Needs for a Holistic Approach to Engineering Reusable, Configurable, and Composable Development Digital Twins

for Multi-disciplinary, Cyber-physical Product Development Purposes

Eugen Schindler
Research & Development
Canon Production Printing
Venlo, The Netherlands
Github / LinkedIn

Hristina Moneva
Research & Development
Canon Production Printing
Venlo, The Netherlands
LinkedIn

ABSTRACT

This text proposes a high-level holistic approach for engineering reusable, configurable, and composable Digital Twins (DTs) for product-development (devDTs). We address the current ambiguity in DT definitions by clearly distinguishing devDTs from other high-level types of DTs and providing operational definitions for them, serving as a mental framework to systematically develop and integrate devDT ingredients.

The core of our approach lies in the concept of a DT-factory, which enables on-demand instantiation of (hybrid) devDTs from an organization's (or an ecosystem's) digital and physical product- and technology-development assets. This enables on one hand a separation of concerns on our company's strategic domains, such as product development, systems modeling, data engineering, data science, and AI. On the other hand, the technological, conceptual, and methodological complexity for users of devDTs can be systematically reduced to its essence.

Many of the separate ingredients mentioned in this text are already well-proven in various software-development and organizational contexts (which is what we want to build on), but the systematic combination of the concepts and technology presented in this paper introduces a novel conceptual base for a methodology that not only focuses on the technological aspects but also enables effective explanation, adoption strategies, and achieving deep digital transformation of complex research and development (R&D) activities for stakeholders.

To scope the validity of our proposal: validation has and is continuously being done in the complex context of Canon Production Printing's R&D activities, which gives us the necessary confidence in the scalability and maintainability of our holistic approach, ensuring its viability in the face of evolving market demands and competitor advancements. Together with our company's strategy to develop products relying primarily on DTs rather than physical assets only, we believe we have a strong high-level starting point, inviting further research and collaboration to refine the approach.

CCS CONCEPTS

• Computing Methodologies → Modeling and simulation • Hardware → Hardware validation → Model checking • Software and its engineering → Software organization and properties → Software system structures → Software system models → Model-driven software engineering

KEYWORDS

digital twin, digital twinning, model-based development, model-driven development, digital transformation, systems modelling, system development

1 Introduction

Disclaimer: The content of this text represents the personal opinions and interpretations of the authors. The information, analyses, and conclusions expressed herein are solely those of the authors and do not necessarily reflect the views or policies of our employer. Our employer cannot be held responsible for any inconsistencies, inaccuracies, or errors in the content of this paper. The authors alone are responsible for any mistakes or omissions and for the validity of the information presented.

In the previous years, a lot of work has been done to conceptualize, define, and systemically map out digital twins (DTs) and their usages [1,18,13,19 and the various transitive citations or linked contents]. Coming from the domain of developing products in multi-disciplinary cyber-physical systems, we could not find an integrated methodology to engineer DTs for everyday practical use in product development activities in a composable way that makes it sustainable for a commercial business driven by strong performance, quality, and pricing market requirements to incrementally roadmap and develop digital assets and corresponding digital competences with separated concerns and spread over various teams, while enabling holistic view and operation of a relevant subset of the digital assets as a DT for a specific purpose.

Many examples of DTs exist, but most real-world examples one can easily find by doing a (limited) best-effort search support the operation of a product (e.g. dashboards, supervisory controllers, any other types that run next to an operational product in the field with a (soft)real-time digital thread). None of these has given us the tools that we need for our company to fully leave behind old (in the core document-

centric) approaches and deeply digitally transform product development activities. Therefore, we will provide an additional set of definitions in this text which can be used to realize digital twins with the above-described desired properties. The presented abstractions are defined in such a way that they can be mapped to logical digital twin component types, such as described in e.g. [1].

One main driver for our approach is interoperability. At the highest level, digital specification assets and digital measurement/logging assets (a.k.a. "models and data") are usually separate worlds, even though there is quite a long history of modeling and data science/engineering. Our approach seeks to make the "two worlds" explicitly interoperable so they can be unified.

Finally, the technological and techno-conceptual base is a good starting point, but not enough to also operationalize deep digital transformation. For this to work, the approach needs to include explanation and incremental introduction (dosing) of the new concepts, method, and way of working in a sustainable way without just "resetting everything overnight".

2 Definitions & Constraints

It should be noted that the definitions given here are not in any way scientific definitions, but rather operational definitions that make it possible to perform engineering on various defined elements (i.e. to combine or compose them in consistent ways and build more complex twinning constructions or (reference) architectures).

In addition to definitions, some constraints on building blocks are described. The principle underlying these constraints is that we specify a system using a heterogeneous set of tools (each context and discipline may choose the digital tools and languages which are optimal for their domain, with some constraints to enable interoperability) instead of "one tool to rule them all". We recognize efforts such as [12] (which make it easier to implement what we propose), but argue that organizations (companies or otherwise) need to take control of their own domain and digital assets to fully enable them in their digitally transformed activities.

2.1 DT Building Blocks

The first basic definition is that of a *system* (i.e. the physical twin), being a group of interacting/interrelated elements that act according to a set of rules to form a unified whole [22]. Everything in this definition can be prefixed by *physical*, which scopes down to physical twins only, but this might exclude twinning of non-physical systems, such as "physical twins" made up of software or a rule system like the law (a discussion we had with various people). Therefore, we prefer to keep the definition broader.

One can argue for a DT to be a fundamental unit of composition (i.e. "DTs all the way down" [23]) and that may be desirable from a scientific or purely mathematical point of view, but we have discovered that in practice it is better to make clear distinction between a DT and the various building blocks it is created of. We call one of these building blocks the *digital asset* and make a distinction between the following types:

- *specification*: describes some functionality, aspect, or structure of a system in a processable digital format and allows for system specification and analysis; examples: component model of a product's architecture, CAD model of the mechanical geometry of a product's part, state machine describing behavior of a product's control software component.
- *data*: digitally captures logging from a product (a specific type of system that can produce its own data) or measurements/information from a laboratory setup; examples: logging from sensors in a product, high-fidelity capturing of some physical process, motion-capture data of people's movements in a lab or in the field while using a system.
- *tool*: enables either the (manual) maintenance of specification assets (e.g. editing of a CAD model, editing of a state machine model, editing of a component in a component model) or computation on specifications and/or data (e.g. simulation, data analysis, any kind of computation), or both; this asset is on a different meta-level from the previous two, i.e. a digital tool typically determines the language or meta-model of the specifications that are written in it, but it also determines the format for computation output.

Note that "all digital twins are digital assets", but not "all digital assets are digital twins". The mental model here is that various contexts (projects, teams, departments) in a company can develop and operate various digital assets potentially in isolation, which makes the "business case" for each context much easier to be concretized for the stakeholders in that specific context.

The other building block is called a *digital link*, which is an interface between two or more digital assets that allows for them to interoperate in a bigger scope, allowing for composable, increasingly complex specification, analysis, and simulation. Here we also distinguish different types:

- traceability link: a "live" association between an element in one specification and an element in another specification; transitive chaining of traceability links enables navigability between specifications
- harmonized specification-to-specification link: an interface which enables automatic export of elements from one specification and import of those elements into another specification; this requires that the tools which are used to maintain the specifications in question speak the same language for the exchanged set of elements, i.e. the concepts (or meta-model) of the exchanged set of elements should match in both tools that enable this automated exchange of elements
- computation-output-to-input link: an interface which enables "extracting" computation results from a computation executed in one tool and "injecting" these as

inputs to a new computation executed another (or the same) tool
- specification-to-computation link: an interface which enables querying (typically numeric) information from elements in a specification and "injecting" it as input(s) to a computation execution

In order to create automated links between specifications and data, the various tools must have non-interactive interfaces (files/command-line or another type of API) for starting the tool, execution of a computation, and data exchange. Furthermore, harmonized linking (and thus automated – and possibly repeated - transport of specification elements between tools) requires the elements in a tool to be uniquely identifiable, such that incremental import/export of elements without breaking internal links in a specification to elements imported from another specification is possible.

2.2 DT Types

Although various types of DTs have already been introduced (e.g. [20] which has been prominently quoted on Wikipedia [21]), we will define another cross-section of types, which is mainly centered around (but not necessarily restricted to) digital twinning for (multi-disciplinary cyber-physical) product development. The mental model behind the distinction of types we make is from a product creation point of view. A high-tech company that creates complex multi-disciplinary cyber-physical products, usually needs to develop a product that optimally balances functionality with various quality attributes (such as performance, costs, and productivity). To find this balance, various questions about potential architecture-, design-, and implementation choices for a product must be answered. In addition to reasoning, there two primary tools to try out different design choices and answer questions: physical assets (e.g. test setups, laboratory measurement setups, physical prototypes) and digital assets (e.g. specifications, simulations, logging/measurement data). The deliverable the product developers make is a comprehensive and holistic specification which describes how to build the product, how to service it, and how to match the capabilities of products with current and emerging market developments.

The three types of DTs we distinguish are (see also Figure 1 for an overview):

- Operational Support DTs (osDTs): these DTs run in parallel with a product in the field and optimize or improve its operations; the digital thread is typically a (soft)real-time streaming data interface and the DT can be effectively anything on the spectrum from a dashboard which allows operators of the product to optimize its operations, all the way to a fully automated supervisory control system which automatically optimizes or improves the product's operation (operator support, service support, etc.); one can wonder whether an osDT is part of the product it is twinning [24,25] and a lot can and has been said from a scientific point of view, but from an engineering point of view, we can pragmatically simplify things by stating that "in the logical view", the osDT and the physical twin are separate, while "in the deployment view" they can be intertwined in sometimes very beneficial and sometimes very tricky ways, leaving consequent considerations on this topic out of scope for this text.
- Natural System DTs (nsDTs): these DTs emulate some natural system's (structural and behavioral) properties with a degree of usable fidelity (for one or more specific purposes); the digital thread is usually made up of measurement data captured from a laboratory measurement setup; examples: game development assets (with their shape, optical properties, and some physical properties) scanned in a lab, a heart, a molecule, the process of grass growing.
- Development DTs (devDTs): these DTs twin a product (platform) that is being developed, all the way from its early inception phase, through development, production, servicing, etc. until product sunset; the main difference with the earlier types is that the physical twin can be anything along the spectrum of product idea, a number of prototype setups (possibly in a hybrid digital/physical setup), an integrated prototype, or several (parts of) deployed products in the field.

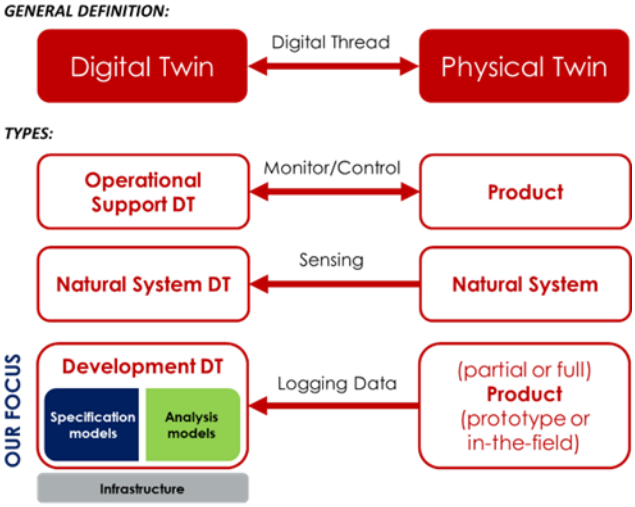

Figure 1: Types of DTs by their use

A devDT can use nsDTs for answering various properties based on physical behavior of important natural systems the product operates on (such as a printer devDT that can operate on the above mentioned nsDTs of ink and a paper substrate). Also, various digital assets that make up a devDT can be evolved, if so desired, to a stage that they can be used in-product or as part of an osDT for a product, which also means that there are various relations possible between devDTs and osDTs.

The rest of this text will focus on devDTs.

3 Vision and Enablers

3.1 Vision

Our vision is to have a holistic approach for digitally transformed product development, where we:

Capture our knowledge and product/platform (in terms of architecture, design, and implementation specifications)

in reusable, composable, and (re)configurable models,

which leads to a

sustainable way to have our knowledge and product documentation

in explorable, learnable, and virtualizable form.

3.2 Composition Mechanisms, Infrastructure, and Use Cases

Having pinned down the fundamental building blocks in Section 2.1, we can start defining ways in which it makes sense to compose them. The devDT as shown in Figure 1 (previous page) contains specification models (encoded in digital specification assets) and analysis models (encoded in tools which have a computational component). The depicted devDT also obtains its assets from which it is constructed from and can be "run" on a storage and execution infrastructure.

Before we can construct and run a digital twin from various digital assets, we need to consider two composition mechanisms (see Figure 2 for a more graphical overview):

3.2.1 Digital Specification Network (DSN). Using the earlier described traceability linking and harmonized specification-to-specification linking mechanisms, we can (but don't necessarily must) connect digital (typically product or product platform) specification assets in a DSN of heterogeneous (as opposed to storing all specifications in one tool or one database) specification assets (depicted as a set of interconnected circles in a grey plane inside the Product Specifications box, named *domain models*). If a DSN has only harmonized links (depicted as a solid line), it is fully harmonized DSN and if it contains traceability links (dotted line), it is hybrid. Obviously, a fully harmonized DSN is the most powerful, since it enables a complete *single-point-of-truth* specification of a product. Note that the DSN doesn't require necessarily require a specific way in which elements are transported between nodes. We encourage as much as possible the use of interoperability mechanisms between standard tools.

There are two types of DSN tools that are special in a sense that they can serve as a *backbone* for a DSN: component models and feature models. Component models make it possible to specify an architectural or design decomposition of the product on system-

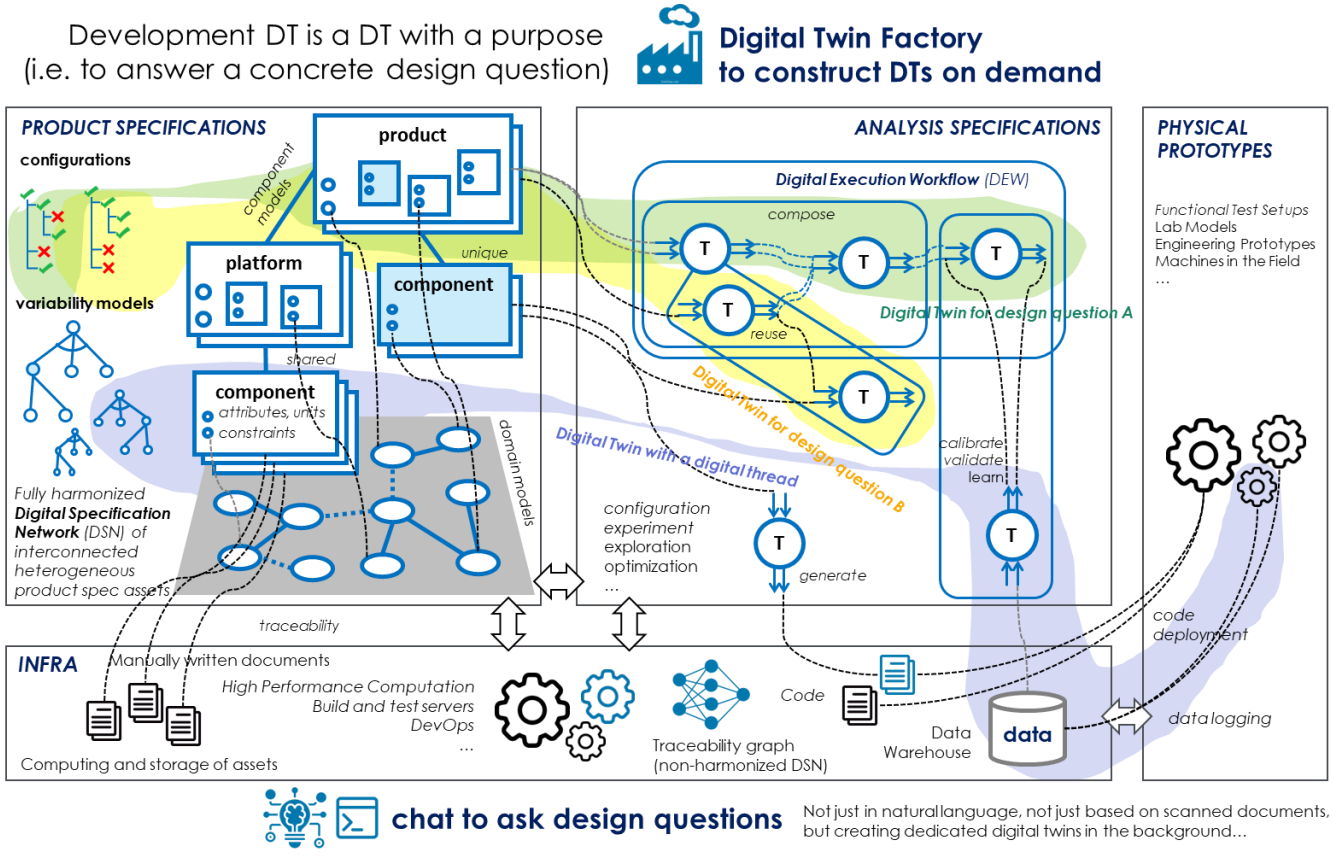

Figure 2: The Digital Twin Factory

level (where a component is a multi-disciplinary subsystem of the product and can have several specifications from various domains linked to it as part of the component specification). This allows for a model of the system design that can be evolved from the beginning through all the lifecycle phases of a product. Investigations into how much of the information which is normally described in documents can be captured using component models with containment, abstract components and inheritance, attributes, units, and constraints (well-proven software system decomposition and reuse constructs) demonstrates a very high coverage on functional requirements and system requirements [26] for a complex system like a production print engine.

Feature models allow for explicit capturing of *intended variability*, where a configuration can describe a specific variant. The top-level specification of (and with proper metadata also a domain model linked to) a component can be extended with variability points, which can be linked to the feature model in order to configure the component. This enables full description of product lines and product platforms with deep traceability and analyzability.

With variability-enabled component models as a backbone to organize DSNs, it becomes possible to treat a set of heterogeneous specifications (grouped under a component) as a "library", i.e. to extend a DSN with another DSN, to import a DSN into a DSN, and thus to specify complete coherent contexts, such as a product (subsystem), a platform (subsystem), or another complex, reusable set of specification assets that can be used in a product or platform. The nodes in a DSN and their interfaces also enable controlled replacement of tools behind the nodes if nodes don't match an evolving reference architecture for a DSN. Finally, having various nodes over which elements can be transported automatically from node to node, enables a high degree of flexibility with respect to "which node is the authoritative source of information", which can change quite dynamically during various product development phases, which in turn allows people to "stay in the DSN" instead of "escaping to hand-written documents" again.

Another interesting aspect of component models, is that a component can technically have a "binding time" (similar to a variable that is bound with a "let" clause in a functional language) towards a context in which it is used. For example: the default binding in early phases would be design-time binding, where the component is mainly static information in the form of the component model and all DSN nodes associated with the component. If the component would have a binding to simulation-time, it could have an instance (similar to a class instance in an object-oriented programming language), keeping state that could be used as part of the DT's computational tools.

3.2.2 Digital Execution Workflow (DEW). Using the earlier described computation-output-to-input links, we can (and again, don't necessarily must) connect the computational part of tools in a DEW (as depicted in the *analysis specifications* box in Figure 2 – previous page). We can reuse the same tool over various DEWs and we can use a DEW in a DEW, making a DEW effectively behave like a bigger tool composed of tools. From logical perspective, a DEW captures an on-demand toolchain for a specific (possibly reusable) use case. This allows for sustainable and controlled road mapping of tool development and shielding of off-the-shelf tools from various specialist use cases that don't work well in a specific context. For example, a complex off-the-shelf tool which requires measurement data of a physical process as well as some (possibly transformed) attribute (or set of attributes) of a product specification as input can be composed as a single DEW. From a product developer perspective, a tool can be used to answer a (usually smaller) question about a potential design choice and a DEW can answer bigger questions by composing tools and other DEWs. DEWs in themselves are only specifications of execution workflows. A mechanism is needed to execute DEWs: an execution engine. In its most general case, a DEW's execution can be distributed over several execution engines (each of which deals best with a specific case, such as (off-the-shelf) co-simulation for a number of domains, run-time optimization of a design, a dedicated multi-paradigm simulation system, etc.) in several execution platforms (in-house high-performance compute infrastructure, cloud computation infrastructure, or specialized infrastructure such as GPU/TPU or quantum computing).

3.2.3 Infrastructure. In order to manage and execute DSNs and DEWs, certain infrastructure is needed (as depicted in the *infra* box in Figure 2 – previous page), such as:

- *high-performance computation infrastructure*: the infrastructure for executing DEWs; many DEWs which execute non-trivial computations, need much more than a local machine to run within a reasonable amount of time
- *digital specification and tool asset storage and versioning*: the various digital assets are stored in various databases or versioning systems; depending on the required longevity for digital assets, it may be strongly recommended to use reliable systems, such as ALM or PLM
- *data warehouse/data lake*: all data assets (physical logging, measurements, simulation logging), which can (if properly combined) serve as digital shadows [1], need to be available for correlation with specifications or computations, for analysis, and for general reuse

3.2.4 From DSN and DEW to the Digital Twin Factory (DTF). The remarkable thing about the decomposition mechanism in DSNs, is that they are not only suitable for product specifications, but also for DT specifications. Considering that a DEW, or any other specifications such as design space explorations on top of a DSN and a (set of) DEW(s), are also just nodes in a DSN and can be extended with variability points, the same reuse mechanisms that exist for product specifications also work for DT specifications. Especially for devDTs (which have a strong relation with a product that is being developed), the organization

of the digital assets in DSN and DEW has a lot of the information available to simply instantiate a (parametric, if so desired), DT on demand in order to perform a design space exploration/optimization or answer other product-development-related questions. Figure 2 (previous page) shows an example of three different DTs composed and instantiated in such a manner:

- the green overlay (starting at the most left configuration in the *product specifications* box) represents the digital assets required to configure and instantiate a DT without a digital thread: a specific product variant that is used to configure the product specifications from a 150% model [27] to a 100% model and then, based on that specific variant, feeding a DEW with settings from the product variant, which, after execution will either present results in the usual visualizations provided by the separate tools, or be further processed by another tool for higher-level visualizations or even use the product specifications to interpret the results deeper in the context of the DT's purpose (the question originally considered by the product developer)
- the yellow overlay (starting at the mostright configuration in the *product specifications* box) illustrates the same as the green overlay, but with a different product variant and different tools (due to a purpose, i.e. a different question considered by the product developer)
- the purple overlay (which is the last one left) represents the digital assets required to configure and instantiate a DT with a digital thread: logging data from a physical twin is stored in the data warehouse and one of the tools in the DEW is a data analysis computation (data processing script, machine learning, or other type of data analysis computation) whose outputs can then be fed into other computations in the DEW.

The stakeholder roles of a DTF are:

1. product developers: benefit most from the high-level connections that can be made with a DSN or from DSN to DEW, because the DTs instantiated from the DTF give them ways to explore various product design choices and come up with the most optimal design; this approach becomes even more powerful if devDTs are combined with design space exploration and multi-objective optimization techniques.
2. (computational) tool developers: should provide connectors with their tools (such as the earlier mentioned "injectors" and "extractors") in order for them to be interoperable with other tools meant for use in a DTF ecosystem; these stakeholders can also use the DTF as an infrastructure for validation, calibration, and learning of their tools, e.g. revalidating a tool's computational model upon extending the feature set or fixing bugs (similar to unit testing of software, but on a computational-model-validation-level).
3. DT integrators: benefit most from the DTF concepts as a specialized language to drive integration of DTs in the organization; also benefit from the DTF specification mechanisms, such as DSN, DEW, and design space explorations and optimizations, which make it possible to analyze DT structures and check progress on various separate digital assets through incremental integration.

A few other topics (such as code generation from product specifications, semantic research document search, and question-based DT instantiation using e.g. LLM technologies) are shown in Figure 2 (two pages back), but not further discussed in this text.

3.3 Technological Enablers

Realization of the vision and DTF concepts can only be made feasible with the right technological ingredients, which we will touch upon here on a high level:

3.3.1 Domain-specific Languages (DSLs). This technology forms the fundamentals for connecting nodes in a DSN (or in other words: for encoding and organizing an organization's subject matter) in a feasible way. In its most naïve form, the connections between the nodes in a DSN are hand-written scripts that query elements from one specification and write them into another. This approach is perfectly fine to start with, but as an organization extends its ecosystem and DSNs start growing in complexity, it may become unscalable. A more systematic approach (enabling much more reuse and automation) would be to capture the metadata for the elements one wants to exchange from/to a specification. Again, a naïve implementation of this idea would be to make a glossary of important subject matter terms of the product's or the organization's domain and then match keywords. Such metadata could help with traceability, but not with harmonization between nodes in a DSN. A more advanced implementation would be to use a decomposable glossary (where a definition can use other terms that can be further defined hierarchically), or even better an ontology with more explicit relations between the concepts of the subject matter. The same purpose can be achieved using DSLs built in a language workbench with an object grammar, such as JetBrains MetaProgrammingSystem (MPS) [2], but in a much more refined way: a DSL encoded in MPS can not only hold the concepts and their relations but also other language aspects, including various language composition and extension mechanisms, which enables deep integration of subject matter from various domains in a maintainable way. Specifying and using DSLs in this way, where the Abstract Syntax Tree (AST) – representing the essential information of a specification – is central used to be unique to JetBrains MPS, but more effort is being done to spread this paradigm beyond MPS, which is demonstrated by efforts such as Modelix [8] and LionWeb [9], where AST-based exchange of

elements between specifications (as opposed to file-based exchange between specifications) is a first-class citizen.

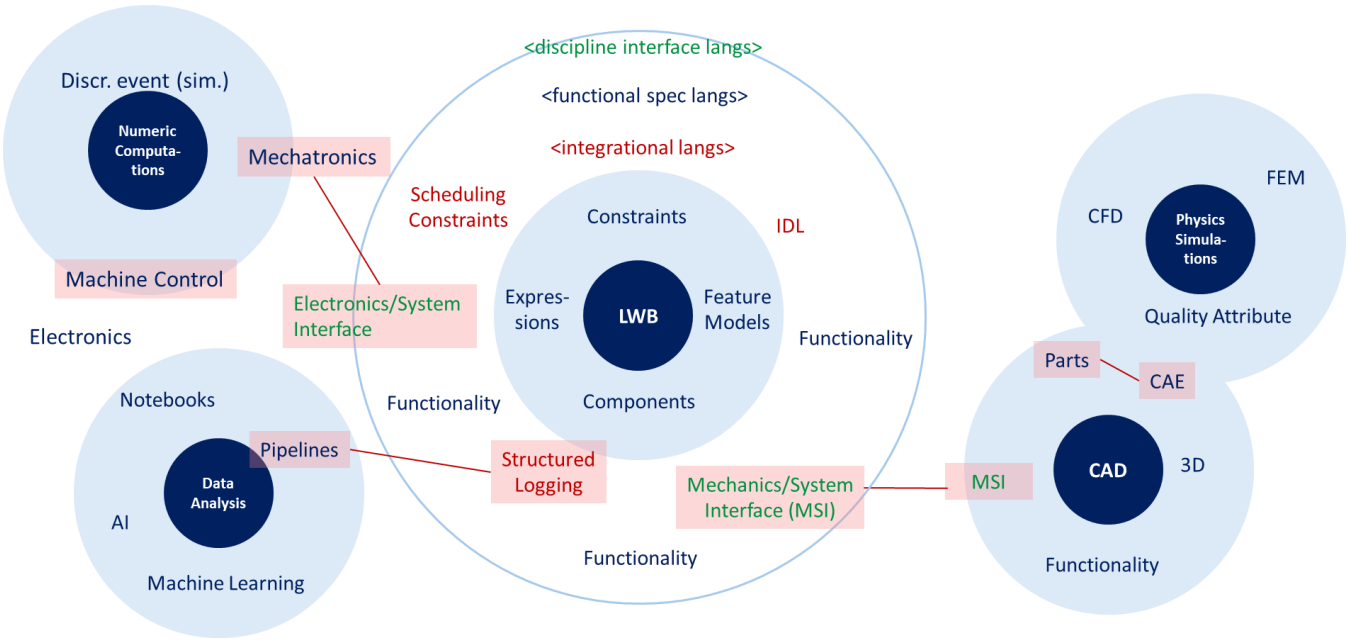

Figure 3: JetBrains MPS language workbench and integrational DSL cores: backbone to organize an organization's subject matter

3.3.2 Integrational DSL cores. Organizing the various subject matter domains from tools (whether they are off-the-shelf or in-house) can be achieved systematically by using integrational DSLs, such as the earlier mentioned component models, feature models, but also at a deeper level of expressions and constraints. The Integrated Environment for Technical Software Systems Specification (IETS3) [4], which is built in MPS, provides important ingredients, such as an out-of-the-box variant management framework that can be connected to and configure models written in MPS-based DSLs for any domain [28,29], a featureful, extensible functional language (KernelF) that serves as a base for building and integrating complex MPS-based DSLs with expressions [5,6,7], and a physical units language with customizable units that are part of an attribute or constraint expression's type. Figure 3 shows how such DSLs (together with a component modeling language) can serve as a backbone to organize and integrate the subject matter through various domains of an organization. For example: the mechanics/software discipline interface language can be used to expose relevant parameters of a CAD model on system-level in a component model (e.g. as component attribute).

3.3.2 Computing platform. High-performance compute infrastructure is an important prerequisite to be able to run non-trivial DEWs. Although computation capacity can be bought online, it may be prudent to invest into building an in-house computation platform, not only for keeping confidential information inside the organization's perimeter, but also for cost considerations. A rule of thumb is that operational (thus recurring) computation capacity should be covered in-house, while incidental spike capacity can be rented externally. In addition to one or more computation platforms, execution engines are needed. Various off-the-shelf workflow engines exist for these purposes, but the mapping from a DEW specification to one or more execution engines that run on one or more execution platforms, would be (at least nowadays) part of a DTF's in-house implementation.

3.3.2 DevOps and Cloud Technologies. Various off-the-shelf ALM and PLM systems are available to manage digital assets. Typically, tools and DTF infrastructure tools benefit a lot from DevOps infrastructure to guard quality and validity through testing and automatic (re-)validation. Cloud technologies, both front-end and back-end are essential to scale the use of a growing set of complex combinations of various tools. Back-end technologies enable scalable deployment of various (collaborative) specification (micro-)services, while front-end technologies allow for user-level integration of many complex services into conceptually clear and unified user interfaces.

3.3.2 Data + modeling strategy. In order for the data in the data warehouse to be reusable for various purposes (such as data analysis, computational model validation, and preparation for input to computations, the production of data must be precisely specified for structured logging [30,31,32] and preferably connected to the product specifications. This makes it possible to make important mappings for data logging (with the product specifications as the leading source) and for data consumption by devDTs [17].

4 Holistic Organizational Embedding

The earlier described technological and conceptual base for a DTF is a good starting point to digitally transform complex product development in a systematic way, but it doesn't guarantee actual "success on the work floor". Additional variables, such as expertise of people, way of working, explicit roles and responsibility in the organization, and a process around asset management (physical and digital together) factor into the equation. All are centered around knowledge/expertise development and applied in product development.

4.1 Physical and digital assets

When considering physical and digital assets, there are several relevant dimensions:

I. Product contents scope
   a. Discipline (i.e. "mono-disciplinary")
   b. Product functionality (one or more disciplines)
   c. Product quality attribute (cuts across one or more functionalities)
   d. Product (overarches one or more quality attributes)
   e. Platform (stretches over one or more products)

II. Organizational
   a. Competences (disciplines, departments) and Platform: captures and maintains knowledge and typically has its own organization-specific set of roles. Focuses on reuse and supporting competences (knowledge – IP and generic, process, roles) across projects, endless; owns physical and digital assets.
   b. Product development: uses knowledge to develop products, places knowledge in a specific context. Solves project needs.

III. Asset types
   a. On one side we have the knowledge/expertise development, involving infrastructure, tools, and model libraries (reusable assets)
   b. On the other side we have product development involving models and configurations (product-specific assets)

IV. Domain-specificity: Generic/fundamental ↔ Quality attributes and functionality ↔ Organization-specific ("core IP")

V. Ownership/development
   a. (Commercial) off-the-shelf ↔ in-house

VI. Expertise
   a. Expertise levels: novice … expert [33]
   b. Expertise areas: application, fundamental, digital literacy and knowledge of working with physical assets
   c. Proto-roles: method owner, user, tool smith (minimum set needed for successful digital asset introduction and sustaining)

The complexity of (a holistic view on a collection of) assets is like a cartesian product along (at least) each of the above described dimensions (imagine a combination of sliders, one or more for each dimension): I × II × III × IV × V × VIa × VIb × VIc. Making no choice in a dimension is also a choice, and in our experience every choice leads to consequences.

4.2 Digitally Transformed Development Processes and Organization

This kind of digital transformation involves moving away from the "old" physical-prototype-based way of working ("build a product demonstrator and see if it works, how it behaves, what its capabilities are") to the new DTF-based way of working ("first think about design questions and risks, then think about how you can answer the questions, and finally answer them"). This opens up new options to answer product-development-related questions: in addition to using integrated testbenches or fully integrated product prototypes, on-demand digital twins (or hybrid physical-digital setups) can be used. This not only enables smarter use of expensive physical assets such as partial testbenches or laboratory measurement setups (optimally for solely mapping out unknown-unknown or known-unknown knowledge, while handling known-knowns and unknown-knowns with digital assets), but also opens up the space of possibilities to answer questions and improves development lead-time by front-loading many integration-related

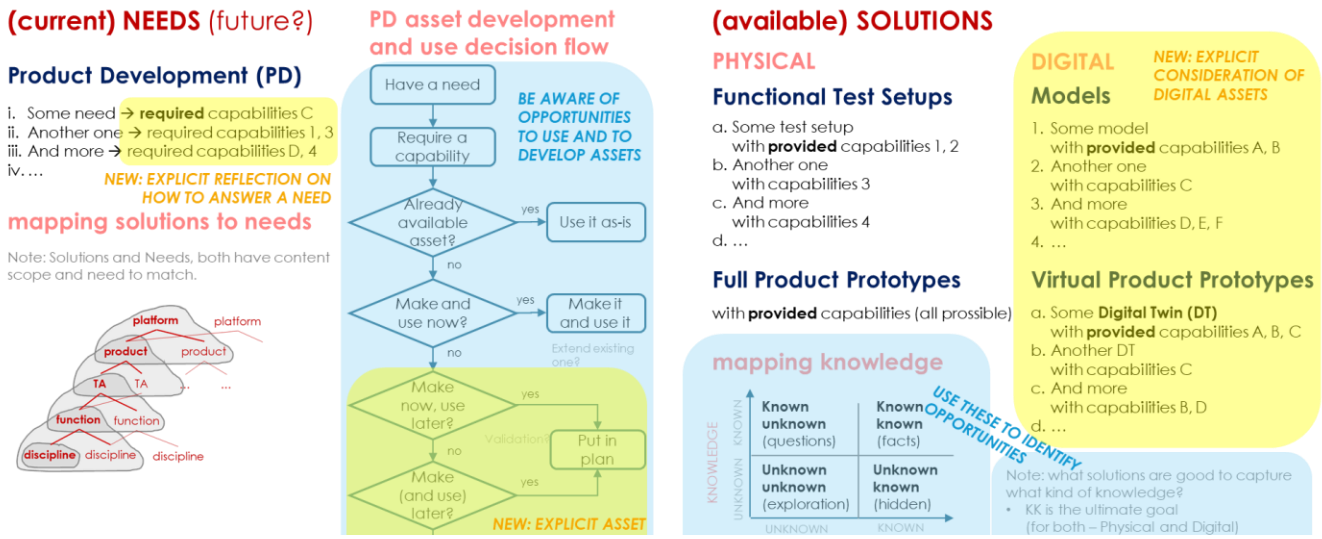

Figure 4: Applying the DTF-based approach in product development

questions to digital assets (which can be made operational much quicker than physical assets).

The new, proposed way of working not only has advantages, but also poses new challenges:

4.2.1 How to develop new assets. Development of new or significant adaptation of existing digital assets (especially over the lifetime of a single product development phase) is not always straightforward. For example, in an early development phase of a product, you already need tools with validated computational models but one or more of the needed tools may not be developed yet; however, it can happen that once a needed tool is made, there is no significant value for the product in question, because important architectural and design decisions may have already been made for that product. This challenge not only requires a new mindset and understanding of product developers on what to expect from digital assets (see Figure 4, second column), but also a road mapping of asset and competence development across products or even entire product portfolios.

4.2.2 How to apply in product development. Figure 5 visualizes the method for applying the DTF-based way of working in such a manner, that development steps maximally leverage any available just-in-time knowledge, front-loading design option investigations as much as possible. This method involves a stepwise, incremental process:

- The starting point is an architectural or design question (left top blue text in capitals).
- If the question can be answered in the DSN of the product (e.g. through simple "Excel-like" computations on the component model or some lightweight computations in other specification assets), the answer is already obtained (possibly after first making new or updating existing product specifications) and a next question is formulated.
- Should the question be more involved, the next step is to try to answer the question using analysis models (i.e. in an existing DEW). In case no ready-to-use DEW or design space exploration/optimization specification exists for the question, the DT integrator(s) is/are involved to create a DEW.
- In case the question cannot be answered using a DT (DSN, DEW, or a mix of both), it may be possible to answer the question through querying and analyzing data from the data warehouse (possibly needing to develop a new data analysis). A check should be made whether the data from the data warehouse can be used to extend a DEW so it could answer this question in the future.
- As a last resort, a (set of) physical setup(s) can be used to do a physical experiment in order to answer the question. In line with data + models strategy, any logging or measurement data from a physical setup must be collected in the data warehouse.

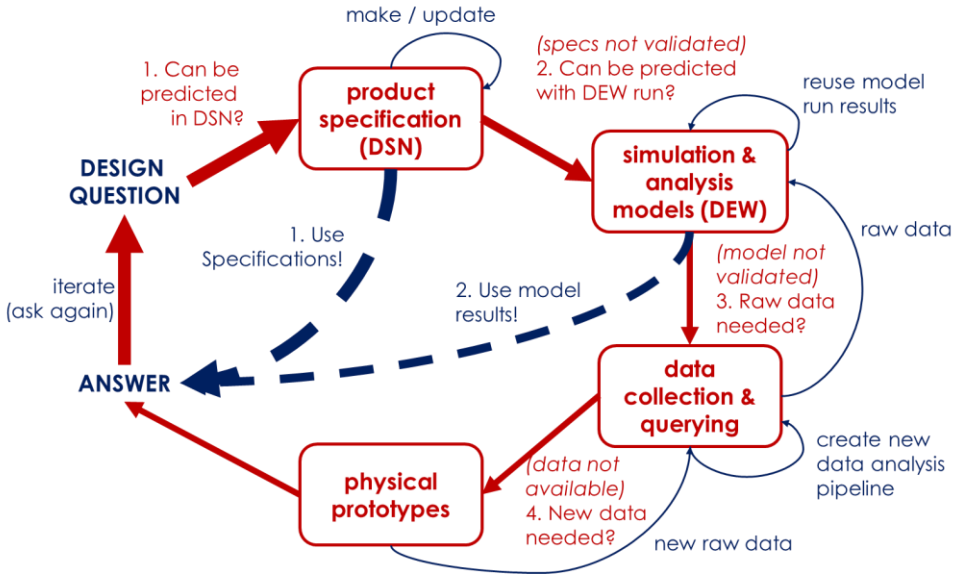

Figure 5: A stepwise, incremental process to using the DTF-based way of working

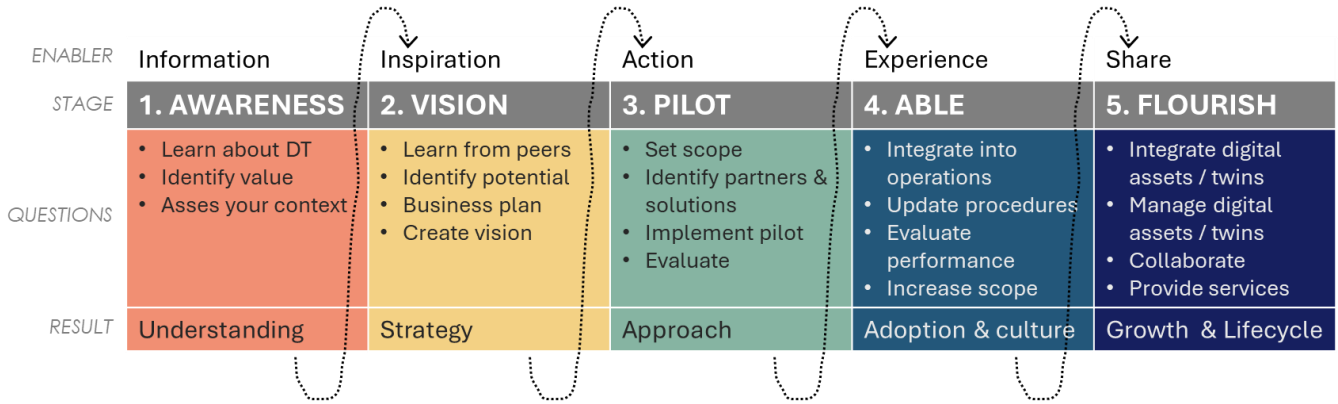

Figure 6: Stages of digital transformation, using DTs as a core conceptual carrier (inspired by "2nd TNO Digital Twinning Round Table 2022" meeting, 16 Nov 2022)

If this process is followed, then every R&D activity in product development contributes to an ever growing corpus of knowledge which is explorable, learnable, and virtualizable. A note must be made about physical assets: these must be developed with reusability over various products or product portfolios in mind. Again, the above described process, combined with the DTF-based way of working and infrastructure, helps a lot in designing reusable physical assets: if (sets of) digital assets are linked up (threaded) with physical assets through the DT abstraction, as defined in the DTF proposal, there can be an organization-wide insight into physical asset needs, physical asset waste, and prevention of accidental dismantling of still needed physical assets.

5 Current State

We believe that we currently have a solid base for a DTF-based way of working and have started to sustainably introduce the technological concepts and method to our organization (starting with the R&D department). While doing so, we are validating tools and methods in this conceptual framework in real-world use cases, which gives us information for further developing the tools, methods, and general approach.

This sounds like a very big step, but we reuse, leverage, and bring together all the existing infrastructure and assets our organization has developed over the years, sustainably spreading the development and maintenance of the tools and infrastructure needed for this approach.

We, as well as our organization, have a significant history in this field [10,11,14,15,16] on various topics such as software development, system development, language development, and (multi-disciplinary) modeling and model interoperability, so to us the DTF is not a surprise or "Aha-moment", but rather a logical continuation and consolidation of many lessons we learnt into a conceptual framework (i.e. it is an evolution rather than a revolution).

Although we have developed and validated digital assets and various building blocks of the DTF in smaller contexts (stage 4 in Figure 6), the full-scale application of the DTF-based way of working for an entire complex multi-disciplinary cyber-physical product is currently in a first-application phase (stage 3 in Figure 6), as part of our company's digital transformation strategy, where in-the-large validation for our incremental, holistic approach is performed.

6 Conclusions and Possible Extensions

We believe that we have created a holistic, scalable, and maintainable approach to digitally enhance and deeply transform our product development activities, with obvious benefits such as shorter time-to-market, customization, and managing the growing complexity of our products.

Our company's trust in using such a sustainable, incremental development approach to physical/digital assets and competences strategically gives us confidence to claim that we have a feasible starting point for a holistic approach to engineering reusable, configurable, and composable DTs for multi-disciplinary, cyber-physical product development purposes.

Since the concepts described in this text are only a starting point for an exciting journey, we invite the DT community to collaborate on this approach. Concrete examples of topics that may be areas of interest for us:

- A solid semantic base for interaction between DSNs, component models, and feature models, as well as for DEWs and all their intricacies
- (Semi-)automatic metadata scanning from digital assets and harmonization of digital assets
- Semantics for component binding times (e.g. design-time, simulation-time, production-time, in-the-field)
- Semantic search and intelligent question-based DT instantiation

REFERENCES

[1] Eramo, R., Bordeleau, F., Combemale, B., van den Brand, M., Wimmer, M., & Wortmann, A. (2022). Conceptualizing Digital Twins. *IEEE Software, 39*(2), 39-46. https://doi.org/10.1109/MS.2021.3130755

[2] JetBrains MPS. 2024. Meta Programming System: Create your own domain-specific language. Retrieved June 2024 from https://www.jetbrains.com/mps/

[3] Antonio Bucchiarone et al (Ed.), 2021. Domain-Specific Languages in Practice: with JetBrains MPS. Springer Cham. https://doi.org/10.1007/978-3-030-73758-0

[4] IETS3. 2024. Integrated Environment for Technical Software Systems Specification. Retrieved June 2024 from https://github.com/iets3

[5] Voelter, M. (2018). The Design, Evolution, and Use of KernelF. In: Rensink, A., Sánchez Cuadrado, J. (eds) Theory and Practice of Model Transformation. ICMT 2018. Lecture Notes in Computer Science(), vol 10888. Springer, Cham. https://doi.org/10.1007/978-3-319-93317-7_1

[6] Voelter, M. (2017). KernelF-an Embeddable and Extensible Functional Language.

[7] KernelF Introduction. 2024. Mbeddr documentation. Retrieved June 2024 from http://mbeddr.com/mps-platform-docs/languages/kernelf/kernelf/

[8] Modelix. 2024. A collaborative and scalable open source platform for domain-specific models on the web and in the cloud. Retrieved June 2024 from https://modelix.org/

[9] LionWeb. 2024. Language Interfaces on the Web. Retrieved June 2024 from https://lionweb.io/

[10] Hristina Moneva talk at Bits & Chips. 2023. Retrieved June 2024 from https://events.bits-chips.nl/past-events/bitschips-event-2023/index.html#moneva

[11] Eugen Schindler. 2021. Video. In Fifth talk in EDT.Community online seminar series on Foundations and Engineering of Digital Twins. Retrieved June 2024 from https://www.youtube.com/watch?v=gocYGyl-6TU

[12] OPC Foundation. 2024. Introduction to the Asset Administration Shell. Retrieved June 2024 from https://reference.opcfoundation.org/I4AAS/v100/docs/4.1

[13] Andreas Wortmann. 2024. Digital Twin Definitions. Retrieved June 2024 from https://awortmann.github.io/research/digital_twin_definitions/

[14] Moneva, H., Hamberg, R., & Punter, T. (2012). A Design Framework for Model-based Development of Complex Systems.

[15] Erdweg, S. *et al.* (2013). The State of the Art in Language Workbenches. In: Erwig, M., Paige, R.F., Van Wyk, E. (eds) Software Language Engineering. SLE 2013. Lecture Notes in Computer Science, vol 8225. Springer, Cham. https://doi.org/10.1007/978-3-319-02654-1_11

[16] Erdweg, S., Storm, T.V., Völter, M., Tratt, L., Bosman, R., Cook, W.R., Gerritsen, A., Hulshout, A., Kelly, S., Loh, A., Konat, G.D., Molina, P.J., Palatnik, M., Pohjonen, R., Schindler, E., Schindler, K., Solmi, R., Vergu, V.A., Visser, E., Vlist, K.V., Wachsmuth, G., & Woning, J.V. (2015). Evaluating and comparing language workbenches: Existing results and benchmarks for the future. Comput. Lang. Syst. Struct., 44, 24-47.

[17] Tan, H. (2019). *Integration of modeling and data science*. Technische Universiteit Eindhoven.

[18] M. Dalibor, N. Jansen, B. Rumpe, D. Schmalzing, L. Wachtmeister, M. Wimmer, A. Wortmann: A Cross-Domain Systematic Mapping Study on Software Engineering for Digital Twins. In: Journal of Systems and Software, 111361,

[19] Andreas Wortmann. 2024. Digital Twins overview page. Retrieved June 2024 from https://awortmann.github.io/research/digital-twins/

[20] Grieves, M. and J. Vickers, Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems, in Trans-Disciplinary Perspectives on System Complexity, F.-J. Kahlen, S. Flumerfelt, and A. Alves, Editors. 2016, Springer: Switzerland. p. 85-114

[21] Wikipedia. 2024. Digital Twin article. Retrieved June 2024 from https://en.wikipedia.org/wiki/Digital_twin

[22] Wikipedia. 2024. System article. Retrieved June 2024 from https://en.wikipedia.org/wiki/System

[23] Wikipedia. 2024. Turtles all the way down article. Retrieved June 2024 from https://en.wikipedia.org/wiki/Turtles_all_the_way_down

[24] Fuller, A., Fan, Z., Day, C., Barlow, C., "Digital Twin: Enabling Technologies, Challenges and Open Research," IEEE Access, vol. 8, pp. 108952-108971, 2020

[25] Grieves, M., Vickers, J., "Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems," Transdisciplinary Perspectives on Complex Systems, Springer, 2017, pp. 85-113

[26] Akhter, S. (2023). Componentization of Product Knowledge of Production Print Systems: Case Study: Fixation. Technische Universiteit Eindhoven.

[27] Rabiser, R., Grünbacher, P., & Dhungana, D. (2007). Supporting Product Derivation by Adapting and Augmenting Variability Models. In Software Product Lines, 11th International Conference, SPLC 2007, Kyoto, Japan, September 10-14, 2007, Proceedings (pp. 141-150). IEEE Computer Society. https://doi.org/10.1109/SPLINE.2007.22

[28] Klaus Birken (itemis). 2023. Adding complex cross-cutting functionality to (nearly) any language – mechanisms and challenges. Retrieved June 2024 from https://youtrack.jetbrains.com/articles/MPS-A-216170508/JetBrains-MPS-Community-Meetup-2023-Videos-and-Slides.

[29] Klaus Birken (itemis). 2023. Adding complex cross-cutting functionality to (nearly) any language – mechanisms and challenges. Video. (11 May 2023). Retrieved June 2024 from https://www.youtube.com/watch?v=eAqh819L_Vw

[30] Legeza, V., Golubtsov, A., & Beyer, B. (2019). Structured logging: Crafting useful message content. login;, 44(2). Retrieved June 2024 from https://research.google/pubs/pub47257/

[31] Kratzke, N. (2022). Cloud-native observability: The many-faceted benefits of structured and unified logging—A multi-case study. Future Internet, 14(10), 274. https://doi.org/10.3390/fi14100274

[32] Nouhoum Traore. 2024. Structured Logs: Best Practices for Effective Management. Retrieved June 2024 from https://medium.com/@nouhoum/structured-logs-best-practices-for-effective-management-0f8dfd3ab31c

[33] Dreyfus, S. E., & Dreyfus, H. L. (1980). A five-stage model of the mental activities involved in directed skill acquisition. University of California, Berkeley, Operations Research Center.