Electronic Communications of the EASST
Volume 81 (2022)

9th International Symposium
on Leveraging Applications of Formal Methods, Verification and Validation
-
Doctoral Symposium, 2021

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

Hafiz Ahmad Awais Chaudhary
Tiziana Margaria

22 pages

Guest Editors: Sven Jörges, Anna-Lena Lamprecht, Anila Mjeda, Stefan Naujokat
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

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

Hafiz Ahmad Awais Chaudhary[1]
Tiziana Margaria[2]

ahmad.chaudhary@ul.ie[1], tiziana.margaria@ul.ie[2]
CSIS, University of Limerick, Limerick, Ireland
Confirm - Centre for Smart Manufacturing
Lero - the Software Research Centre

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.

Keywords: Digital Thread Platform, Model Driven Development, Domain Specific Languages, XMDD, Smart Manufacturing

1 Introduction

The digital transformation in Industry 4.0 integrates manufacturing processes with computing devices, platforms, control systems, communication protocols, and data stores [Kus18]. 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 [DBAF18]. Monitored and optimized manufacturing processes in smart factories demand an increased interoperability between IoT devices and systems with advance capabilities, like autonomous and smart configurations based on historical settings and architectures. These needs are reshaping the way the 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 [SW18, Gou18] 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* [MS19] 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 [Was19].

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 [SGFP20]. LCDEs enable domain experts to take advantage of graphical abstractions and automatic code generation, and develop production-ready applications through model driven engineering principles [VID$^+$19], ideally requiring minimal or no coding experience [RA17, Was19]. 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 semantic, 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 [Sel03]. *Domain specific languages (DSLs)* are tailored for a particular domain, that encapsulate 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 [MHS05]. 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 external 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 [BFK$^+$16] and Pyrus [ZS21] 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 provides an overview of services linking together in the platforms to support the basic operations of smart manufacturing. DSLs with a family of SIBs [SMN$^+$07] are added with small examples that are sufficient to constitute blueprints, i.e. reference examples that the adopters can use as 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 [Mar04a, KM06] intended here as ready-made workflows, where a collections 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 [LM12] is addressed by resorting to hierarchical structuring, the introduction of a feature level abstraction [JMN$^+$01], 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 [KMSN07]. 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 [NFSM14]. 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 [LMS08], and hence also large scale automatic composition and reuse[LNMS11]. 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. 2 discusses the related work, Sect. 3 gives an overview of the principles and architecture of the Digital Thread Platform, Sect. 4 discusses two representative case studies, with integrations in DIME and in Pyrus, and finally in Sect. 5 we conclude and discuss next steps.

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 [BHK16, NLKS18, NNMS16]. According to Gartner [Gar21], 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 [Was19], that are beyond the competence of most domain experts. The first meta-level framework of this category was proposed in 1995 [SMCB96] with immediate industrial applications in the INXpress product by Siemens-Nixdorf [SMB$^+$96]. Framework thinking combined with a low-code approach offers support for 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 [MCF03]. In this line of thought, the jABC platform [SMN$^+$07] based the design and development of applications on formal models and its Lightweight Process Coordination. It accelerated the development process of applications through the concept of reusable building blocks, orchestrated into analyzable control structures called Service Logic Graphs. Following similar practices, several model driven platforms were proposed for model checking [LMS06], early bioinformatics applications [MKNS06, LMS09], fostering collaboration through tool interoperability within the FMICS Working Group on Formal Methods for Industrial Critical Systems [MKSN06, GM12], to compose and combine heterogeneous planning algorithms in Plan-jETI [KMS09, MMK$^+$09] and also mobile apps [BH08] and cross platform [HMK13] applications. Enhancing the model driven paradigm with domain specificity tailors the modeling environment towards a specific application domain, with the purpose of enhanced productivity, reduced complexity and increased domain expert participation [NLKS18]. In general, the development of DSLs may follow any of the available implementation approaches, and may require a collection of guidelines, design patterns, model checking and common language design and implementation challenges for reusability [BAGA14]. A comparison of different implementation approaches of 40 DSLs from very different domains [KMB$^+$08] 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 [BKAU14]. 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 [TPK07].

The Industry 4.0 revolution [JASG21] 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 contribute towards sustainable manufacturing environments. Meanwhile, [BP20] and [SGFP19] 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 [Thr04]. In this context, [MS19] 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 [SGNM19] 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 [NLKS18] 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 [ZNS19, BFK$^+$16, CKL$^+$16, BDG$^+$15]. In our Digital Thread platform we chose to work with two specific products generated from Cinco meta-models: DIME [BFK$^+$16] and Pyrus [ZS21].

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 [MSR05] is an example of the One Thing Approach (OTA) defined in [MS09]. 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 [SGNM19], OTA [MS09] and eXtreme Model Driven Development

(XMDD) [MS20]. 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.

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 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 [MCG+21] 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 [CM21b], we implemented a generic extension mechanism, in analogy to micro-service architectures [New15], 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 [Hev07] 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

[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

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 solutions co-design infrastructure between commercial providers of specific equipment and their customers, as in ADI's Catalyst collaboration hub[2]. In such equipped spaces, the goal is to try out established and new technologies and their combination, experiment with feasibility and fact-finding, with a relatively short turn around cycles of individual experiments and testbeds (a few weeks to a few months), in order to distill 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.

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. 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 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 sub-processes, 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** 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

[2] 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

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 [CM21b], Data analytics with R libraries [MCG+21] and the MongoDB no-SQL database were integrated as services, while R,

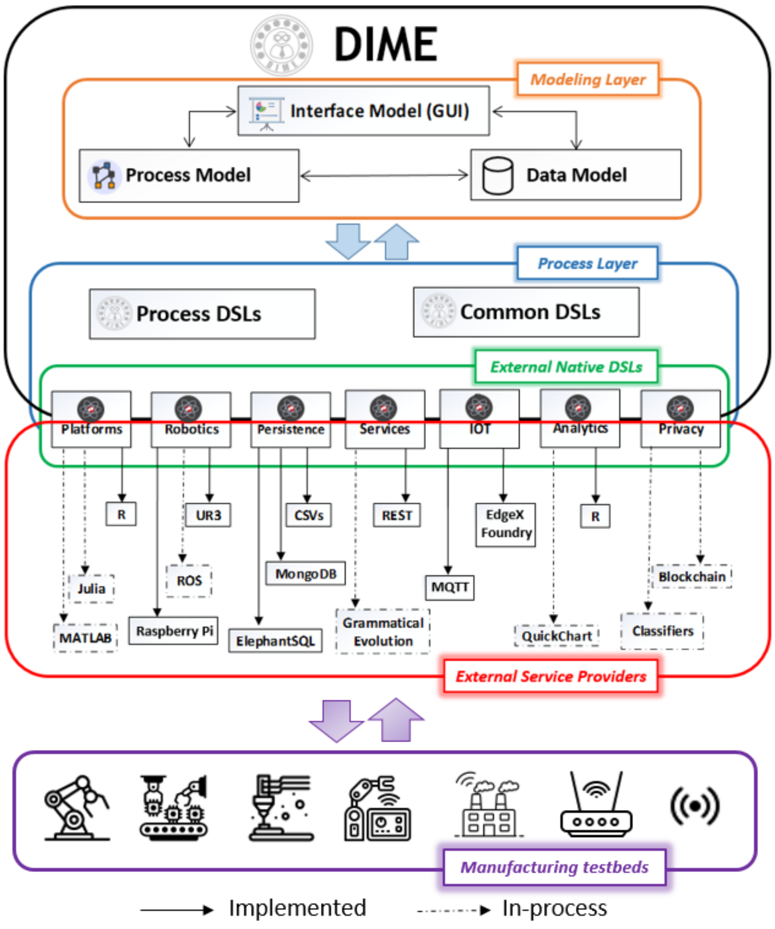


Figure 1: Architecture Overview of the Digital Thread Platform - DIME

seen as a programming language, and the EdgeX Foundry middleware integration and interoperability platform had to be integrated as platforms [CM21a], which is the more general but also more complex integration form. EdgeX Foundry [EFF22] 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.

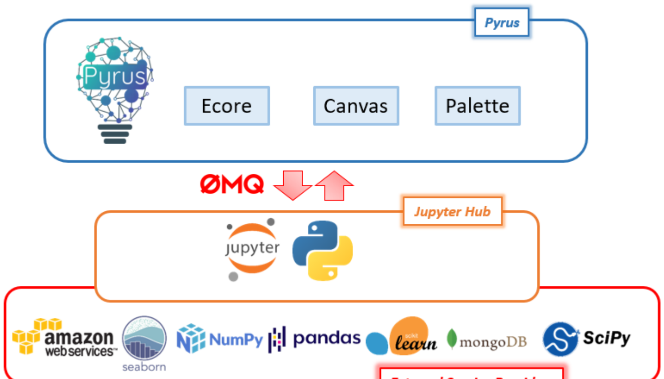


Figure 2: Architecture Overview of the Analytics Platform - Pyrus

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. 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, annotation 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 [CM21b] and cloud databases like many AWS services, MongoDB etc.

For functions discovery and the 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.

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 to them the 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 the 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 [CM21a] 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 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. 4 discuss small workflows that provide examples of end to end implementation of various stages of manufacturing analytics and predictive maintenance.

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 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.

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 [kag22] and consists of historical data from 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.

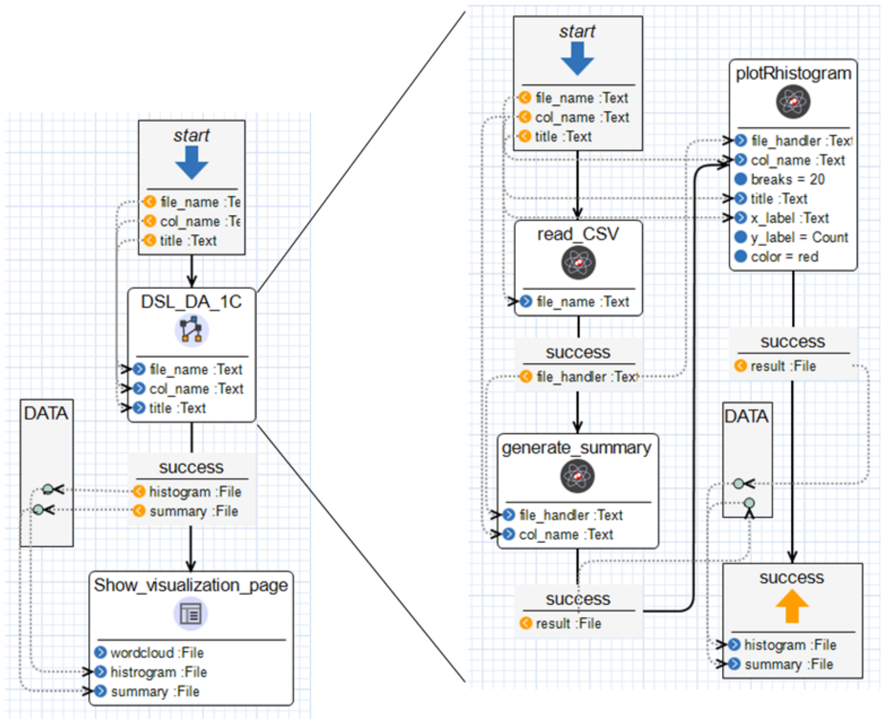
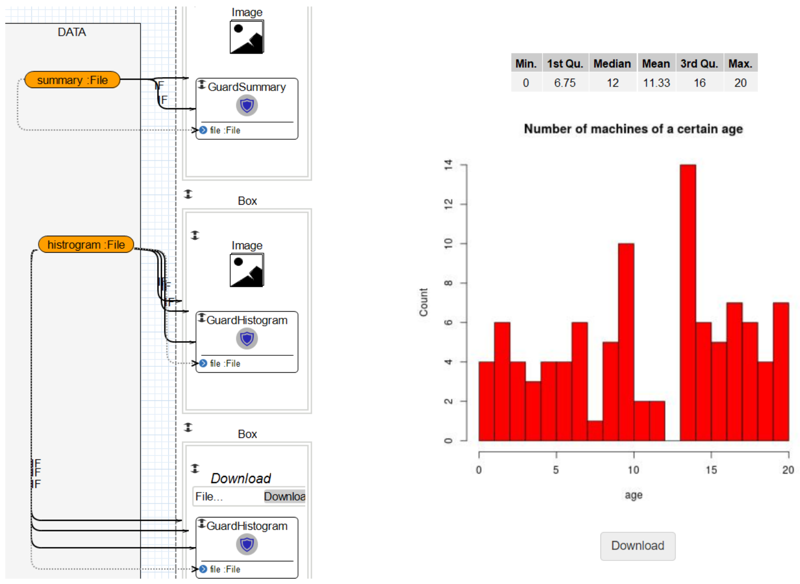


Figure 3: Hierarchical Process Model for Manufacturing Analytics

Fig. 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. 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) [MSS99], they are also amenable to data flow analysis and formal verification, e.g. by means of model checking for temporal logics [Eme90, SMC+96].

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.



Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0	6.75	12	11.33	16	20

Figure 4: Manufacturing Analytics DSL: Interface Model (left) and web page (right) for a Manufacturing Analytics page

Fig. 4 (left) shows a segment of the GUI model for this page and Fig. 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. 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. 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. 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. 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.

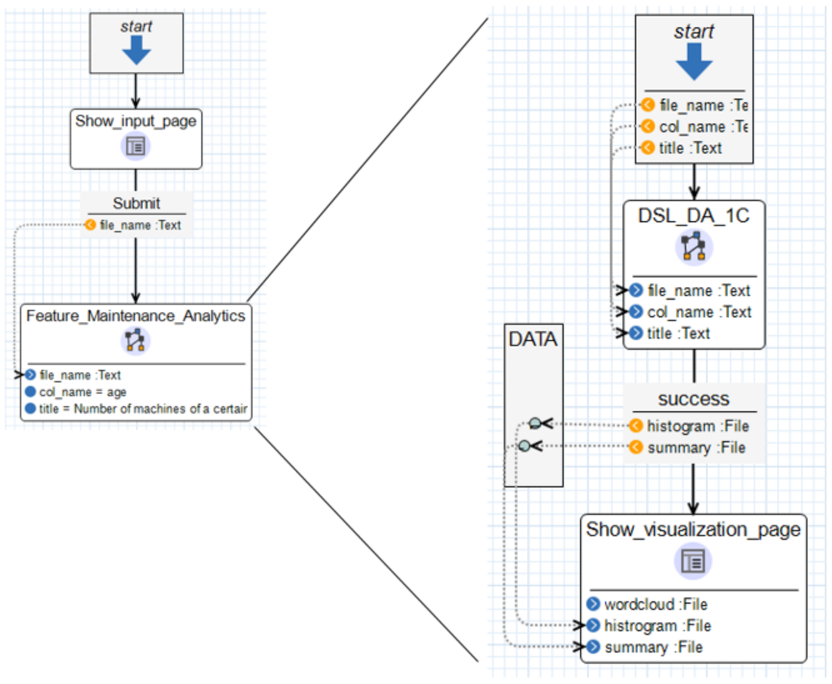
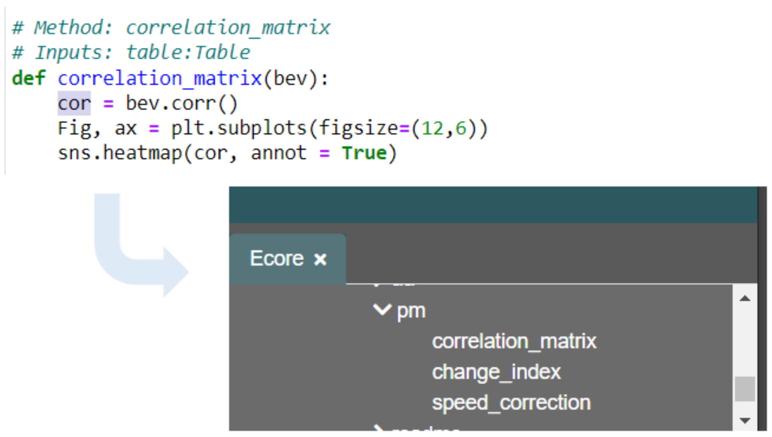


Figure 5: Hierarchical Process SIB for Maintenance Analytics

The hierarchical workflow presented in Fig. 3 and Fig. 4 is easily encapsulated as a single process SIB for maintenance analytics: the Feature_Maintenance_Analytics shown in Fig. 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.



```
# 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 6: Correlation Matrix: Python function annotation in Jupyter and its visualization in Pyrus

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 [git22]. Fig. 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 [sci22] and many other popular libraries like Matplotlib for numerical plotting, Numpy, Pandas, SciPy etc.

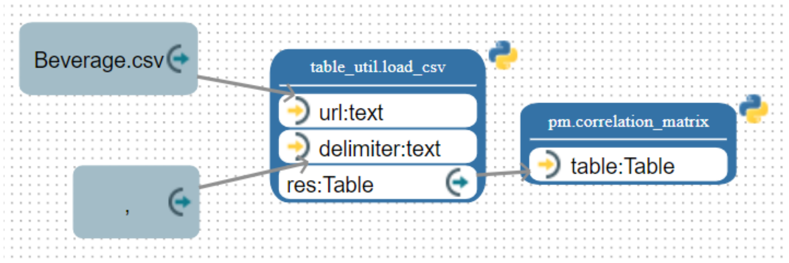


Figure 7: Pyrus Data Pipeline for Correlation Matrix display within a Predictive Maintenance DSL

The Pyrus pipeline shown in Fig. 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

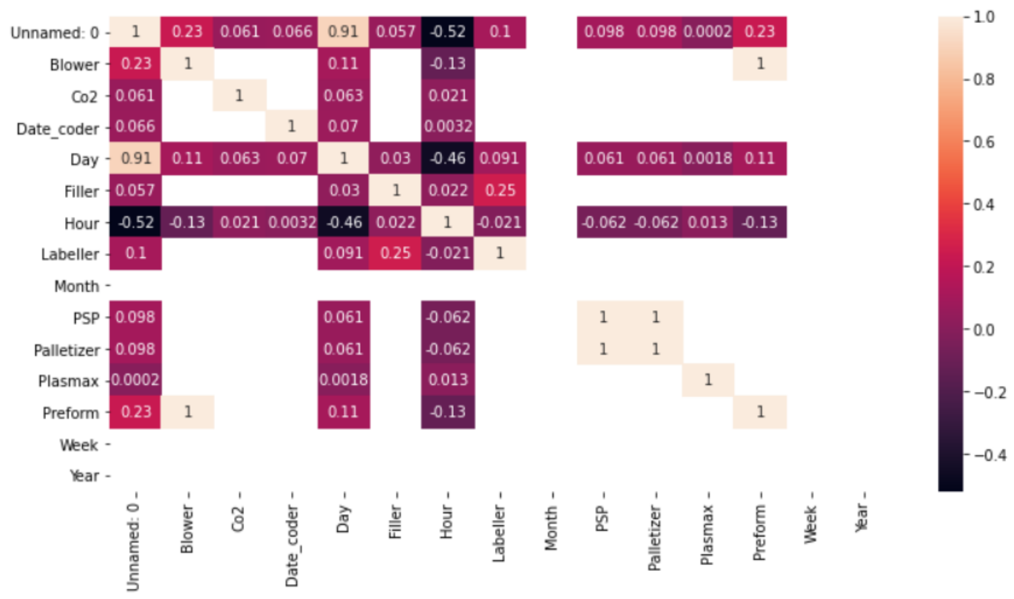


Figure 8: Heat Map for the Beverage Industry

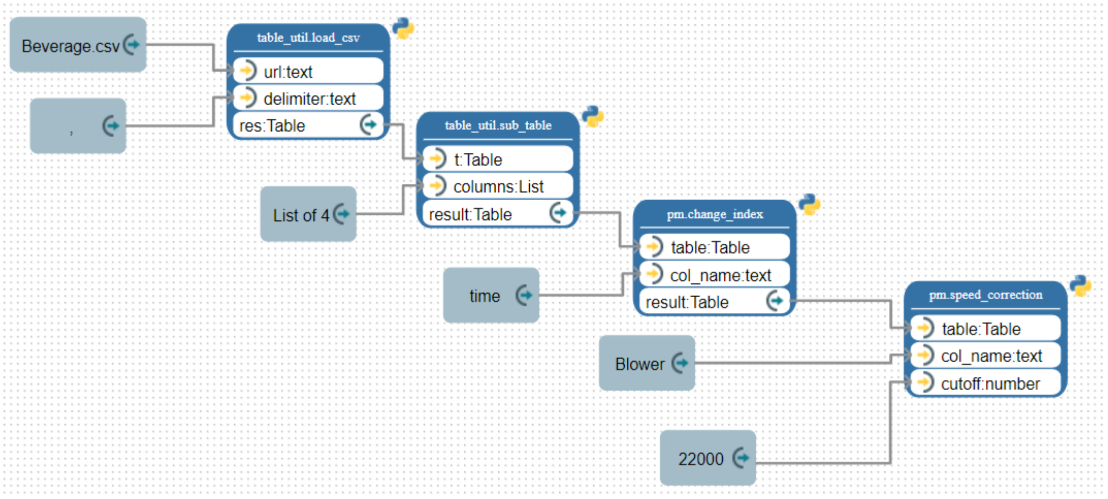


Figure 9: Pyrus Data Pipeline for Predictive Maintenance DSL

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.

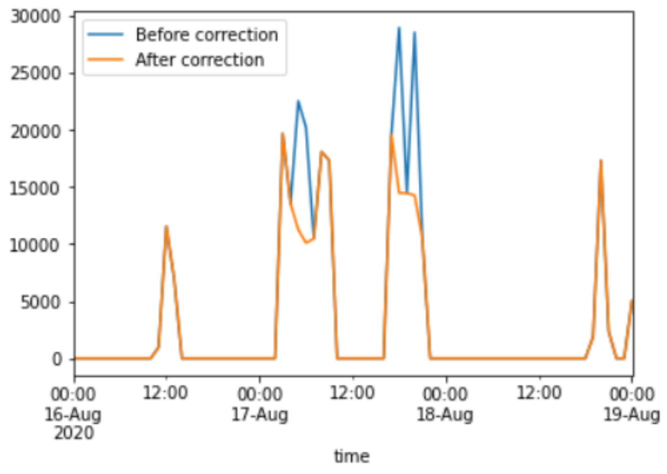


Figure 10: Predictive Maintenance: Plot of the corrected vs. original behaviours

Similarly, Fig. 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 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. 10 shows the Blower speed before (blue plot 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. 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.

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 [MFS11] and reusability, and using hierarchy for scalability to larger and more complex systems [Mar04b, Mar19], we are also moving from individual reusable SIBs to *features*, intended here as ready-made workflows, where a collections 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.

Acknowledgements: 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).

Bibliography

[BAGA14] A. Barisic, V. Amaral, M. Goulao, 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.

[BDG$^+$15] 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.

[BFK$^+$16] S. Boßelmann, M. Frohme, D. Kopetzki, M. Lybecait, S. Naujokat, J. Neubauer, D. Wirkner, P. Zweihoff, B. Steffen. DIME: A Programming-Less Modeling Environment for Web Applications. In Margaria and Steffen (eds.), *ISoLA 2016*. LNCS 9953, pp. 809–832. Springer International Publishing, Cham, 2016.

[BH08] F. T. Balagtas-Fernandez, H. Hussmann. Model-driven development of mobile applications. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. Pp. 509–512. 2008.

[BHK16] M. Broy, K. Havelund, R. Kumar. Towards a unified view of modeling and programming. In *ISoLA 2016*. Pp. 238–257. 2016.

[BKAU14] A. Brossard, C. Kolski, M. Abed, G. Uster. Traveler information in ITS: A Model-Driven Engineering approach to its personalization. In *2014 International Conference on Advanced Logistics and Transport (ICALT)*. Pp. 91–96. 2014.

[BP20] S. Bag, 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.

[CKL$^+$16] M. Chadli, J. H. Kim, A. Legay, L.-M. Traonouez, S. Naujokat, B. Steffen, K. G. Larsen. A model-based framework for the specification and analysis of hierarchical scheduling systems. In *Critical Systems: Formal Methods and Automated Verification*. Pp. 133–141. Springer, 2016.

[CM21a] H. A. A. Chaudhary, T. Margaria. Integrating external services in DIME. In *ISoLA 2021*. LNCS 13036, pp. 41–54. 2021.

[CM21b] H. A. A. Chaudhary, 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* 33(4), 2021.

[DBAF18] L. S. Dalenogare, G. B. Benitez, N. F. Ayala, A. G. Frank. The expected contribution of Industry 4.0 technologies for industrial performance. *International Journal of production economics* 204:383–394, 2018.

[EFF22] EdgeX Foundry: The EdgeX Foundry platform documentation. Accessed Feb, 2022.
https://docs.edgexfoundry.org/2.1/

[Eme90] E. A. Emerson. Temporal and modal logic. In *Formal Models and Semantics*. Pp. 995–1072. Elsevier, 1990.

[Gar21] Gartner. Gartner Forecasts Worldwide Low-Code Development Technologies Market to Grow 23% in 2021. Accessed Feb, 2021.
https://gtnr.it/3pZSQyJ

[git22] github. Github scripts and dataset - Manufacturing Analytics and Predictive Maintenance. Accessed Feb, 2022.
https://github.com/upendradama/Predictive-Maintainance-for-Beverages

[GM12] S. Gnesi, T. Margaria. *Formal methods for industrial critical systems: A survey of applications*. John Wiley & Sons, 2012.

[Gou18] L. S. Gould. What are digital twins and digital threads? *Automotive Design & Production* 23, 2018.

[Hev07] A. R. Hevner. A three cycle view of design science research. *Scandinavian journal of information systems* 19(2):4, 2007.

[HMK13] H. Heitkötter, T. A. Majchrzak, H. Kuchen. Cross-platform model-driven development of mobile applications with md2. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. Pp. 526–533. 2013.

[JASG21] A. Jamwal, R. Agrawal, M. Sharma, A. Giallanza. Industry 4.0 technologies for manufacturing sustainability: a systematic review and future research directions. *Applied Sciences* 11(12):5725, 2021.

[JMN$^+$01] B. Jonsson, T. Margaria, G. Naeser, J. Nyström, B. Steffen. Incremental requirement specification for evolving systems. *Nord. J. Comput.* 8(1):65–87, 2001.

[kag22] kaggle. Kaggle dataset - Manufacturing Analytics and Predictive Maintenance. Accessed Feb, 2022.
https://www.kaggle.com/datasets/arnabbiswas1/microsoft-azure-predictive-maintenance

[KM06] M. Karusseit, T. Margaria. Feature-based modelling of a complex, online-reconfigurable decision support service. *Electronic Notes in Theoretical Computer Science* 157(2):101–118, 2006.

[KMB$^+$08] 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* 50(5):390–405, 2008.

[KMS09] C. Kubczak, T. Margaria, B. Steffen. Mashup Development for Everybody A Planning-Based Approach. 2009.

[KMSN07] C. Kubczak, T. Margaria, B. Steffen, 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*. Pp. 144–147. 2007.

[Kus18] A. Kusiak. Smart manufacturing. *International Journal of Production Research* 56(1-2):508–517, 2018.

[LM12] A.-L. Lamprecht, T. Margaria. Scientific workflows: eternal components, changing interfaces, varying compositions. In *ISoLA 2012*. Pp. 47–63. 2012.

[LMS06] A.-L. Lamprecht, T. Margaria, B. Steffen. Data-flow analysis as model checking within the jABC. In *International Conference on Compiler Construction*. Pp. 101–104. 2006.

[LMS08] A.-L. Lamprecht, T. Margaria, B. Steffen. Seven variations of an alignment workflow-An illustration of agile process design and management in Bio-jETI. In *ISBRA 2008*. Pp. 445–456. 2008.

[LMS09] A.-L. Lamprecht, T. Margaria, B. Steffen. Bio-jETI: a framework for semantics-based service composition. *BMC bioinformatics* 10(10):1–19, 2009.

[LNMS11] A.-L. Lamprecht, S. Naujokat, T. Margaria, B. Steffen. Semantics-based composition of EMBOSS services. *Journal of Biomedical Semantics* 2(1):1–21, 2011.

[Mar04a] T. Margaria. Components, Features, and Agents in the ABC. In Ryan et al. (eds.), *Objects, Agents, and Features*. Pp. 154–174. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[Mar04b] T. Margaria. Components, Features, and Agents in the ABC. In *Objects, Agents, and Features*. Pp. 154–174. Springer, 2004.

[Mar19] T. Margaria. Making sense of complex applications: constructive design, features, and questions. In *Models, Mindsets, Meta: The What, the How, and the Why Not?* Pp. 129–148. Springer, 2019.

[MCF03] S. J. Mellor, T. Clark, T. Futagami. Model-driven development: guest editors' introduction. IEEE Software, 20 (5). pp. 14-18. ISSN 0740-7459. *IEEE software* 20(5):14–18, 2003.

[MCG$^+$21] T. Margaria, H. A. A. Chaudhary, I. Guevara, S. Ryan, A. Schieweck. The interoperability challenge: building a model-driven digital thread platform for CPS. In *ISoLA 2021*. LNCS 13036, pp. 393–413. 2021.

[MFS11] T. Margaria, B. D. Floyd, B. Steffen. IT simply works: simplicity and embedded systems design. In *2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*. Pp. 194–199. 2011.

[MHS05] M. Mernik, J. Heering, A. M. Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)* 37(4):316–344, 2005.

[MKNS06] T. Margaria, C. Kubczak, M. Njoku, B. Steffen. Model-based design of distributed collaborative bioinformatics processes in the jABC. In *11th IEEE ICECCS*. Pp. 8 pp.–. 2006.
doi:10.1109/ICECCS.2006.1690366

[MKSN06] T. Margaria, C. Kubczak, B. Steffen, S. Naujokat. The FMICS-jETI platform: Status and perspectives. In *ISoLA 2006)*. Pp. 402–407. 2006.

[MMK$^+$09] T. Margaria, D. Meyer, C. Kubczak, M. Isberner, B. Steffen. Synthesizing semantic web service compositions with jMosel and Golog. In *ISWC 2009*. Pp. 392–407. 2009.

[MS09] T. Margaria, B. Steffen. Business process modeling in the jABC: the one-thing approach. In *Handbook of research on business process modeling*. Pp. 1–26. IGI Global, 2009.

[MS19] T. Margaria, A. Schieweck. The digital thread in industry 4.0. In *International Conference on Integrated Formal Methods*. LNCS 11918, pp. 3–24. 2019.

[MS20] T. Margaria, 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.

[MSR05] T. Margaria, B. Steffen, M. Reitenspieß. Service-Oriented Design: The Roots. In Benatallah et al. (eds.), *Service-Oriented Computing - ICSOC 2005*. Pp. 450–464. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[MSS99] M. Müller-Olm, D. Schmidt, B. Steffen. Model-checking. In *International Static Analysis Symposium*. Pp. 330–354. 1999.

[New15] S. Newman. *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.

[NFSM14] J. Neubauer, M. Frohme, B. Steffen, T. Margaria. Prototype-driven development of web applications with DyWA. In *ISoLA 2014*. Pp. 56–72. 2014.

[NLKS18] S. Naujokat, M. Lybecait, D. Kopetzki, B. Steffen. CINCO: a simplicity-driven approach to full generation of domain-specific graphical modeling tools. *STTT 2018* 20:1–28, 06 2018.
doi:10.1007/s10009-017-0453-6

[NNMS16] S. Naujokat, J. Neubauer, T. Margaria, B. Steffen. Meta-level reuse for mastering domain specialization. In *ISoLA 2016*. Pp. 218–237. 2016.

[RA17] J. Rymer, K. Appian. The Forrester Wave™: Low-Code Development Platforms For AD&D Pros, Q4 2017. *Cambridge, MA: Forrester Research*, 2017.

[sci22] scikit-learn: Machine Learning in Python. Accessed Feb, 2022.
https://scikit-learn.org/stable/

[Sel03] B. Selic. The pragmatics of model-driven development. *IEEE software* 20(5):19–25, 2003.

[SGFP19] R. Sanchis, Ó. García-Perales, F. Fraile, R. Poler. Low-code as enabler of digital transformation in manufacturing industry. *Applied Sciences* 10(1):12, 2019.

[SGFP20] R. Sanchis, Ó. García-Perales, F. Fraile, R. Poler. Low-code as enabler of digital transformation in manufacturing industry. *Applied Sciences* 10(1):12, 2020.

[SGNM19] B. Steffen, F. Gossen, S. Naujokat, T. Margaria. Language-driven engineering: from general-purpose to purpose-specific languages. In *Computing and Software Science*. Pp. 311–344. Springer, 2019.

[SMB$^+$96] 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*. Pp. 418–421. 1996.

[SMC$^+$96] B. Steffen, T. Margaria, A. Claßen et al. Heterogeneous Analysis and Verification for Distributed Systems. In *Software—Concepts and Tools*. Volume 17(1), pp. 13–25. 1996.

[SMCB96] B. Steffen, T. Margaria, A. Claßen, V. Braun. The METAFrame'95 environment. In Alur and Henzinger (eds.), *CAV*. Pp. 450–453. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[SMN$^+$07] B. Steffen, T. Margaria, R. Nagel, S. Jörges, C. Kubczak. Model-Driven Development with the jABC. In Bin et al. (eds.), *Hardware and Software, Verification and Testing*. LNCS 4383, pp. 92–108. Springer Berlin Heidelberg, 2007.

[SW18] V. Singh, K. E. Willcox. Engineering design with digital thread. *AIAA Journal* 56(11):4515–4528, 2018.

[Thr04] 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*. Pp. 497–502. 2004.

[TPK07] K. Thramboulidis, D. Perdikis, S. Kantas. Model driven development of distributed control applications. *IJAMT 2007* 33(3):233–242, 2007.

[VID$^+$19] P. Vincent, K. Iijima, M. Driver, J. Wong, Y. Natis. Magic quadrant for enterprise low-code application platforms. *Gartner report*, 2019.

[Was19] R. Waszkowski. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine* 52(10):376–381, 2019.

[ZNS19] P. Zweihoff, S. Naujokat, B. Steffen. Pyro: Generating Domain-Specific Collaborative Online Modeling Environments. In Hähnle and Aalst (eds.), *FASE 2019*. LNCS 11424, pp. 101–115. Springer International Publishing, Cham, 2019.

[ZS21] P. Zweihoff, B. Steffen. Pyrus: an online modeling environment for no-code data-analytics service composition. In *ISoLA 2021*. LNCS 13036, pp. 18–40. 2021.