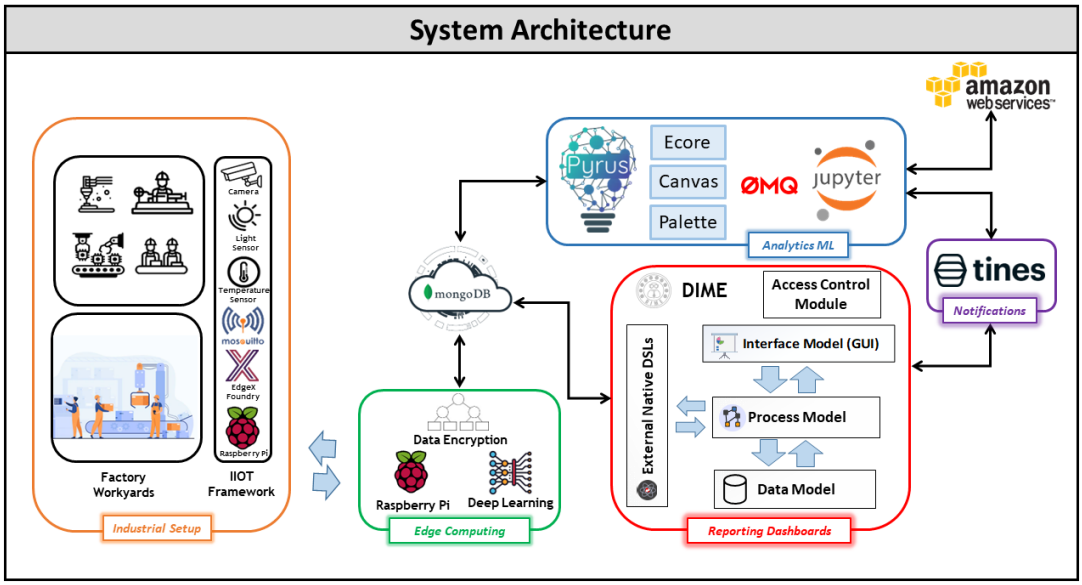


Figure 10.1: System architecture of the case study

to visually supervise the safety of the workers. When the workers are working with critical equipment, the machine and environmental parameters/conditions, e.g. vibrations, temperature, light, etc. are recorded using the deployed sensors, belonging to the IIoT Framework. In addition, the camera feed is also used to record the workyard at random time intervals (Factory Workyard). These sensors and cameras are connected by an Edge-controlling device, here a Raspberry Pi, which securely sends the collected data to another Raspberry Pi that acts as an Edge-computing and collection node for all the devices across the factory workyard. This computing/collection node is responsible for communication with all the data nodes and for sending this data to a reliable database, in our case a MongoDB Atlas instance. It also performs deep learning-based computations at the edge (the Edge Computing system). Since the computing/collection node is a Raspberry Pi, its computing capacity is limited and is only used for computations that require an extra layer of privacy, accountability and scrutability. We consider here the facial recognition of the workers for an attendance call, or the identification of workers present in the workyard.

From MongoDB, the data is accessed by the Analytics/ML system. There,

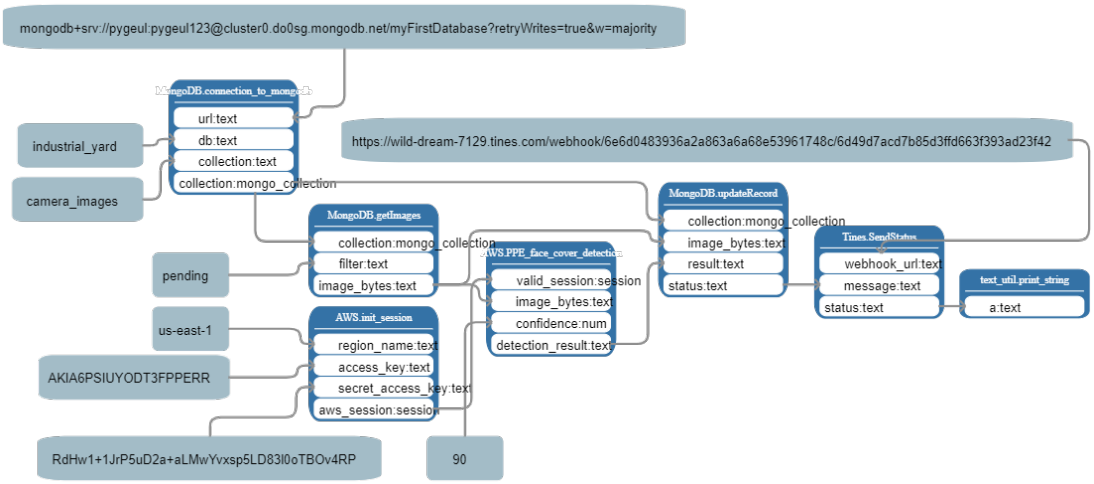


Figure 10.2: Pyrus PPE detection pipeline

Pyrus is responsible for periodically running ML pipelines at fixed intervals by communicating with the Amazon Rekognition API to detect the PPE/safety equipment as shown in Fig. 10.2. The results from these pipelines are a number of reports on whether all workers are satisfying the safety requirements of their respective tasks. They are sent to the MongoDB Atlas database for remote secure access by workyard supervisors. When the results are stored on MongoDB, a notification about this event is sent via the Tines automation pipeline as shown in Fig. 10.3. Triggered by the Tines web-hook notification, a web application implemented in DIME fetches the data from MongoDB and generates reporting dashboards that can be viewed by the supervisors to make critical decisions, creating adequate Reporting Dashboards. The Access Control Module implements an attribute-based mechanism that abstracts from individuals and specific entities, basing the access through specific cryptography on attributes, that can be roles or other elements of a profile.

10.2.2 The IT Ecosystem: Tools and Technologies

In this section, we briefly discuss the individual tools and technologies that form the heterogeneous ecosystem involved in this case study.

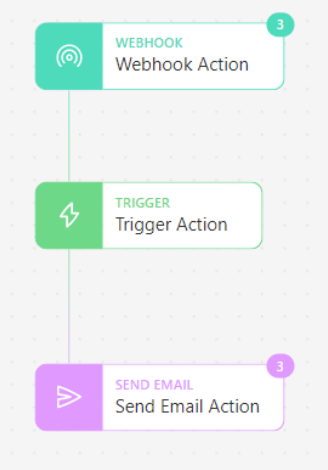


Figure 10.3: Tines automation pipeline

Raspberry Pi

The devices chosen for collection, computation and sharing of data in this research were Raspberry Pis due to their cost effectiveness, support for all devices used, support for Python and Linux, and availability of many interface options onboard. Two types of Raspberry Pis cover the two roles in this setup, as shown in Sect. 10.2.1:

- The Edge-controlling device is a Raspberry Pi Compute Module 3 (CM3) board connected with a StereoPi V0.9 carrier board [218] that is capable of connecting to two cameras on the same board using ribbon cables. The StereoPi device is running the *StereoPi Livestream Playground* v2 (SLP) image [219] that provides a consumer-friendly administration panel (similar to those in WiFi routers) without the need of setting up separate peripherals such as keyboard, mouse, monitor, etc., simplifying the setup and deployment. The SLP image also allows streaming the camera feed [219] to a UDP client through private IP address and port capturing using tools like *Gstreamer* [220]. Using this feature, the video feed is sent to the Edge-computing and collection node, from where it can be processed/stored according to the needs.

- The Edge-computing and collection device is a Raspberry Pi 4, model B with 4GB RAM. It is the latest series in the Raspberry Pi series of single board computers with a high-performance 64-bit quad-core processor.

Sensors and Devices

The most commonly used sensors for the machine or tool maintenance are vibration and temperature sensors. *EPH-V11, EPH-V17, EPH-V18, EPH-T20* [109] are some of the widely used sensors that provide vibration and temperature data over wireless communication. They report data over ModBus, which is one of the most widely used standards for industrial communication.

The cameras are Raspberry Pi *High Quality Camera* (HQCam) Modules with CGL interchangeable lenses [110]. The HQCam modules have 12.3 megapixel cameras, 7.9mm diagonal image size, support 12-bit RAW footage, have adjustable back focus and are compatible with C/CS mount lenses. The CGL lenses are 3 megapixel 6mm HD CCTV lenses with an inbuilt IR filter. The camera modules are connected to the Raspberry Pi through 200mm ribbon cables.

Edgex Foundry

EdgeX Foundry [4] is an open source, vendor neutral, flexible, inter-operable, software platform at the edge of the network, that interacts with the physical world of devices, sensors, actuators, and other IoT objects. It is used as a middleware integration and virtualization platform, as it directly connects with the IoT devices and exposes high-level services through a REST API. We have described the EdgeX integration in DIME in [5, 221].

DIME

The DIME [2] integrated modelling environment is a development environment to easily design, develop and deploy Web Applications in a low-code/no-code manner. It supports different model types (GUI, process and data models) that address different aspects of a Web Application. Built-in checks are supported both at the model and the project level for the purpose of debugging, as well

as one-click code generation and deployment. Its *External Native DSL* layer, described in detail in [30], provides the flexibility to extend the platform capabilities with external services and platforms. This is the capability that we exploit for the integration of external devices, tools and platforms. We have not integrated everything in DIME for a number of reasons. First of all, the DIME database and front-end components do not support complex data types like video streams. In addition, we wanted to show the interoperability among the different heterogeneous tools and technologies.

Pyrus

The Pyrus [3] web-based no-code collaborative platform for data analytics provides the support of basic data manipulation and analytics operations in a model-driven fashion. It represents the individual capabilities as a collection of taxonomically grouped SIBs (Service-Independent Building blocks) in its Ecore (name of SIBs palette) section. On the backend, Pyrus communicates over the ZeroMQ protocol [222] with the connected Jupyter Hub, initially for functions discovery of the available SIBs, then at runtime to call and execute the advanced Python programs that the SIBs represent as proxies. Technically, the Pyrus workflows are data-flow orchestrations of SIBs.

Tines

The Tines [107] story board is a no-code automation tool that was initially built for automating workflows (called stories) in the domain of security. It also works in a low-code approach, with seven 'actions', i.e. generic components that are similar to highly parametric SIB templates in the world of DIME and Pyrus. Its webhook actions are used as triggers, and its notification capabilities are domain-independent, so we use here a small story that implements an automation workflow for notifications.

Amazon Rekognition

The Amazon Rekognition [223] pre-trained deep learning API provides the capabilities of images and video analytics. Its models are trained by Amazon on

billions of public photos from Amazon Prime and optimised for specific use-cases such as face detection, PPE detection, etc. We use this API in our Pyrus pipelines for the detection of safety-equipment among staff working in the factory work-yard. The advantage of using Amazon Rekognition API in our Pyrus pipelines is that it does not require any additional setup and it worked out-of-the-box.

MongoDB

The MongoDB Atlas [205] is a cloud-based NoSQL database service that is used for high-volume data storage of semi-structured or unstructured data. In contrast to the structured records and tables used by relational databases, Atlas stores the data in the form of documents and collections which support the flexibility of different non-structured data types. The Atlas database is also scalable for Big Data storage with support for clusters that can store millions of documents. We use it here to store all the observational and processed data from different sensors, edge devices and compute systems. Atlas is pre-integrated in Tines, which provides no-code SIBs called 'actions' for easy communication with Atlas instances.

10.3 Access Control using Attribute Based Encryption

To facilitate a fine-grained access control in the smart factory to which the work-yard belongs, we utilize a public-key encryption, Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [224] for this particular use-case. The CP-ABE algorithm allows for the identification of the ciphertexts with access structures and the private keys with attributes. Whenever a message is encrypted using CP-ABE, it generates a ciphertext based on the condition that only a single user who is the owner of the specific attributes and satisfies the access structure will be able to produce the private key, and thereby decrypt the message. One of the highlights of CP-ABE is that it permits the definition of top-level policies, and is particularly suitable in scenarios where an individual wants to restrict the access to a specific information only to a subset of users within the same

broadcast domain [225]. Another aspect of CP-ABE is that it is robust by design against collusion attacks [226]. A CP-ABE scheme consists of the following four basic algorithms:

- SETUP(). This algorithm generates the public key pk_c and master key mk_c.
- KEYGEN(mk_c, $Attr_c$). This algorithm takes mk_c and the user attribute list $Attr_c$ as input and returns a private key pv_c of user C.
- $\text{ABE}_{pk_c,w}(m)$. The encryption algorithm takes pk_c, an access policy w over the pool of attributes, and sensor reading m as input. It returns a ciphertext that can only be decrypted by a user that possesses a set of attributes $Attr_c$ such that $Attr_c$ satisfies w.
- $\text{ABD}_{pv_c}(\mathcal{C})$. The decryption algorithm takes pk_c, pv_c of user C and the ciphertext $\mathcal{C}$ as input. It outputs the plaintext m if and only if the user $Attr_c$ satisfies w.

For our industrial use case, we define a secure access policy along the lines of CP-ABE as follows: we consider three attributes in our use case scenario: *Sensor Examiner* (SE), *Video Analyst* (VA) and *Decision Manager* (DM) for three departments in our industrial use-case setting. The main characteristics of the three departments are described as follows:

- We first consider the Technical Fault Monitoring Department (TFMD). The primary task of TFMD is to monitor the readings of the three sensors (temperature, pressure, vibration) and it has access to the data collected by these three sensors. We assign SE attribute to TFMD.
- We then consider the Safety Surveillance Department (SSD). The main responsibility of SSD is to analyse the videos captured by the video camera. Thereby, the SSD has access to the readings of the video camera. We assign VA attribute to SSD.

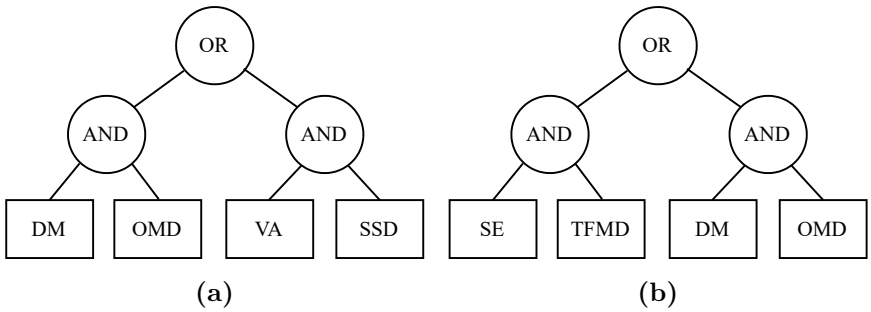


Figure 10.4: Structure of access policies for our scheme: (a) Video Surveillance, (b) Sensor Reading.

- Finally, we consider the Operation Management Department (OMD). OMD takes the final call for the need of generating an alarm if any emergency occurs. Emergencies are reflected through the readings of the corresponding sensors and/or the videos. We assign DM attribute to OMD.

10.3.1 Bilinear Map

Our CP-ABE is based on a bilinear map. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbb{G}$ and e be a bilinear map, $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. The bilinear map e has the following properties:

- Bilinearity: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
- Non-degenerate: $e(g, g) \neq 1$.

Here, $\mathbb{G}$ is a bilinear group if the group operation in $\mathbb{G}$ and the bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ are both efficiently computable. It is worth noting that the map e is symmetric as $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$.

10.3.2 Decision Tree

Let $\mathcal{T}$ be a decision tree representing an access structure. Figure 10.4 illustrates the simple decision trees that are generated from the access policies defined using the specific attributes and their entities. The access policies define which entities

have access to specific data generated by the devices. As mentioned previously, we intend to allow fine-grained secure access control based on attributes, and for this we leverage public-key encryption, i.e., CP-ABE. In our case, the policies define that in a normal context, the Video Surveillance access policy is that either a DM belonging to the OMD department or a VA belonging to the SSD department can have access. Similarly, the Sensor reading policy is that either a SE belonging to the TFMD department or a DM belonging to the OMD department can have access.

As we complete the use case with the access to other elements of the ecosystem, the set of policies will become more complex, as different entities will be added, with typically partially overlapping rights. Only the individuals with the specific attributes will be able to access the data and/or perform operations.

10.3.3 Our Construction

We now provide our main construction of the fine-grain access control approach.

Setup. The setup algorithm chooses a bilinear group $\mathbb{G}$ of prime order p with generator g. It then selects two random exponents $\alpha, \beta \in \mathbb{Z}_p$. The public key is determined as:

$$pk_c = \mathbb{G}, g, h = g^{\beta}, f = g^{1/\beta}, e(g,g)^{\alpha}$$

and the master key mk_c is set as (β, g^{α}).

Keygen(mk_c, $Attr_c$). Our key generation algorithm takes the master key mk_c and a set of attributes $S = \{SE, VA, DM\}$ associated with the department as inputs and provides a private key that identifies with that set. This algorithm initially chooses a random $r \in \mathbb{Z}_p$, and next random $r_j \in \mathbb{Z}_p$ for each attribute $j \in S$. In our algorithm, the private key is generated as follows:

$$pv_c = (D = g^{(\alpha+r)/\beta}, \forall j \in S : D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j}),$$

where H is a hash function, $H : \{0,1\}^* \rightarrow \mathbb{G}$.

Abe$_{pk_c,w}(m)$. Our encryption algorithm encrypts a message m under the decision tree structure $\mathcal{T}$. Beginning with the root node N, our algorithm chooses

a random $\delta \in \mathbb{Z}_p$. Let L be the set of leaf nodes in $\mathcal{T}$. The ciphertext is generated by giving the decision tree $\mathcal{T}$ and determining:

$$\mathcal{C} = (\mathcal{T}, \bar{C} = me(g,g)^{\alpha s}, C = h^s, \forall y \in Y : C_y = g^{q_y(0)}, C_{yp} = H(att(y))^{q_y(0)}),$$

where the function $att(x)$ signifies the attribute associated with the leaf node x in the decision tree $\mathcal{T}$. In the above equation, our algorithm generates random value s to calculate the shared value $q_y(0)$ for each attribute in the decision tree $\mathcal{T}$ using linear secret sharing. In CP-ABE, as private keys are generated randomly using the decision tree $\mathcal{T}$, it thus prevents collusion attack.

$\mathbf{Abd}_{pv_c}(\mathcal{C})$. Our decryption algorithm executes recursively. Here, we present the simplest form of our decryption algorithm. If x is a leaf node, we let $i = att(x)$. If $i \in S$, our algorithm computes message m from the ciphertext $\mathcal{C}$ using pv_c as follows:

$$m = \frac{e(D_i, C_x)}{e(D'_i, C'_x)}.$$

We refer the reader to [224] for further details about the above computations.

10.4 Conclusions and Reflections

In terms of the Digital Thread [13], work on the overall lifecycle ecosystem that supports the smooth interoperation of a physical facility (like a machine, a factory or even a supply chain) and its direct or derived digital components (data and capabilities, but also processes, decisions, security) are still topic of ongoing research and are under-development. The integration is often done ad hoc, and successful platform attempts are domain and layer-specific, like EdgeX for IoT middleware. In this case study we have decided not to integrate everything in DIME, for a number of reasons.

First of all, we wanted to leverage the different integrations that already exist (like the IoT sensors in EdgeX, MongoDB in Tines, AWS in Pyrus) and preexisting communication routes (like EdgeX and MongoDB) and some of the workflows, and see how these islands of integration could be further brought together to a complex scenario. The experience is that there is still a considerable need of adaptation and testing, as for example the networks and protocols (which depend on configurations) and the specific software versions do matter.

We also decided against a full integration in DIME as for example the current DIME data types do not support video streams, therefore a direct integration of camera output would not be feasible at present. In this sense, we play to the individual strengths of the different platforms and integration and abstraction/virtualization approaches.

We wanted to show how a quite complex system of systems can come together in quasi-realistic settings. We also showed that it allows a piece-wise integration using four different platforms, three of whom using models and low code approaches (Tines, Pyrus and DIME). Of those, Pyrus and DIME are Cinco-products [25] and XMDD [138] approaches, whereby Pyrus is a web-based data-flow specialized tool for data analytics, while DIME is a much more complex and general-purpose tool with complex interdependent model types. Taken together, this is a step towards enabling a heterogeneous analysis and verification for distributed systems, as in [122], and the possibly hierarchical organization of reusable portions of logic in terms of features [159].

The inclusion of security is at the moment still simple: it is based on attributes that are roles. In this sense it is de facto similar to Role Based Access Control, but the use of attributes and their connection to the encryption make it more flexible (as one could also consider more attributes that are context-dependent), and more secure.

Acknowledgements

This project received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Smart 4.0 Co-Fund, grant agreement No. 847577; research grants from Science Foundation Ireland (SFI) under Grant Number 16/RC/3918 (CONFIRM Centre), 2094-1 (Lero, the Software Research Centre) and 18/CRT/6223 (CRT-AI). We are also thankful to Dr. Kevin Moerman (National University of Ireland, Galway) for providing us the equipment and support for the StereoPi and cameras setup.

10. PAPER V (AP5)

Part III

Appendixes

Exemplary AD-DSL

SIBs Signatures for R-Programming and Analytics AD-DSL

```
package common
sib read_CSV : info.scce.dime.app.demo.R_Scripts#read_CSV
        file_name: text
        -> success
                file_handler: text

sib generate_summary : info.scce.dime.app.demo.R_Scripts#generate_summary
        file_handler: text
        col_name: text
        -> success
                result: file

sib  moveDoubleColToRserver: info.scce.dime.app.demo.R_Scripts#MoveDoubleColToRserver
        col: [real]
        -> success
                col_name: text

sib mean : info.scce.dime.app.demo.R_Scripts#calculate_mean
        col_name: text
        -> success
                result: real

sib plotRhistogram : info.scce.dime.app.demo.R_Scripts#plot_R_histogram
        file_handler: text
        col_name: text
        breaks: text
        title: text
        x_label: text
        y_label: text
        color: text
        -> success
                result: file
        -> noresult
        -> failure


sib plotRwordcloud : info.scce.dime.app.demo.R_Scripts#plot_R_wordcloud
        file_name: text
        col_name: text
        min_frequency: text
        max_words: text
        -> success
                result: file
        -> noresult
        -> failure


sib executeRcommand : info.scce.dime.app.demo.R_Scripts#executeRcommand
        command: text
        -> success
                result: text
        -> noresult
        -> failure


sib split_data_frame : info.scce.dime.app.demo.R_Scripts#split_data_frame
        file_handler: text
        col1: text
        col2: text
        col3: text
        col4: text
        -> success
                df: text
        -> noresult
```

```
        -> failure

sib plot_dual_linechart : info.scce.dime.app.demo.R_Scripts#plot_dual_linechart
        dataframe: text
        x1: text
        y1: text
        title1: text
        x2: text
        y2: text
        title2: text
        -> success
                result: file
        -> noresult
        -> failure

sib plot_quad_summary_charts : info.scce.dime.app.demo.R_Scripts#plot_quad_summary_charts
        dataframe: text
        x1: text
        y1: text
        -> success
                result: file
        -> noresult
        -> failure
```

Backend Java Implementation for R-Programming and Analytics AD-DSL

```
package info.scce.dime.app.demo;

import java.io.FileInputStream;
import java.io.IOException;
import com.opencsv.CSVReader;
import com.opencsv.exceptions.CsvException;
import java.io.FileReader;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Date;
import java.util.Random;
import org.rosuda.REngine.REXP;
import org.rosuda.REngine.Rserve.RConnection;
import org.rosuda.REngine.Rserve.RFileInputStream;
import org.rosuda.REngine.Rserve.RFileOutputStream;
import org.rosuda.REngine.Rserve.RserveException;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.nio.charset.StandardCharsets;
import javax.enterprise.context.spi.CreationalContext;
import javax.enterprise.inject.spi.Bean;
import javax.enterprise.inject.spi.BeanManager;
import javax.enterprise.inject.spi.CDI;
import de.ls5.dywa.generated.util.DomainFileController;
```

```
import de.ls5.dywa.generated.util.FileReference;

public class R_Scripts {
        /**
         * @return The current system time.
         */
        public static Date getCurrentTime() {
                return new Date();
        }

        public static String createRandomWord(int len) {
                String name = "";
                for (int i = 0; i < len; i++) {
                        int v = 1 + (int) (Math.random() * 26);
                        char c = (char) (v + 'a' - 1);
                        name += c;
                }
                return name;
        }

        public static String split_data_frame(String file_handler, String col1, String col2,
             String col3, String col4) {
                String df = "filtered_dfs";
                try {
                        String command = df + "=dataframe_split('" + file_handler + "','" +
                             col1 + "','" + col2 + "','" + col3
                                        + "','" + col4 + "')";
                        Rcon.getInstance().getConnection().parseAndEval(command);

                } catch (Exception ex) {
                        return ex.toString();
                }
                return df;
        }

        public static String read_CSV(String file_name) {
                String RandomS = createRandomWord(4);
                String server_file = RandomS + ".csv";
                String file_handler = "dataR";
                try {
                        transfer_toserver(Rcon.getInstance().getConnection(), file_name,
                             server_file);
                        Rcon.getInstance().getConnection()
                                        .parseAndEval(file_handler + "<- read.csv('" +
                                             server_file + "', stringsAsFactors = FALSE)");
                } catch (Exception ex) {

                }
                return file_handler;
        }

        public static FileReference plot_quad_summary_charts(String dataframe, String x1,
             String y1) {
                return quad_summary_charts(Rcon.getInstance().getConnection(), dataframe, x1,
                      y1);
        }

        public static FileReference quad_summary_charts(RConnection c, String dataframe,
             String x1, String y1) {
                String RandomS = createRandomWord(4);
                String resultant_file = RandomS + "_plot4.jpg";
                try {
                        c.parseAndEval("jpeg('temp1.jpg')");
                        String command = "plot_quad_summary_charts(" + dataframe + ",'" + x1
                             + "','" + y1 + "')";
                        c.parseAndEval(command);
                        c.parseAndEval("dev.off()");
                } catch (Exception ex) {
                        System.out.println("Summary ended with exception");
                }
```

```
        return transfer_toclient(c, resultant_file, "temp1.jpg");
    }

    public static FileReference plot_dual_linechart(String dataframe, String x1, String
        y1, String title1, String x2,
                String y2, String title2)
    {
        return dual_line_chart(Rcon.getInstance().getConnection(), dataframe, x1, y1,
              title1, x2, y2, title2);
    }

    public static FileReference dual_line_chart(RConnection c, String dataframe, String
        x1, String y1, String t1,
                String x2, String y2, String t2) {
        String RandomS = createRandomWord(4);
        String resultant_file = RandomS + "_plot4.jpg";
        try {
                c.parseAndEval("jpeg('temp1.jpg')");
                String command = "plot_dual_line_chart(" + dataframe + ",'" + x1 + "
                     ','" + y1 + "',1,'" + t1 + "',2,'" + x2
                             + "',3,'" + y2 + "',4,'" + t2 + "')";
                c.parseAndEval(command);
                c.parseAndEval("dev.off()");
        } catch (Exception ex) {
                System.out.println("Summary ended with exception");
        }
        return transfer_toclient(c, resultant_file, "temp1.jpg");
    }

    public static FileReference R_summary(RConnection c, String file_handler, String
        col_name) {
        String RandomS = createRandomWord(4);
        String resultant_file = RandomS + "_plot4.jpg";
        try {
                if (!(file_handler.isEmpty() || file_handler == null)) {
                        col_name = file_handler + "$" + col_name;
                }
                c.parseAndEval("jpeg('temp1.jpg')");
                c.parseAndEval("d<-summary(" + col_name + ")");
                c.parseAndEval("grid.newpage()");
                c.parseAndEval("grid.table(t(d))");
                c.parseAndEval("dev.off()");
        } catch (Exception ex) {
                System.out.println("Summary ended with exception");
        }
        return transfer_toclient(c, resultant_file, "temp1.jpg");
    }

    public static FileReference R_histogram(RConnection c, String file_handler, String
        col_name, String breaks,
                String title, String x_label, String y_label, String color) {
        String RandomS = createRandomWord(4);
        String resultant_file = RandomS + "_plot4.jpg";
        try {
                if (!(file_handler.isEmpty() || file_handler == null)) {
                        col_name = file_handler + "$" + col_name;
                }
                c.parseAndEval("jpeg('temp.jpg')");
                c.parseAndEval("hist(" + col_name + ",breaks=" + breaks + ",main='"
                    + title + "',xlab='" + x_label
                            + "',ylab='" + y_label + "',col='" + color + "')");
                c.parseAndEval("dev.off()");

        } catch (Exception ex) {
                System.out.println("histogram end with exception");
        }
        return transfer_toclient(c, resultant_file, "temp.jpg");
    }
```

```
public static FileReference R_wordcloud(RConnection c, String fileName, String
    col_name, String min_frequency,
                String max_words) {
        String RandomS = createRandomWord(4);
        String resultant_file = RandomS + "_plot.jpg";
        try {

                col_name = "dataD$" + col_name;
                String server_file = RandomS + ".csv";
                transfer_toserver(c, fileName, server_file);

                c.parseAndEval("dataD <- read.csv('" + server_file + "',
                    stringsAsFactors = FALSE)");
                c.parseAndEval("text <-" + col_name);
                c.parseAndEval("jpeg('temp.jpg')");
                c.parseAndEval(
                                "docs <- Corpus(VectorSource(text)); dtm <-
                                    TermDocumentMatrix(docs); matrix <- as.matrix(
                                    dtm); words <- sort(rowSums(matrix),decreasing=
                                    TRUE); df <- data.frame(word = names(words),freq
                                    =words)");
                c.parseAndEval("wordcloud(words = df$word, freq = df$freq, min.freq =
                    " + min_frequency + ",max.words="
                                + max_words + ", random.order=FALSE, rot.per=0.35,
                                    colors=brewer.pal(8, 'Dark2'))");
                c.parseAndEval("dev.off()");
        } catch (Exception ex) {
                System.out.println("word cloud end with exception");
        }
        return transfer_toclient(c, resultant_file, "temp.jpg");
}

public static FileReference plot_R_wordcloud(String fileName, String col_name, String
     min_frequency,
                String max_words) {
        return R_wordcloud(Rcon.getInstance().getConnection(), fileName, col_name,
            min_frequency, max_words);

}

public static FileReference plot_R_histogram(String file_handler, String col_name,
    String breaks, String title,
                String x_label, String y_label, String color) // throws IOException,
                    CsvException
{
        return R_histogram(Rcon.getInstance().getConnection(), file_handler, col_name
            , breaks, title, x_label, y_label,
                        color);
}

public static FileReference generate_summary(String file_handler, String col_name) //
     throws IOException,
{
        return R_summary(Rcon.getInstance().getConnection(), file_handler, col_name);
}

public static String executeRcommand(String command) {
        try {
                REXP x = Rcon.getInstance().getConnection().eval(command);// eval("R.
                    version.string");
                return x.asString().toString();
        } catch (Exception ex) {
                return "Error in execution";
        }
}

public static double mean(RConnection c, String col_name) {

        try {
                REXP x1 = c.eval("mean(" + col_name + ")");
```

```
                return x1.asDouble();
        } catch (Exception xx) {
                System.out.println("inside mean function exception" + xx.toString());
                return 0.0;
        }
}

public static double calculate_mean(String col_name) {
        return mean(Rcon.getInstance().getConnection(), col_name);
}

public static String MoveDoubleColToRserver(List<Double> list) {
        return DoubleColToRSERVER(Rcon.getInstance().getConnection(), list);
}

public static String DoubleColToRSERVER(RConnection c, List<Double> list) {
        String RandomS = createRandomWord(4);
        double[] myvalues = new double[list.size()];
        int i = 0;
        for (Double d : list) {
                myvalues[i] = (d);
                i++;
        }
        try {
                c.assign(RandomS, myvalues);
                return RandomS;
        } catch (Exception ex) {
                return "Error in execution";
        }
}

public static void transfer_toserver(RConnection r, String client_file, String
    server_file) {
        byte[] b = new byte[8192000];
        try {
                ClassLoader classloader = Thread.currentThread().
                    getContextClassLoader();
                BufferedInputStream client_stream = new BufferedInputStream(
                    classloader.getResourceAsStream(client_file));
                RFileOutputStream server_stream = r.createFile(server_file);
                int c = client_stream.read(b);
                while (c >= 0) {
                        server_stream.write(b, 0, c);
                        c = client_stream.read(b);
                }
                server_stream.close();
                client_stream.close();
        } catch (IOException e) {
                e.printStackTrace();
        }
}

public static FileReference transfer_toclient(RConnection r, String client_file,
    String server_file) {
        byte[] b = new byte[8192000];
        try {
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                BufferedOutputStream client_stream = new BufferedOutputStream(new
                    FileOutputStream(new File(client_file)));
                RFileInputStream server_stream = r.openFile(server_file);
                int c = server_stream.read(b);
                while (c >= 0) {
                        client_stream.write(b, 0, c);
                        baos.write(b, 0, c);
                        c = server_stream.read(b);
                }
                client_stream.close();
                server_stream.close();
                baos.close();
                InputStream stream = new ByteArrayInputStream(baos.toByteArray());
```

```
                    return getDomainFileController().storeFile(client_file, stream);
                } catch (Exception emn) {
                }
                return exportToFile("Error");
        }

        private static DomainFileController getDomainFileController() {
                final BeanManager bm = CDI.current().getBeanManager();
                final Bean<DomainFileController> bean = (Bean<DomainFileController>) bm
                                .resolve(bm.getBeans(DomainFileController.class));
                final CreationalContext<DomainFileController> cctx = bm.
                    createCreationalContext(bean);
                final DomainFileController fileController = (DomainFileController) bm.
                    getReference(bean, bean.getBeanClass(),
                                cctx);
                return fileController;
        }

        public static FileReference exportToFile(String xx) {
                InputStream stream = new ByteArrayInputStream(exportToText().getBytes(
                    StandardCharsets.UTF_8));
                return getDomainFileController().storeFile(xx + "entry.csv", stream);
        }

        public static String exportToText() {
                StringBuffer buffer = new StringBuffer();
                buffer.append("There is some error, somewhere");
                return buffer.toString();
        }
}
```

Model-Driven Digital Thread Platform for Smart Manufacturing

AHMAD CHAUDHARY, PROF. TIZIANA MARGARIA

CSIS, UNIVERSITY OF LIMERICK

Confirm Smart Manufacturing

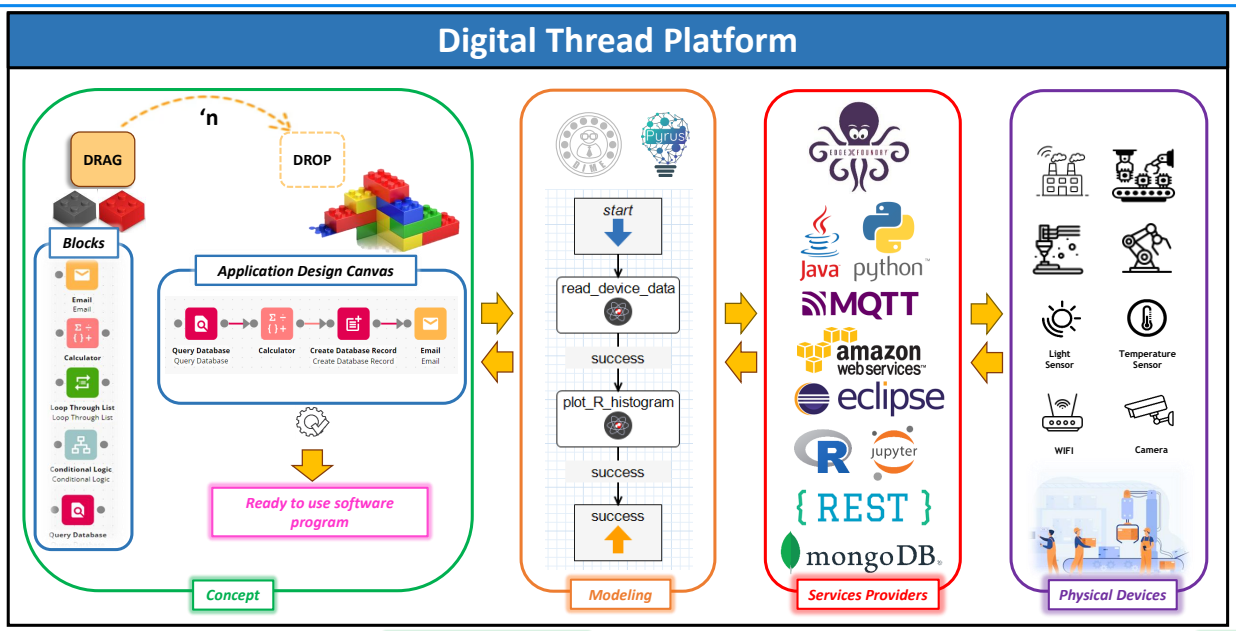
A World Leading SFI Research Centre

Science Foundation Ireland sfi For what's next

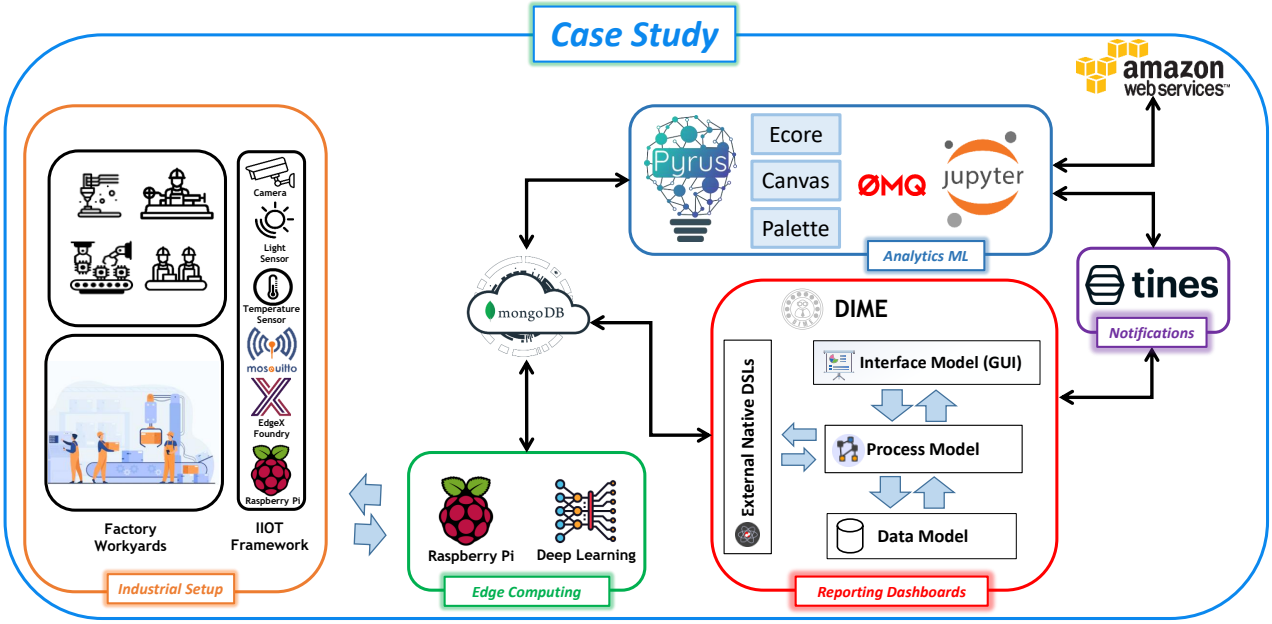
Abstract

In the industry 4.0 ecosystem, a Digital Thread connects the data and processes for smarter manufacturing. It provides an end to end integration of the various digital entities thus fostering interoperability, with the aim to design and deliver complex and heterogeneous interconnected systems. We develop a service oriented domain specific Digital Thread platform in a Smart Manufacturing research and prototyping context. We address the principles, architecture and individual aspects of a growing Digital Thread platform. It conforms to the best practices of coordination languages, integration and interoperability of external services from various platforms, and provides orchestration in a formal methods based, low-code and graphical model driven fashion. We chose the Cinco products DIME and Pyrus as the underlying IT platforms for our Digital Thread solution to serve the needs of the smart manufacturing operations. The small case studies constitute blueprints for the methodology, addressing the knowledge, terminology and concerns of domain stakeholders. Over time, we expect reuse to increase, reducing the new integration and development effort to a progressively smaller portion of the models and code needed for at least the most standard applications.

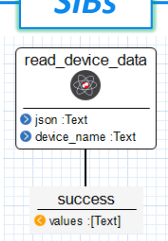
Digital Thread Platform



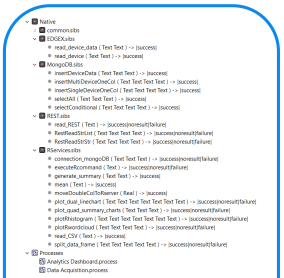
Case Study



SIBs



Palette



References

- Margaria, T., Chaudhary, H.A.A., Guevara, I., Ryan, S., Schieweck, A.: The interoperability challenge: building a model-driven digital thread platform for cps. In: International Symposium on Leveraging Applications of Formal Methods. LNCS, vol. 13036, pp. 393–413 (2021). Springer
- Chaudhary, H. A. A., Guevara, I., John, J., Singh, A., Margaria, T., & Pesch, D. (2022, October). Low-code internet of things application development for edge analytics. In Internet of Things. IoT through a Multi-disciplinary Perspective: 5th IFIP International Cross-Domain Conference, IFIPIoT 2022, Amsterdam, The Netherlands, October 27–28, 2022, Proceedings (pp. 293-312). Cham: Springer International Publishing.

HOST INSTITUTION

UNIVERSITY OF LIMERICK OLLSCOIL LUIMNIGH

PARTNER INSTITUTIONS

Tyndall UCC Maynooth University NUI Galway OE Gaillimh MTU AIT LIT

SCCE

Project Funding

This PhD is funded by the Science Foundation Ireland grant 16/RC/3918 for CONFIRM. CONFIRM is the world leading SFI Research Centre for Smart Manufacturing, The Centre is funded under the SFI Research Centres programme, which has established a network of SFI Research Centres focusing on key research areas in Ireland. The link to the centre website is https://confirm.ie/.

Bibliography

[1] S. Boßelmann, "Evolution of ecosystems for language-driven engineering," PhD thesis, Technical University Dortmund, Germany, 2023, available at https://eldorado.tu-dortmund.de/handle/2003/41375. xv, 11, 44, 148

[2] S. Boßelmann, M. Frohme, D. Kopetzki, M. Lybecait, S. Naujokat, J. Neubauer, D. Wirkner, P. Zweihoff, and B. Steffen, "Dime: A programming-less modeling environment for web applications," in *ISoLA 2016*, ser. LNCS 9953. Cham: Springer International Publishing, 2016, pp. 809–832. xv, 11, 22, 23, 29, 30, 41, 78, 79, 80, 97, 121, 124, 145, 148, 176

[3] P. Zweihoff and B. Steffen, "Pyrus: an online modeling environment for no-code data-analytics service composition," in *ISoLA 2021*, ser. LNCS, vol. 13036. Springer, 2021, pp. 18–40. xv, 11, 21, 22, 23, 29, 33, 35, 41, 107, 121, 124, 145, 149, 177

[4] EdgeX Foundry, "The preferred edge IoT plug and play ecosystem - open source software platform," Accessed, March 2022. [Online]. Available: https://www.edgexfoundry.org/ xv, 35, 37, 101, 129, 152, 170, 176

[5] H. A. A. Chaudhary, I. Guevara, J. John, A. Singh, T. Margaria, and D. Pesch, "Low-code internet of things application development for edge analytics," in *Internet of Things. IoT through a Multi-disciplinary Perspective.* Cham: Springer International Publishing, 2022, pp. 293–312. xvii, 12, 49, 50, 55, 66, 153, 155, 158, 159, 160, 176

BIBLIOGRAPHY

[6] R. Pozzi, T. Rossi, and R. Secchi, "Industry 4.0 technologies: Critical success factors for implementation and improvements in manufacturing companies," *Production Planning & Control*, vol. 34, no. 2, pp. 139–158, 2023. 5

[7] D. Y. Pimenov, M. Mia, M. K. Gupta, Á. R. Machado, G. Pintaude, D. R. Unune, N. Khanna, A. M. Khan, Í. Tomaz, S. Wojciechowski *et al.*, "Resource saving by optimization and machining environments for sustainable manufacturing: A review and future prospects," *Renewable and Sustainable Energy Reviews*, vol. 166, p. 112660, 2022. 5

[8] A. M. Madni, C. C. Madni, and S. D. Lucero, "Leveraging digital twin technology in model-based systems engineering," *Systems*, vol. 7, no. 1, p. 7, 2019. 5

[9] A. Napoleone, M. Macchi, and A. Pozzetti, "A review on the characteristics of cyber-physical systems for the future smart factories," *Journal of manufacturing systems*, vol. 54, pp. 305–335, 2020. 5, 93

[10] A. P. Allian, F. Schnicke, P. O. Antonino, D. Rombach, and E. Y. Nakagawa, "Architecture drivers for trustworthy interoperability in industry 4.0," *IEEE Systems Journal*, 2021. 5, 93

[11] T. Margaria, H. A. A. Chaudhary, I. Guevara, S. Ryan, and A. Schieweck, "The interoperability challenge: Building a model-driven digital thread platform for cps," in *ISoLA 2021, Leveraging Applications of Formal Methods, Verification and Validation*. Cham: Springer International Publishing, 2021, pp. 393–413. 5, 12, 55, 126, 129, 143

[12] K. Cao, S. Hu, Y. Shi, A. W. Colombo, S. Karnouskos, and X. Li, "A survey on edge and edge-cloud computing assisted cyber-physical systems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, pp. 7806–7819, 2021. 6, 18

[13] T. Margaria and A. Schieweck, "The digital thread in industry 4.0," in *International Conference on Integrated Formal Methods*. Springer, 2019, pp. 3–24. 6, 17, 21, 41, 48, 80, 93, 102, 104, 119, 124, 182

[14] J. Nilsson and F. Sandin, "Semantic interoperability in industry 4.0: Survey of recent developments and outlook," in *2018 IEEE 16th international conference on industrial informatics (INDIN)*. IEEE, 2018, pp. 127–132. 6, 95

[15] M. Kovatsch, Y. N. Hassan, and S. Mayer, "Practical semantics for the internet of things: Physical states, device mashups, and open questions," in *2015 5th International Conference on the Internet of Things (IoT)*. IEEE, 2015, pp. 54–61. 6, 95

[16] H. Lee, K. Ryu, and Y. Cho, "A framework of a smart injection molding system based on real-time data," *Procedia Manufacturing*, vol. 11, pp. 1004–1011, 2017. 7, 95

[17] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM computing surveys (CSUR)*, vol. 37, no. 4, pp. 316–344, 2005. 7, 120

[18] B. Steffen, F. Gossen, S. Naujokat, and T. Margaria, "Language-driven engineering: from general-purpose to purpose-specific languages," in *Computing and Software Science*. Springer, 2019, pp. 311–344. 10, 107, 124, 125

[19] F. Gossen, T. Margaria, A. Murtovi, S. Naujokat, and B. Steffen, "Dsls for decision services: a tutorial introduction to language-driven engineering," in *Leveraging Applications of Formal Methods, Verification and Validation. Modeling: 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part I 8*. Springer, 2018, pp. 546–564. 10

[20] M. Voelter, "Language and ide modularization and composition with mps," *Generative and Transformational Techniques in Software Engineering IV: International Summer School, GTTSE 2011, Braga, Portugal, July 3-9, 2011. Revised Papers*, pp. 383–430, 2013. 10

BIBLIOGRAPHY

[21] B. Steffen, T. Margaria, R. Nagel, S. Jörges, and C. Kubczak, "Model-driven development with the jabc," in *Hardware and Software, Verification and Testing: Second International Haifa Verification Conference, HVC 2006, Haifa, Israel, October 23-26, 2006. Revised Selected Papers 2*. Springer, 2007, pp. 92–108. 10, 22, 26, 27, 29, 79, 110, 121, 123

[22] S. J. Mellor, T. Clark, and T. Futagami, "Model-driven development: guest editors' introduction. ieee software, 20 (5). pp. 14-18. issn 0740-7459," *IEEE software*, vol. 20, no. 5, pp. 14–18, 2003. 10, 22, 77, 95, 123, 171

[23] R. Sanchis, Ó. García-Perales, F. Fraile, and R. Poler, "Low-code as enabler of digital transformation in manufacturing industry," *Applied Sciences*, vol. 10, no. 1, p. 12, 2019. 10, 77, 95, 120, 124, 171

[24] K. Ordoñez, J. Hilera, and S. Cueva, "Model-driven development of accessible software: a systematic literature review," *Universal Access in the Information Society*, pp. 1–30, 2020. 11, 79, 107

[25] S. Naujokat, M. Lybecait, D. Kopetzki, and B. Steffen, "Cinco: a simplicity-driven approach to full generation of domain-specific graphical modeling tools," *International Journal on Software Tools for Technology Transfer*, vol. 20, pp. 1–28, 06 2018. 11, 21, 28, 80, 81, 98, 122, 123, 124, 147, 183

[26] P. Zweihoff, S. Naujokat, and B. Steffen, "Pyro: Generating domain-specific collaborative online modeling environments," in *Fundamental Approaches to Software Engineering*, ser. LNCS 11424. Cham: Springer International Publishing, 2019, pp. 101–115. 11, 21, 28, 78, 80, 107, 124

[27] B. Selic, "The pragmatics of model-driven development," *IEEE software*, vol. 20, no. 5, pp. 19–25, 2003. 11, 120

[28] H. A. A. Chaudhary and T. Margaria, "Integration of micro-services as components in modeling environments for low code development," *Proceedings of the Institute for System Programming of the RAS*, vol. 33, no. 4, 2021. 12, 48, 55, 102, 103, 111, 126, 129, 130

[29] H. A. A. Chaudhary and T. Margaria, "Integrating external services in dime," in *ISoLA 2021, Leveraging Applications of Formal Methods, Verification and Validation*. Cham: Springer International Publishing, 2021, pp. 41–54. 12, 44, 46, 55, 67, 129, 131

[30] H. A. A. Chaudhary and T. Margaria, "Dsl-based interoperability and integration in the smart manufacturing digital thread," *Electronic Communications of the EASST*, vol. 81, 2022. 12, 55, 64, 162, 177

[31] I. Guevara, H. A. A. Chaudhary, and T. Margaria, "Model-driven edge analytics: Practical use cases in smart manufacturing," in *ISoLA 2022, Leveraging Applications of Formal Methods, Verification and Validation. Practice*. Cham: Springer Nature Switzerland, 2022, pp. 406–421. 12, 55, 171

[32] H. A. A. Chaudhary, I. Guevara, J. John, A. Singh, A. Ghosal, D. Pesch, and T. Margaria, "Model-driven engineering in digital thread platforms: A practical use case and future challenges," in *ISoLA 2022, Leveraging Applications of Formal Methods, Verification and Validation. Practice*. Cham: Springer Nature Switzerland, 2022, pp. 195–207. 12, 55

[33] H. A. A. Chaudhary, I. Guevara, A. Singh, A. Schieweck, J. John, T. Margaria, and D. Pesch, "Efficient model-driven prototyping for edge analytics," *Electronics*, vol. 12, no. 18, 2023. [Online]. Available: https://www.mdpi.com/2079-9292/12/18/3881 12, 55, 64

[34] A. Jamwal, R. Agrawal, M. Sharma, and A. Giallanza, "Industry 4.0 technologies for manufacturing sustainability: a systematic review and future research directions," *Applied Sciences*, vol. 11, no. 12, p. 5725, 2021. 15, 22, 123

[35] F. Lelli, "Interoperability of the time of industry 4.0 and the internet of things," *Future Internet*, vol. 11, no. 2, p. 36, 2019. 15, 93

[36] L. D. Xu, "The contribution of systems science to industry 4.0," *Systems Research and Behavioral Science*, vol. 37, no. 4, pp. 618–631, 2020. 15, 93

BIBLIOGRAPHY

[37] N. Carvalho, O. Chaim, E. Cazarini, and M. Gerolamo, "Manufacturing in the fourth industrial revolution: A positive prospect in sustainable manufacturing," *Procedia Manufacturing*, vol. 21, pp. 671–678, 2018. 15

[38] E. A. Lee, "Cyber-physical systems-are computing foundations adequate," in *Position paper for NSF workshop on cyber-physical systems: research motivation, techniques and roadmap*, vol. 2. Austin, TX, 2006, pp. 1–9. 15

[39] D. Gürdür, "Making interoperability visible: A novel approach to understand interoperability in cyber-physical systems toolchains," Ph.D. dissertation, KTH Royal Institute of Technology, 2016. 15

[40] C. Klötzer, J. Weißenborn, and A. Pflaum, "The evolution of cyber-physical systems as a driving force behind digital transformation," in *2017 IEEE 19th Conference on Business Informatics (CBI)*, vol. 02, 2017, pp. 5–14. 15

[41] B. Wang, P. Zheng, Y. Yin, A. Shih, and L. Wang, "Toward human-centric smart manufacturing: A human-cyber-physical systems (hcps) perspective," *Journal of Manufacturing Systems*, vol. 63, pp. 471–490, 2022. 15

[42] A. BAGGIA, B. WERBER, and A. ŽNIDARŠIČ, "Human in society 5.0: Changes in the attitude towards microchip implants," *Znanstveno-raziskovalni izzivi na poti digitalne preobrazbe*, p. 124, 2022. 15

[43] X. Xu, Y. Lu, B. Vogel-Heuser, and L. Wang, "Industry 4.0 and industry 5.0—inception, conception and perception," *Journal of Manufacturing Systems*, vol. 61, pp. 530–535, 2021. 15

[44] L. Calderone, "Ai could beat humans at everything by 2030— roboticstomorrow," 2018. 16

[45] K. Schwab and N. Davis, *Shaping the future of the fourth industrial revolution*. Currency, 2018. 16

[46] D. G. Broo, U. Boman, and M. Törngren, "Cyber-physical systems research and education in 2030: Scenarios and strategies," *Journal of Industrial Information Integration*, vol. 21, p. 100192, 2021. 16, 17

[47] J. A. Stankovic, J. W. Sturges, and J. Eisenberg, "A 21st century cyber-physical systems education," *Computer*, vol. 50, no. 12, pp. 82–85, 2017. 17

[48] J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu, "Deep learning for smart manufacturing: Methods and applications," *Journal of manufacturing systems*, vol. 48, pp. 144–156, 2018. 17, 18

[49] F. Tao, Q. Qi, A. Liu, and A. Kusiak, "Data-driven smart manufacturing," *Journal of Manufacturing Systems*, vol. 48, pp. 157–169, 2018. 17

[50] J. Wang, C. Xu, J. Zhang, and R. Zhong, "Big data analytics for intelligent manufacturing systems: A review," *Journal of Manufacturing Systems*, vol. 62, pp. 738–752, 2022. 18

[51] M. Wocker, N. K. Betz, C. Feuersänger, A. Lindworsky, and J. Deuse, "Unsupervised learning for opportunistic maintenance optimization in flexible manufacturing systems," *Procedia CIRP*, vol. 93, pp. 1025–1030, 2020. 18

[52] R. Zhao, D. Wang, R. Yan, K. Mao, F. Shen, and J. Wang, "Machine health monitoring using local feature-based gated recurrent unit networks," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 2, pp. 1539–1548, 2017. 18

[53] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016. 18, 170

[54] K. Cao, L. Li, Y. Cui, T. Wei, and S. Hu, "Exploring placement of heterogeneous edge servers for response time minimization in mobile edge-cloud computing," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 1, pp. 494–503, 2020. 18, 19

[55] N. Harth, C. Anagnostopoulos, and D. Pezaros, "Predictive intelligence to the edge: impact on edge analytics," *Evolving Systems*, vol. 9, pp. 95–118, 2018. 19

[56] J. Zhou, H.-N. Dai, and H. Wang, "Lightweight convolution neural networks for mobile edge computing in transportation cyber physical systems," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 6, pp. 1–20, 2019. 19

[57] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge analytics in the internet of things," *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, 2015. 19

[58] K. Thramboulidis, "Model integrated mechatronics: An architecture for the model driven development of manufacturing systems," in *Proceedings of the IEEE International Conference on Mechatronics, 2004. ICM'04.* IEEE, 2004, pp. 497–502. 19, 124

[59] L. S. Dalenogare, G. B. Benitez, N. F. Ayala, and A. G. Frank, "The expected contribution of industry 4.0 technologies for industrial performance," *International Journal of production economics*, vol. 204, pp. 383–394, 2018. 20, 119

[60] A. Geraci, *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries.* IEEE Press, 1991. 20, 94

[61] T. Y. Pang, J. D. Pelaez Restrepo, C.-T. Cheng, A. Yasin, H. Lim, and M. Miletic, "Developing a digital twin and digital thread framework for an 'industry 4.0'shipyard," *Applied Sciences*, vol. 11, no. 3, p. 1097, 2021. 20, 96

[62] V. Singh and K. E. Willcox, "Engineering design with digital thread," *AIAA Journal*, vol. 56, no. 11, pp. 4515–4528, 2018. 20, 119

[63] L. S. Gould, "What are digital twins and digital threads?" *Automotive Design & Production*, vol. 23, 2018. 20, 119

[64] M. Broy, K. Havelund, and R. Kumar, "Towards a unified view of modeling and programming," in *ISoLA 2016.* Springer, 2016, pp. 238–257. 21, 122

[65] S. Naujokat, J. Neubauer, T. Margaria, and B. Steffen, "Meta-level reuse for mastering domain specialization," in *ISoLA 2016*. Springer, 2016, pp. 218–237. 21, 122

[66] A. Bainczyk, D. Busch, M. Krumrey, D. S. Mitwalli, J. Schürmann, J. Tagoukeng Dongmo, and B. Steffen, "Cinco cloud: a holistic approach for web-based language-driven engineering," in *Leveraging Applications of Formal Methods, Verification and Validation. Software Engineering: 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22–30, 2022, Proceedings, Part II*. Springer, 2022, pp. 407–425. 21

[67] S. Tahmasebi, A. Layegh, N. Nikolov, A. H. Payberah, K. Dinh, V. Mitrovic, D. Roman, and M. Matskin, "Dataclouddsl: Textual and visual presentation of big data pipelines," in *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2022, pp. 1165–1171. 21, 22

[68] A. S. Maiya, "ktrain: A low-code library for augmented machine learning," *arXiv preprint arXiv:2004.10703*, 2020. 21, 79, 109

[69] R. B. Pereira, J. C. Ramalho, and M. A. Brito, "Development of self-diagnosis tests system using a dsl for creating new test suites for integration in a cyber-physical system," *Schloss Dagstuhl – Leibniz-Zentrum für Informatik*, vol. 94, pp. 19:1–19:16, 2021. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/14436 21

[70] N. AJ, "A survey on domain-specific modeling and languages in robotics," *JOURNAL OF SOFTWARE ENGINEERING FOR ROBOTICS*, vol. 7, pp. 75–99, 2016. 21

[71] G. S. Nandi, D. Pereira, J. Proença, and E. Tovar, "Work-in-progress: a dsl for the safe deployment of runtime monitors in cyber-physical systems," in *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2020, pp. 395–398. 21

BIBLIOGRAPHY

[72] A. Barisic, V. Amaral, M. Goulao, and A. Aguiar, "Introducing usability concerns early in the dsl development cycle: Flowsl experience report," in *Proceedings of the 1st International Workshop on MD2P2 co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages & Systems*, 2014. 21, 123

[73] T. Kosar, P. E. Martı, P. A. Barrientos, M. Mernik *et al.*, "A preliminary study on various implementation approaches of domain-specific language," *Information and software technology*, vol. 50, no. 5, pp. 390–405, 2008. 21, 123

[74] A.-L. Lamprecht, T. Margaria, and B. Steffen, "Data-flow analysis as model checking within the jabc," in *International Conference on Compiler Construction*. Springer, 2006, pp. 101–104. 22, 123

[75] T. Margaria, C. Kubczak, M. Njoku, and B. Steffen, "Model-based design of distributed collaborative bioinformatics processes in the jabc," in *11th IEEE ICECCS*, 2006, pp. 8 pp.–. 22, 108, 123

[76] A.-L. Lamprecht, T. Margaria, and B. Steffen, "Bio-jeti: a framework for semantics-based service composition," *BMC bioinformatics*, vol. 10, no. 10, pp. 1–19, 2009. 22

[77] T. Margaria, C. Kubczak, B. Steffen, and S. Naujokat, "The fmics-jeti platform: Status and perspectives," in *ISoLA 2006)*. IEEE, 2006, pp. 402–407. 22, 123

[78] S. Gnesi and T. Margaria, *Formal methods for industrial critical systems: A survey of applications*. John Wiley & Sons, 2012. 22, 123

[79] C. Kubczak, T. Margaria, and B. Steffen, "Mashup development for everybody: a planning-based approach," in *Workshop on Service Matchmaking & Resource Retrieval in the Semantic Web (CEUR-WS)*, 2009. 22, 123

[80] T. Margaria, D. Meyer, C. Kubczak, M. Isberner, and B. Steffen, "Synthesizing semantic web service compositions with jmosel and golog," in *ISWC 2009*. Springer, 2009, pp. 392–407. 22, 108, 123

[81] F. T. Balagtas-Fernandez and H. Hussmann, "Model-driven development of mobile applications," in *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2008, pp. 509–512. 22, 123

[82] H. Heitkötter, T. A. Majchrzak, and H. Kuchen, "Cross-platform model-driven development of mobile applications with md2," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013, pp. 526–533. 22, 123

[83] J. A. Barriga, P. J. Clemente, E. Sosa-Sánchez, and Á. E. Prieto, "Simulateiot: Domain specific language to design, code generation and execute iot simulation environments," *IEEE Access*, vol. 9, pp. 92 531–92 552, 2021. 22, 147

[84] J. Lee, H.-A. Kao, and S. Yang, "Service innovation and smart analytics for industry 4.0 and big data environment," *Procedia cirp*, vol. 16, pp. 3–8, 2014. 22

[85] K. Thramboulidis, D. Perdikis, and S. Kantas, "Model driven development of distributed control applications," *IJAMT 2007*, vol. 33, no. 3, pp. 233–242, 2007. 23, 123

[86] G. T. Heineman and W. T. Councill, "Component-based software engineering," *Putting the pieces together, addison-westley*, vol. 5, p. 1, 2001. 24, 147

[87] I. Crnkovic, S. Sentilles, A. Vulgarakis, and M. R. Chaudron, "A classification framework for software component models," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 593–615, 2010. 24, 147

[88] T. Bureš, J. Carlson, S. Sentilles, and A. Vulgarakis, "A component model family for vehicular embedded systems," in *2008 The Third International Conference on Software Engineering Advances*. IEEE, 2008, pp. 437–444. 24

[89] S. Sentilles, A. Vulgarakis, T. Bureš, J. Carlson, and I. Crnković, "A component model for control-intensive distributed embedded systems," in *International Symposium on Component-Based Software Engineering*. Springer, 2008, pp. 310–317. 25, 147

[90] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," *Present and ulterior software engineering*, pp. 195–216, 2017. 25

[91] M. Matskin and J. Rao, "Value-added web services composition using automatic program synthesis," in *International Workshop on Web Services, E-Business, and the Semantic Web*. Springer, 2002, pp. 213–224. 26

[92] M. Dalla Preda, M. Gabbrielli, C. Guidi, J. Mauro, and F. Montesi, "Service integration via target-transparent mediation," in *2012 Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2012, pp. 1–5. 26

[93] M. Gabbrielli, S. Giallorenzo, C. Guidi, J. Mauro, and F. Montesi, "Self-reconfiguring microservices," in *Theory and Practice of Formal Methods: Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday*. Springer, 2016, pp. 194–210. 26, 27

[94] T. Margaria, B. Steffen, and M. Reitenspieß, "Service-Oriented Design: The jABC Approach," in *Service Oriented Computing (SOC)*, ser. Dagstuhl Seminar Proceedings, F. Cubera, B. J. Krämer, and M. P. Papazoglou, Eds., no. 05462. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2006/521 26

[95] G. Jung, T. Margaria, R. Nagel, W. Schubert, B. Steffen, and H. Voigt, "SCA and jABC: Bringing a Service-Oriented Paradigm to Web-Service Construction," in *Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008, Porto Sani,*

Greece, October 13-15, 2008. Proceedings, ser. Communications in Computer and Information Science, T. Margaria and B. Steffen, Eds., vol. 17. Springer, 2008, pp. 139–154. 27

[96] B. Margolis and J. Sharpe, "Soa for the business developer: Concepts," *BPEL, and SCA: Concepts, BPEL and SCA: MC Press, Lewisville*, 2007. 27

[97] "Xtext - integration with emf," Accessed July, 2022. [Online]. Available: https://eclipse.dev/Xtext/documentation/308_emf_integration.html 29

[98] T. Margaria and B. Steffen, "Business process modeling in the jabc: the one-thing approach," in *Handbook of research on business process modeling*. IGI Global, 2009, pp. 1–26. 29, 82, 98, 124, 125, 148

[99] T. Margaria and B. Steffen, "Extreme model-driven development (xmdd) technologies as a hands-on approach to software development without coding," *Encyclopedia of Education and Information Technologies*, pp. 732–750, 2020. 29, 79, 98, 110, 125, 148

[100] B. Steffen, T. Margaria, A. Claßen, and V. Braun, "The metaframe'95 environment," in *CAV*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 450–453. 29, 79, 122

[101] J. Neubauer, M. Frohme, B. Steffen, and T. Margaria, "Prototype-driven development of web applications with dywa," in *ISoLA 2014*, ser. LNCS 8802. Springer, 2014, pp. 56–72. 29, 83, 110, 122

[102] F. Gossen, A. Murtovi, P. Zweihoff, and B. Steffen, "Add-lib: Decision diagrams in practice," *arXiv preprint arXiv:1912.11308*, 2019. 29

[103] T. Tegeler, S. Teumert, J. Schürmann, A. Bainczyk, D. Busch, and B. Steffen, "An introduction to graphical modeling of ci/cd workflows with rig," in *Leveraging Applications of Formal Methods, Verification and Validation: 10th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2021, Rhodes, Greece, October 17–29, 2021, Proceedings 10*. Springer, 2021, pp. 3–17. 29, 164

[104] M. Merten and B. Steffen, "Simplicity driven application development," *Journal of Integrated Design and Process Science*, vol. 17, no. 3, pp. 9–23, 2013. 33, 99

[105] T. Margaria and B. Steffen, "Simplicity as a driver for agile innovation," *Computer*, vol. 43, no. 6, pp. 90–92, 2010. 33, 99

[106] S. Windmüller, J. Neubauer, B. Steffen, F. Howar, and O. Bauer, "Active continuous quality control," in *Proceedings of the 16th International ACM Sigsoft symposium on Component-based software engineering*, 2013, pp. 111–120. 33, 86, 99

[107] "Tines — no-code automation for security teams," Accessed May, 2022. [Online]. Available: https://www.tines.com/lessons/storyboard/ 38, 146, 177

[108] Docker Inc., "Docker," May 2022. [Online]. Available: https://www.docker.com/ 39, 152

[109] Erbessd instruments. (Accessed May, 2022) Condition monitoring & industrial automation. [Online]. Available: https://www.erbessd-instruments.com/wireless-vibration-sensors/ 40, 176

[110] "Raspberry Pi High Quality Camera," Accessed May, 2022. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-high-quality-camera/ 40, 176

[111] J. John, A. Ghosal, T. Margaria, and D. Pesch, "Dsls for model driven development of secure interoperable automation systems with edgex foundry," in *2021 Forum on Specification & Design Languages (FDL)*. IEEE, 2021, pp. 1–8. 48, 102, 152

[112] J. John, A. Ghosal, T. Margaria, and D. Pesch, "Dsls and middleware platforms in a model-driven development approach for secure predictive maintenance systems in smart factories," in *Leveraging Applications of Formal Methods, Verification and Validation: 10th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2021, Rhodes, Greece,*

October 17–29, 2021, Proceedings 10. Springer, 2021, pp. 146–161. 48, 102, 152

[113] A. R. Hevner, "A three cycle view of design science research," *Scandinavian journal of information systems*, vol. 19, no. 2, p. 4, 2007. 51, 126

[114] S. Ryan and T. Margaria, "Digitalisation for organisations in industry 4.0: A working example," in *ITM Web of Conferences*, vol. 51. EDP Sciences, 2023. 70

[115] S. Ryan and B. Steffen, "Towards multi-perspective consulting in times of disruption," in *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2022, pp. 1129–1134. 70

[116] I. H. Guevara, "Synthesis of smart manufacturing environments. towards evolvable robotic navigation scenarios." *Electronic Communications of the EASST*, vol. 81, 2022. 70

[117] R. Waszkowski, "Low-code platform for automating business processes in manufacturing," *IFAC-PapersOnLine*, vol. 52(10), pp. 376–381, 2019. 77, 79, 112, 119, 120, 122

[118] P. REPORT, "Intelligent process automation and the emergence of digital automation platforms," Accessed Feb, 2021. [Online]. Available: https://www.redhat.com/cms/managed-files/mi-451-research-intelligent-process-automation-analyst-paper-f11434-201802.pdf 77

[119] S. Newman, *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015. 78, 111, 126

[120] Gartner, "Gartner forecasts worldwide low-code development technologies market to grow 23% in 2021," Accessed Feb, 2021. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2021-02-15-gartner-forecasts-worldwide-low-code-development-technologies-market-to-grow-23-percent-in-2021 78, 122

[121] G. Daniel, J. Cabot, L. Deruelle, and M. Derras, "Xatkit: A multimodal low-code chatbot development framework," *IEEE Access*, vol. 8, pp. 15 332–15 346, 2020. 79, 111

[122] B. Steffen, T. Margaria, A. Claßen *et al.*, "Heterogeneous analysis and verification for distributed systems," in *Software—Concepts and Tools*, vol. 17, no. 1. Springer Verlag, 1996, pp. 13–25. 79, 134, 164, 183

[123] F. Santos, I. Nunes, and A. L. Bazzan, "Quantitatively assessing the benefits of model-driven development in agent-based modeling and simulation," *Simulation Modelling Practice and Theory*, vol. 104, p. 102126, 2020. 79, 111

[124] H. Moradi, B. Zamani, and K. Zamanifar, "Caasset: A framework for model-driven development of context as a service," *Future Generation Computer Systems*, vol. 105, pp. 61–95, 2020. 79, 110

[125] "Cloud and desktop ide platform," Accessed Feb, 2021. [Online]. Available: https://theia-ide.org/ 80

[126] L. Baresi and M. Garriga, "Microservices: The evolution and extinction of web services?" *Microservices*, pp. 3–28, 2020. 80, 111

[127] F. Rademacher, J. Sorgalla, P. Wizenty, S. Sachweh, and A. Zündorf, "Graphical and textual model-driven microservice development," in *Microservices*. Springer, 2020, pp. 147–179. 80, 111

[128] S. Jörges, C. Kubczak, F. Pageau, and T. Margaria, "Model Driven Design of Reliable Robot Control Programs Using the jABC," in *Proceedings of 4th IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe 2007)*, 2007, pp. 137–148. 80, 102, 108

[129] "Angular dart open source packages," Accessed Feb, 2021. [Online]. Available: https://github.com/angulardart 83

[130] "Amazon translate; fluent and accurate machine translation," Accessed Feb, 2021. [Online]. Available: https://aws.amazon.com/translate/ 87

[131] R. Khan, A. Schieweck, C. Breathnach, and T. Margaria, "Historical civil registration record transcription using an extreme model driven approach," *Proceedings of the Institute for System Programming of the RAS*, vol. 33, no. 3, 2021. 90

[132] V. Jirkovskỳ, M. Obitko, and V. Mařík, "Understanding data heterogeneity in the context of cyber-physical systems integration," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 660–667, 2016. 93

[133] T. Burns, J. Cosgrove, and F. Doyle, "A review of interoperability standards for industry 4.0." *Procedia Manufacturing*, vol. 38, pp. 646–653, 2019. 93

[134] M. Sanchez, E. Exposito, and J. Aguilar, "Industry 4.0: survey from a system integration perspective," *International Journal of Computer Integrated Manufacturing*, vol. 33, no. 10-11, pp. 1017–1041, 2020. 93

[135] S. C. Jepsen, T. I. Mørk, J. Hviid, and T. Worm, "A pilot study of industry 4.0 asset interoperability challenges in an industry 4.0 laboratory," in *2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. IEEE, 2020, pp. 571–575. 94, 95

[136] Z. You and L. Feng, "Integration of industry 4.0 related technologies in construction industry: a framework of cyber-physical system," *IEEE Access*, vol. 8, pp. 122 908–122 922, 2020. 95

[137] H. da Rocha, A. Espirito-Santo, and R. Abrishambaf, "Semantic interoperability in the industry 4.0 using the ieee 1451 standard," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2020, pp. 5243–5248. 95

[138] T. Margaria and B. Steffen, "Service-orientation: conquering complexity with xmdd," in *Conquering complexity*. Springer, 2012, pp. 217–236. 98, 110, 171, 183

[139] B. Rumpe, *Modeling with UML*. Springer, 2016. 98

BIBLIOGRAPHY

[140] "Gaia-x: A federated data infrastructure for europe." Accessed July, 2021. [Online]. Available: https://www.data-infrastructure.eu/ 101

[141] "Fi-ware: The open-source platform for our smart digital future." Accessed July, 2021. [Online]. Available: https://www.fiware.org/ 101

[142] I. Guevara and T. Margaria, "Mazegen: An evolutionary generator for bootstrapping robotic navigation scenarios," 2021(in print). 102

[143] G. A. Farulla, M. Indaco, A. Legay, and T. Margaria, "Model driven design of secure properties for vision-based applications: A case study," in *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2016, p. 159. 102

[144] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000. 103

[145] P. Zweihoff, T. Tegeler, J. Schürmann, A. Bainczyk, and B. Steffen, "Aligned, purpose-driven cooperation: The future way of system development," in *Proc. ISoLA 2021*, ser. LNCS, vol. 13036, 2021. 107

[146] D. Balasubramanian, A. Coglio, A. Dubey, and G. Karsai, "Towards model-based intent-driven adaptive software," in *Proc. ISoLA 2021*, ser. LNCS, vol. 13036, 2021. 107

[147] T. C. Lethbridge, "Low-code is often high-code, so we must design low-code platforms to enable proper software engineering," in *Proc. ISoLA 2021*, ser. LNCS, vol. 13036, 2021. 108

[148] T. Margaria and B. Steffen, "Backtracking-Free Design Planning by Automatic Synthesis in METAFrame," in *Proc. of 1st Int. Conf. on Fundamental Approaches to Software Engineering (FASE 1998), Lisbon, Portugal*, 1998, pp. 188–204. [Online]. Available: http://www.springerlink.com/content/fxtm66d5049hkpt0 108

[149] T. Margaria, M. Bakera, C. Kubczak, S. Naujokat, and B. Steffen, "Automatic Generation of the SWS-Challenge Mediator with jABC/ABC," in *Semantic Web Services Challenge. Results from the First Year*, C. Petrie, T. Margaria, M. Zaremba, and H. Lausen, Eds. Springer Verlag, 2008, pp. 119–138. 108

[150] B. Steffen, F. Howar, M. Isberner, S. Naujokat, and T. Margaria, "Tailored generation of concurrent benchmarks," *Software Tools for Technology Transfer*, vol. 16, no. 5, pp. 543–558, 8 2014. 108

[151] T. Margaria and B. Steffen, "Service Engineering: Linking Business and IT," *Computer*, vol. 39, no. 10, pp. 45–55, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1175939 109

[152] K. Havelund and R. Bocchino, "Integrated modeling and development of component-based embedded software in scala," in *Proc. ISoLA 2021*, ser. LNCS, vol. 13036, 2021. 109

[153] B. Steffen and T. Margaria, "METAFrame in Practice: Design of Intelligent Network Services." in *Correct System Design*, ser. Lecture Notes in Computer Science, E.-R. Olderog and B. Steffen, Eds., vol. 1710. Springer, 1999, pp. 390–415. 110

[154] "Jolie: The service-oriented programming language," Accessed September, 2021. [Online]. Available: https://www.jolie-lang.org/index.html 111

[155] A. Kusiak, "Smart manufacturing," *International Journal of Production Research*, vol. 56, no. 1-2, pp. 508–517, 2018. 119

[156] P. Vincent, K. Iijima, M. Driver, J. Wong, and Y. Natis, "Magic quadrant for enterprise low-code application platforms," *Gartner report*, 2019. 120

[157] J. Rymer and K. Appian, "The forrester wave™: Low-code development platforms for ad&d pros, q4 2017," *Cambridge, MA: Forrester Research*, 2017. 120

[158] T. Margaria, "Components, features, and agents in the abc," in *Objects, Agents, and Features*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 154–174. 121, 140

[159] M. Karusseit and T. Margaria, "Feature-based modelling of a complex, online-reconfigurable decision support service," *Electronic Notes in Theoretical Computer Science*, vol. 157, no. 2, pp. 101–118, 2006. 121, 183

[160] A.-L. Lamprecht and T. Margaria, "Scientific workflows: eternal components, changing interfaces, varying compositions," in *ISoLA 2012*. Springer, 2012, pp. 47–63. 121

[161] B. Jonsson, T. Margaria, G. Naeser, J. Nyström, and B. Steffen, "Incremental requirement specification for evolving systems," *Nord. J. Comput.*, vol. 8, no. 1, pp. 65–87, 2001. 121

[162] C. Kubczak, T. Margaria, B. Steffen, and S. Naujokat, "Service-oriented mediation with jeti/jabc: Verification and export," in *2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Workshops*. IEEE, 2007, pp. 144–147. 121

[163] A.-L. Lamprecht, T. Margaria, and B. Steffen, "Seven variations of an alignment workflow-an illustration of agile process design and management in bio-jeti," in *ISBRA 2008*. Springer, 2008, pp. 445–456. 122

[164] A.-L. Lamprecht, S. Naujokat, T. Margaria, and B. Steffen, "Semantics-based composition of emboss services," *Journal of Biomedical Semantics*, vol. 2, no. 1, pp. 1–21, 2011. 122

[165] B. Steffen, T. Margaria, V. Braun, R. Nisius *et al.*, "A constraint-oriented service creation environment," in *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 1996, pp. 418–421. 122

[166] A.-L. Lamprecht, T. Margaria, and B. Steffen, "Bio-jETI: a framework for semantics-based service composition," *BMC Bioinformatics*, vol. 10 Suppl 10, p. S8, 2009. 123

[167] A. Brossard, C. Kolski, M. Abed, and G. Uster, "Traveler information in its: A model-driven engineering approach to its personalization," in *2014 International Conference on Advanced Logistics and Transport (ICALT)*. IEEE, 2014, pp. 91–96. 123

[168] S. Bag and J. H. C. Pretorius, "Relationships between industry 4.0, sustainable manufacturing and circular economy: proposal of a research framework," *International Journal of Organizational Analysis*, 2020. 124

[169] M. Chadli, J. H. Kim, A. Legay, L.-M. Traonouez, S. Naujokat, B. Steffen, and K. G. Larsen, "A model-based framework for the specification and analysis of hierarchical scheduling systems," in *Critical Systems: Formal Methods and Automated Verification*. Springer, 2016, pp. 133–141. 124

[170] A. Berg, C. P. Donfack, J. Gaedecke, E. Ogkler, S. Plate, K. Schamber, D. Schmidt, Y. Sönmez, F. Treinat, J. Weckwerth *et al.*, *PG 582-industrial programming by example*. Universitätsbibliothek Dortmund, 2015. 124

[171] T. Margaria, B. Steffen, and M. Reitenspieß, "Service-oriented design: The roots," in *Service-Oriented Computing - ICSOC 2005*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 450–464. 124

[172] kaggle, "Kaggle dataset - manufacturing analytics and predictive maintenance," Accessed Feb, 2022. [Online]. Available: https://www.kaggle.com/datasets/arnabbiswas1/microsoft-azure-predictive-maintenance 133

[173] M. Müller-Olm, D. Schmidt, and B. Steffen, "Model-checking," in *International Static Analysis Symposium*. Springer, 1999, pp. 330–354. 134

[174] E. A. Emerson, "Temporal and modal logic," in *Formal Models and Semantics*. Elsevier, 1990, pp. 995–1072. 134

[175] Github, "Github scripts and dataset - manufacturing analytics and predictive maintenance," Accessed Feb, 2022. [Online]. Available: https://github.com/upendradama/Predictive-Maintainance-for-Beverages 137

BIBLIOGRAPHY

[176] "scikit-learn: Machine learning in python," Accessed Feb, 2022. [Online]. Available: https://scikit-learn.org/stable/ 137

[177] T. Margaria, B. D. Floyd, and B. Steffen, "It simply works: simplicity and embedded systems design," in *2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*. IEEE, 2011, pp. 194–199. 140, 164

[178] T. Margaria, "Making sense of complex applications: constructive design, features, and questions," in *Models, Mindsets, Meta: The What, the How, and the Why Not?* Springer, 2019, pp. 129–148. 140

[179] E. Irmak and M. Bozdal, "Internet of things (iot): The most up-to-date challenges, architectures, emerging trends and potential opportunities," *International Journal of Computer Applications*, vol. 975, p. 8887, 2017. 143

[180] Siemens, "Creating a digital thread using low-code to enable digital twins." [Online]. Available: https://www.cimdata.com/en/resources/complimentary-reports-research/commentaries/item/18508-creating-a-digital-thread-using-low-code-to-enable-digital-twins-commentary 143

[181] Siemens, "Low-code platforms assist in the creation and maintenance of digital threads and digital twins," Accessed, March 2023. [Online]. Available: https://resources.sw.siemens.com/en-US/article-mendix-low-code-platforms-assist-digital-thread-digital-twin 143

[182] Statista, "How much faster is low-code development comparing to traditional development?" [Online]. Available: https://www.statista.com/statistics/1254662/low-code-development-speed-compared-traditional-it/ 145

[183] Statista, "Low-code development global platform market revenue," Accessed, March 2022. [Online]. Available: https://www.statista.com/statistics/1226179/low-code-development-platform-market-revenue-global/ 146

[184] PTC, "Thingworx," Accessed, March 2022. [Online]. Available: https://www.ptc.com/ 146

[185] AWS, "Amazon web services iot," Accessed, March 2022. [Online]. Available: https://aws.amazon.com/iot/ 146

[186] Microsoft, "Azure iot," Accessed, March 2022. [Online]. Available: https://www.microsoft.com/ 146

[187] H2o.ai, "H2o.ai," Accessed, March 2022. [Online]. Available: https://www.h2o.ai/ 146

[188] A. Moin, M. Challenger, A. Badii, and S. Günnemann, "Supporting ai engineering on the iot edge through model-driven tinyml," in *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2022, pp. 884–893. 146

[189] P. Iyenghar and E. Pulvermueller, "A model-driven workflow for energy-aware scheduling analysis of iot-enabled use cases," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4914–4925, 2018. 146

[190] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework*. Pearson Education, 2008. 146

[191] S. Jörges, *Construction and Evolution of Code Generators - A Model-Driven and Service-Oriented Approach*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Germany, 2013, vol. 7747. 147

[192] M. Gamboa and E. Syriani, "Improving user productivity in modeling tools by explicitly modeling workflows," *Software & Systems Modeling*, vol. 18, no. 4, pp. 2441–2463, 2019. 147

[193] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. Van Mierlo, and H. Ergin, "Atompm: A web-based modeling environment," in *Joint proceedings of MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition co-located with the 16th International Conference on Model Driven Engineering Languages and Systems*

(MODELS 2013): September 29-October 4, 2013, Miami, USA, 2013, pp. 21–25. 147

[194] S. Becker, H. Koziolek, and R. Reussner, "Model-based performance prediction with the palladio component model," in *Proceedings of the 6th international workshop on Software and performance*, 2007, pp. 54–65. 147

[195] A. C. Contieri, G. G. Correia, T. E. Colanzi, I. M. Gimenes, E. A. Oliveira, S. Ferrari, P. C. Masiero, and A. F. Garcia, "Extending uml components to develop software product-line architectures: Lessons learned," in *Software Architecture: 5th European Conference, ECSA 2011, Essen, Germany, September 13-16, 2011. Proceedings 5*. Springer, 2011, pp. 130–138. 147

[196] C. Seceleanu, A. Vulgarakis, and P. Pettersson, "Remes: A resource model for embedded systems," in *2009 14th IEEE International Conference on Engineering of Complex Computer Systems*. IEEE, 2009, pp. 84–94. 147

[197] "Outsystems: High-performance low-code for app development," Accessed, March 2022. [Online]. Available: https://www.outsystems.com/ 147

[198] "Mendix: Low-code application development platform," Accessed, March 2022. [Online]. Available: https://www.mendix.com/ 147

[199] "Appian platform for process automation," Accessed, March 2022. [Online]. Available: https://appian.com/ 147

[200] N. Nikolov, Y. D. Dessalk, A. Q. Khan, A. Soylu, M. Matskin, A. H. Payberah, and D. Roman, "Conceptualization and scalable execution of big data workflows using domain-specific languages and software containers," *Internet of Things*, vol. 16, p. 100440, 2021. 147

[201] "Node-RED; Low-code programming for event-driven applications," Accessed, March 2023. [Online]. Available: https://nodered.org/ 149

[202] G. Jung, Y. Hong, S. Hong, D. Jang, Y. Jeong, W. Shin, J. Park, D. Kim, C. B. Jeong, D. U. Kim *et al.*, "A low-power embedded poly-si micro-heater for gas sensor platform based on a fet transducer and its application for no2 sensing," *Sensors and Actuators B: Chemical*, vol. 334, p. 129642, 2021. 151

[203] D. Shadrin, A. Menshchikov, D. Ermilov, and A. Somov, "Designing future precision agriculture: Detection of seeds germination using artificial intelligence on a low-power embedded system," *IEEE Sensors Journal*, vol. 19, no. 23, pp. 11 573–11 582, 2019. 151

[204] F. Karray, M. W. Jmal, A. Garcia-Ortiz, and M. a. A. M. Abid, "A comprehensive survey on wireless sensor node hardware platforms," *Computer Networks*, vol. 144, pp. 89–110, 2018. 151

[205] "MongoDB Atlas Database — Multi-Cloud Database Service," Accessed, March 2022. [Online]. Available: https://www.mongodb.com/atlas/database 152, 178

[206] The R Foundation, "R: The r project for statistical computing," Accessed, March 2022. [Online]. Available: https://www.r-project.org/ 152

[207] T. Margaria and B. Steffen, "Lightweight coarse-grained coordination: a scalable system-level approach," *International Journal on Software Tools for Technology Transfer*, vol. 5, no. 2, pp. 107–123, 2004. 164

[208] M. Bakera, T. Margaria, C. Renner, and B. Steffen, "Tool-supported enhancement of diagnosis in model-driven verification," *Innovations in Systems and Software Engineering*, vol. 5, pp. 211–228, 2009. [Online]. Available: http://dx.doi.org/10.1007/s11334-009-0091-6 164

[209] A.-L. Lamprecht, S. Naujokat, T. Margaria, and B. Steffen, "Synthesis-Based Loose Programming," in *Proc. of the 7th Int. Conf. on the Quality of Information and Communications Technology (QUATIC 2010), Porto, Portugal*. IEEE, Sep. 2010, pp. 262–267. 164

[210] O. Niese, B. Steffen, T. Margaria, A. Hagerer, G. Brune, and H.-D. Ide, "Library-Based Design and Consistency Checking of System-Level Industrial Test Cases," in *Fundamental Approaches to Software Engineering*, ser. Lecture Notes in Computer Science, H. Hussmann, Ed. Springer V., 2001, vol. 2029, pp. 233–248. 164

BIBLIOGRAPHY

[211] T. Margaria and B. Steffen, "Continuous model-driven engineering," *Computer*, vol. 42, no. 10, pp. 106–109, 2009. 165

[212] Cisco, "Cisco," March 2022. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html 170

[213] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020. 170

[214] Standard. (2013, May) Iec 62264-1:2013. 170

[215] P. Salhofer and F. Joanneum, "Evaluating the fiware platform: A case-study on implementing smart application with fiware," in *Proceedings of the 51st Hawaii International Conference on System Sciences*, vol. 9, 2018, pp. 5797–5805. 170

[216] A. Braud, G. Fromentoux, B. Radier, and O. Le Grand, "The road to european digital sovereignty with gaia-x and idsa," *IEEE Network*, vol. 35, no. 2, pp. 4–5, 2021. 170

[217] M. Boldo, N. Bombieri, S. Centomo, M. D. Marchi, F. Demrozi, G. Pravadelli, D. Quaglia, and C. Turetta, "Integrating wearable and camera based monitoring in the digital twin for safety assessment in the industry 4.0 era," in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2022. 172

[218] "StereoPi - DIY stereoscopic camera based on Raspberry Pi," Accessed May, 2022. [Online]. Available: https://stereopi.com/ 175

[219] "StereoPi Wiki Main Page," Accessed May, 2022. [Online]. Available: https://wiki.stereopi.com/ 175

[220] "GStreamer — Open Source Multimedia Framework," Accessed May, 2022. [Online]. Available: https://gstreamer.freedesktop.org/ 175

[221] I. Guevara, H. A. A. Chaudhary, and T. Margaria, "A low-code proposal for a rule-based engine integration in a digital thread platform context," *International Manufacturing Conference*, vol. IMC 38, 2022. 176

[222] "Zeromq — an open-source universal messaging library," Accessed May, 2022. [Online]. Available: https://zeromq.org/ 177

[223] "Amazon rekognition — automate your image and video analysis with machine learning," Accessed May, 2022. [Online]. Available: https://aws.amazon.com/rekognition/ 177

[224] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*. IEEE, 2007, pp. 321–334. 178, 182

[225] M. Ambrosin, C. Busold, M. Conti, A.-R. Sadeghi, and M. Schunter, "Updaticator: Updating billions of devices by an efficient, scalable and secure software update distribution over untrusted cache-enabled networks," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 76–93. 179

[226] H. Song, F. Yin, X. Han, T. Luo, and J. Li, "Mpds-rca: Multi-level privacy-preserving data sharing for resisting collusion attacks based on an integration of cp-abe and ldp," *Computers & Security*, vol. 112, p. 102523, 2022. 179