Killing the PLM Monolith –

the Emergence of cloud-native System Lifecycle Management (SysLM)

Authors:

Fraunhofer IESE
M.Sc. Oliver Bleisinger
M.Sc. Thomas Psota

Fraunhofer IAO
Dipl.-Ing. Jonathan Masior

University of Applied Sciences Esslingen
Dr.-Ing. Michael Pfenning

Accenture GmbH
M.Sc. Arnold Roth

Koneksys LLC
Dr.-Ing. Axel Reichwein

Mercedes-Benz AG
Dr.-Ing. Yousef Hooshmand

Contact Software GmbH
Dipl.-Wirtsch.-Ing. Christian Muggeo

xPLM Solution GmbH
Dipl.-Wirtsch.-Ing. Marc Hutsch

Version 1.0
May 30, 2022

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer-Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr.-Ing. Peter Liggesmeyer

Fraunhofer-Platz 1
67663 Kaiserslautern
Germany

Abstract

The rapid increase of complexity in modern products introduces numerous challenges for manufacturing companies around the globe. *Model-Based Systems Engineering* (MBSE) is seen as the best choice to handle this huge increase in complexity and as one cornerstone to realize the so called *Digital Thread.* In many industries MBSE constitutes an additional engineering discipline that needs to be established in organizations and that comes with sophisticated digital models which have not been used before. In fact, it also needs to be accompanied with the right set of tools, processes and methods – and with the right open, scalable and flexible IT-architecture to make it reality. On the one side MBSE needs mature Lifecycle and Configuration Management – on the other hand it must live within an open IT-environment satisfying the need to have the right set of engineering data always just a click away.

This paper shows why legacy monolithic PLM tools cannot support the introduction of MBSE and are currently preventing the implementation of the *Digital Thread* vision. Instead, it postulates a modern cloud-native Web architecture based on Microservices and Linked Data that allows companies to introduce MBSE on the large scale and helps to avoid the establishment of just yet another silo. Only with Linked Data and a modern open Web architecture MBSE can unfold its full potential and is able to find its way into the daily work of engineers.

Keywords: MBSE, PLM, Microservices, Linked Data, Digital Thread, Monolith, Cloud-native, Low Code Platform, Engineering 4.0, Industrie 4.0, Industry 4.0, DevOps, SaaS, Semantic Web, Production Level 4.

Table of Contents

1 Introduction

With the start of the third decade of the 21st century we are in the midst of realizing the next industrial revolution. The most complex system ever created by human beings – the internet – is still growing and conquering new grounds where it not just connects people and businesses, enables collaboration and the sharing of ideas and knowledge. It is now also connecting machines. This creates a variety of new business opportunities for companies around the globe that are willing to adapt their way of thinking. In fact, it enables people with ideas and a minimal set of programming skills to be entrepreneurs as it tremendously lowers market entry costs.

For long established companies the Internet of Things and Services (IoTS) is also seen as a driving factor to face current challenges. Be it the ever increasing need for highly customer-specific products (lot-size one) and therefore the ability to master the craft of variant management. Be it data-driven optimization in maintenance of large-scale manufacturing systems (Predictive Maintenance). Or be it the ever wished for reduction of Time to Market – the time necessary to introduce a new product.

All this needs one enabler that is not within maintenance processes, IoT-platforms, IoT-ready sensors in a factory or a Web portal that allows customers to configure a new product. This necessary prerequisite is within the engineering phase. It is the way models are created, it is the way engineering data is made accessible for later lifecycle phases and it is about the degree of formalization achieved when describing the product under development. Model-Based Systems Engineering (MBSE) is aiming to fill this gap. Its ultimate goal is to formalize the Systems Engineering process by using a formal domain model and by doing so replacing document-driven approaches. This of course goes to the heart and brain of every industrial company – it urges us to rethink the way a product is developed and illustrates the significant shortcomings of existing PLM tools.

The Industry 4.0 initiative that has been started quite some time ago, did see engineering and development as one prominent field of action (see [1]). A little after that the term Engineering 4.0 (see [2]) came up to underline this fact even more. Industry 4.0 strongly needs accessible, high quality and formalized engineering data. And most importantly: Accessible means that data must be able to flow smoothly between applications – even if they are not provided by the same tool vendor. Without this openness every implementation of an industrial IoTS use case will fall far too short and will never reach an enterprise scale. Data Analytics, Artificial Intelligence are truly promising fields of action for nearly every enterprise – but to really make use of it we need to get data out of its cage, out of the many silos that have been created over the past decades [3].

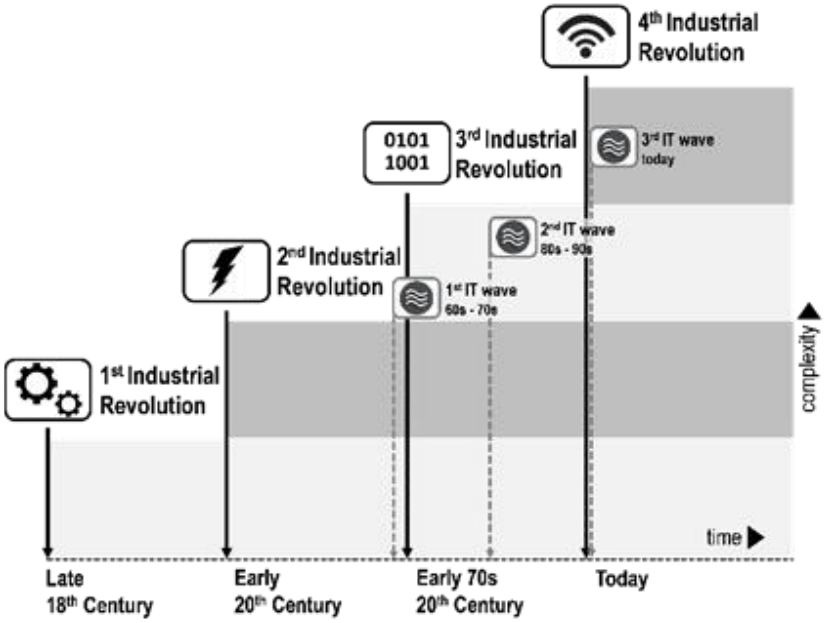


Figure 1 - Industrial Revolutions and IT waves, taken from [4]

Besides the fact that we are in the middle of the 4th industrial revolution becoming a reality, we do also see the third big wave of IT technology currently sloping over from Internet companies to Enterprise IT (see Fig.1). Porter and Heppelmann described this 3rd wave in their article for the Harvard Business Review in 2014 "How Smart Connected Products are transforming Competition" (see [5]). They stated that this wave is about how IT is revolutionizing products, what they ignore is the fact that this 3rd wave also helped to create large scale cloud environments. With the establishment of those environments cloud-native technologies and architectural principles have been invented and validated over years. They are ready to use now in the engineering domain.

This architectural transformation finds its ideals in Amazon, Facebook and Google. But also wants to adapt those lessons learned to the specific needs of the engineering domain. It can be said, that the level of federation of data and services will increase and centralized, monolithic architectures will get more and more obsolete. It started with Mainframe architectures in the 60's and 70's, transformed to Client-Server Architectures and lately moved forward to Microservices. The problem that many PLM vendors face is that just adding a Web client to their monolithic IT system only modernizes the UI, it is not helpful with making their system's internals future ready.

Monolithic Architectures were born in the 90's where the database of choice has always been a relational one, where the WWW has been around only a couple of years and Semantic Web technology has not been thought of. The classical monolithic architecture has been *database – application – rich client*. With all data in one Relational Database Management System (RDBMS) with its transactions based on the well-known ACID[1] principle. In the meantime, Web clients for PLM tools

[1] atomicity, consistency, isolation and durability (ACID)

have of course been introduced, but as stated before, this did not change the underlying architecture.

After nearly 30 years of developing more and more functionality on top of this monolithic architecture, PLM vendors are now struggling with changing it and moving forward with a more modern approach. One of the problems they face is upgrading their customers to newer releases that offer enhanced functionality. Another one is the lack of consistent architecture due to the fact that no single individual oversees the complete architecture and especially the data model anymore – leading to the need to communicate between a growing number of individuals only overseeing portions of the whole and fragments of the dependencies in the PLM tool. The legacy that those tools carry around with them is too complex to master the necessary leap forward. This may also be referred to as "lost in Conway's law"[2] where trying to change a minor thing in its architecture is bringing the house of cards to collapse. To paraphrase Conway: Good organization and good processes lead to good software. If communication and responsibilities in an organization are efficient and well-defined, so is its IT-architecture. This is not just true for organizations using the software, but also for organizations developing it. Every product and every application has its lifecycle and so do monolithic PLM tools – they reached the end of theirs and will be replaced.

Especially for PLM this monolithic architecture came along with a sales strategy aiming to lock customers in (➔ *Vendor Lock-in*) and sell addon applications on top of the monolith that would not be competitive as standalone solutions. As Semantic Web technology has not been around and integration standards such as CORBA[3] failed for many different reasons, this monolithic but integrated approach to PLM has been the dominant strategy for the last years. This slowly but effectively is starting to change with Linked Data, Microservices and the experience that has been gained lately with establishing huge federated cloud environments such as the SAP BTP[4], Amazon Web-Services (AWS), Azure, Alibaba or Google Cloud.

More than a decade ago PDM/PLM vendors followed different philosophies, not just the integrated PLM approach based on a monolith. The alternative approach to that has been the federated approach that allowed to connect best in class engineering tools with a PLM backbone also called "best-in-breed". In that scenario the PLM tool focused strictly on data management and left the authoring part to highly specialized tools that actually were able to solve the underlying engineering problem.

The federated approach is currently experiencing a real renaissance as current Web-based technologies today support federation and integration of heterogeneous tool chains way better. In addition the related but different discussion between Out of the box (OOTB) vs. Customizations is also on the agenda again. Low Code Platforms help tremendously in digitalization projects and will also start disrupting the PLM market, making customization easy, fast and upgradable.

---

[2] Conway's Law states that an organizations chosen IT architecture follows its communication patterns. It can further be interpreted in a way that for good software architecture a well-defined development process is necessary (which is quite common sense).

[3] Common Object Request Broker Architecture (CORBA)

[4] SAP Business Technology Platform, formerly known as SAP Cloud Platform

2 Integrated vs. Federated, OOTB vs. Customizations

The discussion about the best PLM philosophy is as old as the PLM market. The first and biggest PDM implementations have been based on toolboxes (e.g. Metaphase) with a high amount of company specific customizations[5]. That helped companies to design exactly the processes and data models they needed to support their business. The drawback has been the lack of upgradeability and the relatively high costs to maintain such a highly customized PLM environment. Those environments were also federated with best-in-class tools that solved the underlying engineering problems in depth. But on the downside, they lacked a standard integration technology because of the vendors inability and unwillingness to agree on an integration standard.

The next evolution has been OOTB PLM tools, that were designed to contain predefined standard processes. The aim has been to solve the upgrade issue, as data model and process model does not change between implementations – it is only enhanced, not changed. Connecting to external tools has not been part of the concept as specific domain applications were just developed on top of the monolith, so point to point connectors have not been necessary. In retrospective it can be said that the OOTB approach to PLM has at best never been more than a dream – or worse just a sales strategy.

OOTB PLM did not have customization capabilities built in, but had to introduce abilities to configure and customize later on. Because realistically speaking in all PLM projects configuration and certain customizations are always necessary – and therefore those tools ran into the same upgrade issues they were designed to avoid. Figure 2 shows that real implementations of PLM in Aerospace and Defense are usually far behind the existing technology and the promised vision – the situation in other markets such as automotive is not much different. In the same report it is stated that at least a third of all PLM implementations are more than 5 years behind the newest available release. Upgrades of OOTB PLM tools are expensive and therefore where possible avoided.

[5] And many of those implementations exist still today, as they were not able to be migrated to an OOTB PLM tool.

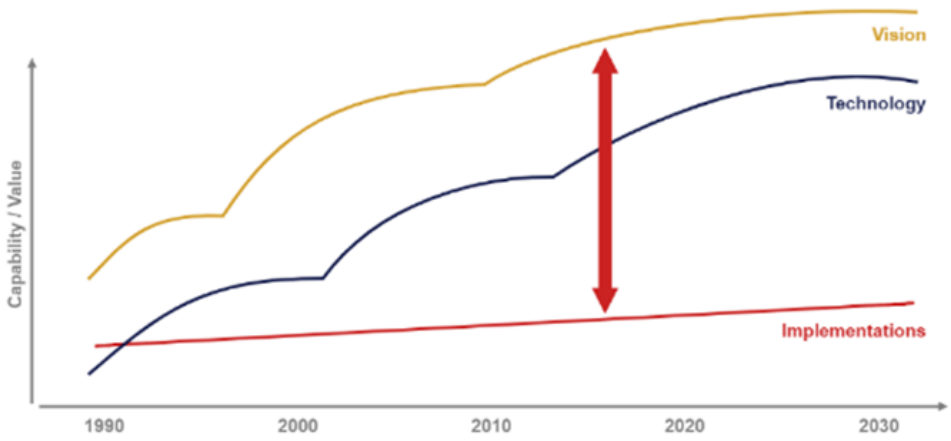


Figure 2 – Real Implementations compared to Vision and Technology in legacy PLM[6]

Good applications come from specializing in a domain – this is also referred to as rule one of the Unix philosophy: "Make each program do one thing well. To do a new job, build a fresh one rather than complicate old programs by adding new features"[7]. PLM monoliths have not been built using this philosophy. Most of the time the applications that run on such an architecture were just superficially solving the underlying engineering problem, not competitive with standalone tools. The incorporation of acquired tools for example took years – but mostly requirements changed in the meantime. This created a situation where the so called "incorporation strategy" was just not keeping up with the actual emergence of market requirements due to the complex nature of such endeavors. Connecting to external tools on the other hand (following an "integration" rather than an "incorporation" strategy) has never had a priority with OOTB PLM. Thus, the field has been left to IT service providers and IT departments leading to a situation with an uncountable number of connector technologies existent and no such thing as a standard or framework has ever been developed and largely adapted.

[6] Picture ©ARAS Corp. Based on a CIMData report "Aerospace & Defense Industry PLM Value Gap Survey", 2013.

[7] The Unix philosophy has been documented by Doug McIlroy in the Bell System Technical Journal from 1978.

3 The Emergence of modern Web Architectures

One fundamental principle of Industry 4.0 is the commitment to solve interoperability issues by learning from the basic principles of the World Wide Web. The Industry IoT Consortium (IIC) made that commitment even stronger when Richard Soley[8] described the foundation of the Industrial Internet as the application of "Internet Thinking". In other words, "use the way of thinking the internet is built on to solve interoperability issues and apply it to other domains", namely the Industry and its Enterprise-IT and OT[9].

The Internet and its architecture as we know it today can obviously be described as non-centralistic. It is not controlled by a central instance. In contrast the Internet can just not be controlled as a whole. It is the obvious role model for a decentralized system where federation of data and services are the goto-approach. And in addition, the Internet itself of course is not static. It is constantly changing, not just by its available services and data, but also by the constant change of available and proven technologies. It is evolving quickly. Learning from it to find solutions for enterprise IT not only means looking at the issue of interoperability and how it is handled, but also looking into architectural patterns and technologies beyond the original HTTP, HTML and URL.

3.1 CAP dilemma

Decentralization and federation of data are the standard in cloud computing. One of its theoretical underlying principles can be identified with the CAP- Theorem[10] illustrated in figure 3. It states that it is impossible for a distributed computer system to provide simultaneously more than two out of the following three guarantees: Consistency (C), Availability (A) and Partition Tolerance (P). All existing PLM tools are basically CA-Systems whereas modern cloud-services like google, Facebook, LinkedIn are AP-systems – guaranteeing availability and partition tolerance, but lacking consistency.

[8] Chairman of the IIC and the Object Management Group (OMG).

[9] Operational Technology

[10] CAP-Theorem or Brewer's Theorem appeared the first time in 1998 and has been presented at the Symposium on Principles for distributed computing in 2000. In 2002 Seth Gilbert and Nancy Lynch of MIT proved Brewers initial conjecture and therefore rendered it a theorem.

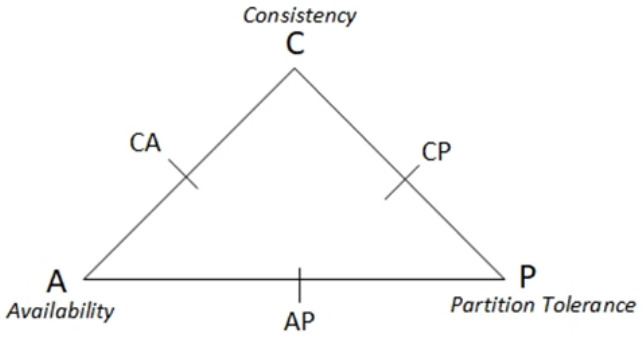


Figure 3 – The CAP- or Brewer's Theorem for distributed computing, see also [6]

This lack in consistency means it cannot be guaranteed that two users asking an AP-system the same question at the same time are always getting the same answer. CA-Systems on the other hand and here especially RDBMS – like all existing PLM-tools – theoretically guarantee consistency by only having a data set exactly once in one database. It may be referred here to the already mentioned ACID-principles of relational databases. This creates the lack of partition tolerance. If that instance is not available, there is no answer.
Partition Tolerance is highly related to the issue of scalability[11]. If you have only one database instance scaling means adding resources (e.g. RAM, CPU, Bandwidth) – this is referred to as vertical scaling (*scale up*). Even today with the most modern hardware components that way of scaling comes to an end with growing user numbers, growing amounts of data and network latency.

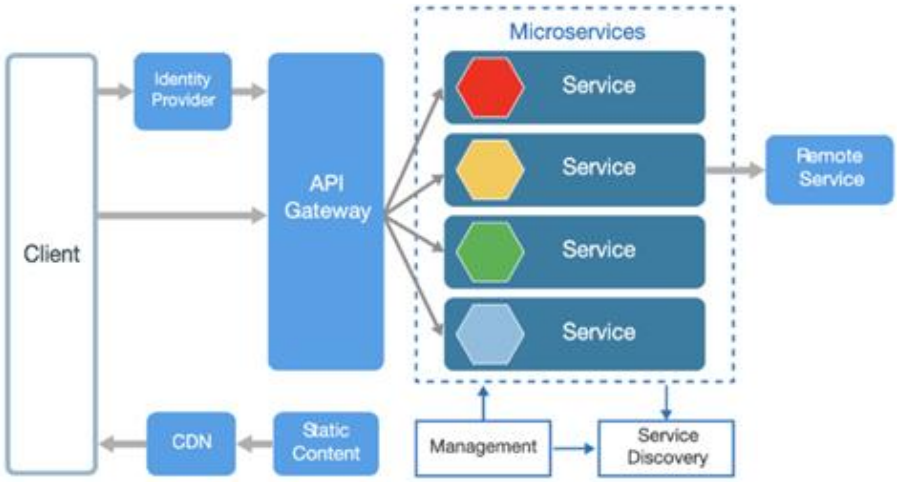


Figure 4 – Generic Cloud Architecture[12]

[11] To learn more about scalability in web architectures see [23].
[12] Courtesy of Cloud Application Architecture Guide Azure

Horizontal scaling (*scale out*) in contrast does not have these hardware restrictions. People often talk about the "Pets vs. Cattle" analogy when asked to describe the difference. In former days hardware servers were treated like pets: they were given a name, if they were sick they were nursed back to health. Should they become unavailable, everyone will notice. In a cattle approach each instance is a virtual machine or container, scaling is done by creating more of them. If the underlying hardware has a defect the instance is moved to another server – all done via automation. No one really cares about the underlying hardware anymore. The cattle approach is one of the core concepts of the DevOps service model (see [7]). As software becomes more complex the one that is developing it, is seen as the best operator of the software. It therefore brings together responsibilities for developing, updating and operating the software – most of the time in a cloud infrastructure. DevOps teams are interdisciplinary teams capable of providing a service via a cloud infrastructure from end to end with a maximum level of team autonomy.

This constitutes a major shift in the business model of PLM vendors. If PLM runs in an DevOps mode in the cloud the vendor becomes responsible for everything needed to provide the service a customer has subscribed to. For a manufacturing company this may mean to outsource a huge portion of its IT operations to the PLM vendor or to invest in a company-wide cloud environment that will be developed further in house. Both scenarios are imaginable and the implementations of both strategies can be identified in the market already: PLM vendors offering SaaS[13] and Managed Services and PLM vendors providing the platform with customers developing additional Application on top in house. This also underlines that PLM may mean different things to a large enterprise and to a SME[14].

In a modern cloud-native approach – often referred to as AP-approach in the CAP terminology – the classical ACID-principles are replaced by the so called BASE[15]-principles. It is to be discussed later on what implications that has on PLM. In general it can be said that eventual consistency shifts the paradigm from "always consistent" as in ACID to "probability to experience an inconsistent state is statistically non-significant" – depending of course on the quality of the used cloud management.

In the PLM community the implementation of cloud computing has only cowardly been started. Nearly all existing implementations in PLM are still on-premise with the drawbacks already described and more to be identified. Most people will know the statement "There is no cloud – it is just someone else's computer". Looking at the available cloud technology in the market, this must be characterized as a tremendous and unfortunate oversimplification of the situation.

In the PLM community most of the times the argument is made that PLM has higher requirements on consistency because of its integrated configuration management capabilities and that's why a PLM-tool could never be based on an AP-system approach. In PLM this is referred to as "Single Source of Truth". This argument is currently challenged by academia and cloud PLM startups.

---

13 Software as a Service

14 Small and Medium-size enterprises

15 **B**asically **A**vailable, **S**oft state, **E**ventual consistency

Also established PLM vendors have started offering PLM in the "cloud" – unfortunately that usually only means hosting the same monolithic architecture outside the customers network and make it available over the Internet. That approach is usually not able to leverage most of the features modern cloud environments are offering (e.g. horizontal scaling) – not because those features are not needed for PLM, but because transitioning from a legacy architecture built to operate on-premise to a cloud-native architecture is a hard and potentially an impossible endeavor, that takes years to transition. The architectural patterns are just too different to reuse most of the legacy technology stack of a classical PLM tool and therefore "cloud-native PLM" requires a fresh start.

3.2 Cloud native Microservices

To better understand what is needed to realize cloud native environments in general, it is necessary to look into a couple of definitions. "Cloud-native" itself, the so called "twelve factors"[16], and the architectural pattern of Microservices. An official definition of cloud native is given by the Cloud Native Computing foundation:

Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, Microservices, immutable infrastructure, and declarative APIs[17] exemplify this approach.
These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

Scalability can be seen as one of the biggest advantages of cloud-native over the classical monoliths. All RDBMS and especially PLM today create many challenges if those environments want to be scaled over the level they originally have been designed for. With the concept of loosely coupled systems running in containers scalability is no longer a challenge, it is built in and can to a large extend be automated (➔*Autoscaling*).

[16] The twelve-factor app is a methodology for building software-as-a-service (SaaS) apps, see https://12factor.net/.

[17] Swagger that uses the OpenAPI standard is one prominent example

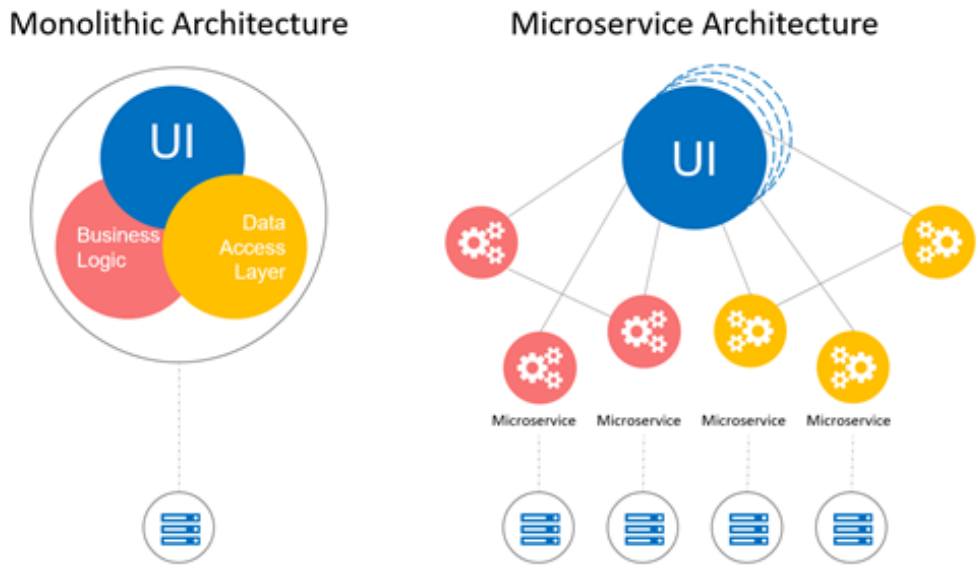


Figure 5 – Monolithic vs. Microservice Architecture[18]

The concept of loosely coupled systems is what the architectural pattern of Microservices is intending to establish. Instead of one clunky monolith providing all the features needed in a domain, it federates the provided functionality into multiple instances of Microservices that communicate loosely and are maintained and deployed separately. That helps to incrementally upgrade and even migrate each Microservice by its own – giving organizations the chance to enhance the architecture and adapt it to new business challenges gradually without having to fund giant migration or upgrade projects. In addition, organizations are also able to pick the best in class tools for their specific needs and avoid too strong dependencies on a single software vendor (➔ *Vendor Lock-in*). A good working definition of the term Microservices is given by Sam Newman:

Microservices are independently deployable services modeled around a business domain. They communicate with each other via networks, and as an architecture choice may offer many options for solving the problems you may face. It follows that a Microservice architecture is based on multiple collaborating Microservices. [8]

That means Microservices, unlike monoliths, are not set up in a classical three-tiered architecture, but are designed to scale horizontally. That means they are modeled on the basis of a certain business domain that is described by a certain bounded context.

A bounded context consists of functions that interact with each other frequently, whereas functions that are rarely interacting belong to different bounded contexts – if they need to communicate, they can do so over the network. Figure 5 shows the different concepts and also highlights the fact that Microservices can have different purposes – Business Logic or Data Access – and in addition

[18] Picture taken and modified from https://appinventiv.com/blog/microservices-vs-monolithic-architecture/,visited 03/06/2021

each Microservice may have its own database in order to satisfy different requirements such as scalability or independent deployability.

That creates the opportunity to implement the best suited database technology with each of the Microservices (➔ *Polyglot Persistence*). On the one hand this especially is beneficial where NoSQL databases are stronger than regular SQL[19]. Many use cases in the engineering domain can better be supported with the use of a *Graph, Object* or *Timeseries Database.* On the other hand, there are of course still use cases that are best supported by classical SQL. A Microservice Architecture gives freedom to pick the best persistence layer for each business domain (or bounded context). It is recommended to give each Microservice where possible its own database to make separate deployments easier. Also from a UI perspective Fig. 5 suggests to have one UI, but also here exceptions are imaginable. Looking at the huge number of mobile apps with different UIs users interact with frequently today, it is not always a hard requirement to standardize the UI in order to enhance usability. Besides the freedom to choose the best suited database technology, the same level of freedom applies to all other levels of the technology stack. Each Microservice should be set up in a way that it is independently deployable and supports it's business domain's use cases optimally.

But most importantly Microservices are an architectural prerequisite to enable DevOps team's autonomy. Here again Conway's law applies. Organization and architecture go always hand in hand. Changing the way a development organization is set up without changing the architecture of the software under development will cause a lot of friction. Why is that the case?

Classical monoliths were usually developed with a so called horizontal development organization. Ideally speaking this has been a database team in charge of the underlying data model and transactions (Persistence), an application team in charge of an applications business logic (Business) and a UI team responsible for the UI (Presentation). In order to develop a new feature there is the need to coordinate between all teams, leading to the fact that for minor feature enhancements a lot of people need to be involved.

[19] See [26]

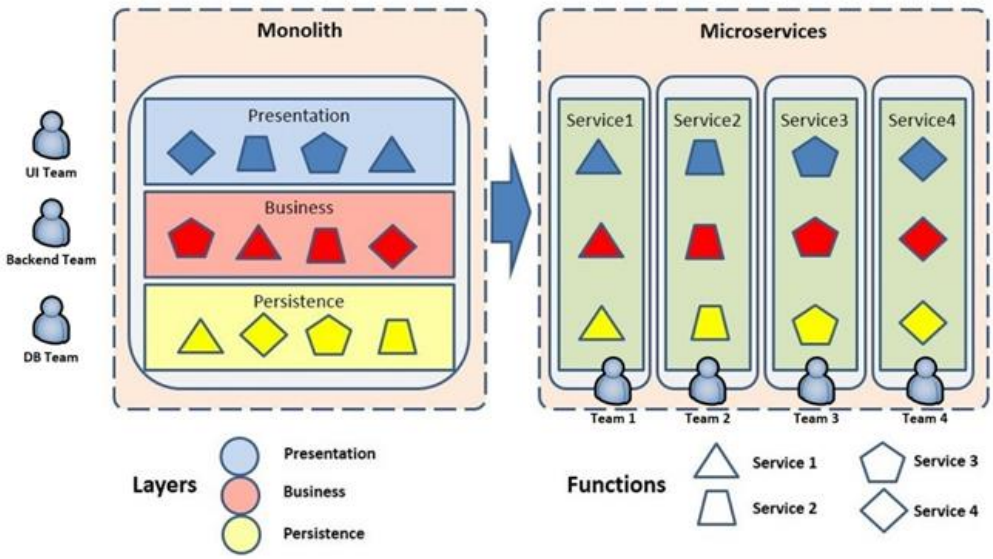


Figure 6 – Monolithic vs. Microservice Architecture with Team allocation[20]

Fig. 6 shows the difference in an applications architecture and corresponding team allocation. In a Microservice approach those teams are interdisciplinary with database experts, application programmers, UI experts and product owners/managers all in one team. The main difference is they are responsible for a rather small business domain / bounded context and can make decisions autonomously. Once they start implementing a new feature they can work on their own, they are not confronted with a lot of dependencies to other pieces of the architecture that they don't have under control. This leads to more pace and agility as it limits the number of people who need to be part of the development – or to state it differently the less people you need to communicate with, the higher is the efficiency of this communication. Decisions can be made faster.

This can also be seen as lessons-learned from the introduction of the SAFe[21] methodology. SAFe takes the concept of agile development and tries to scale it beyond just software development, including product management and marketing. It defines interdisciplinary teams that are able to develop an application with a high degree of team autonomy. And as stated before: way to often here is the problem. Team autonomy can only work if there are little to no architectural dependencies to other teams. If SAFe is introduced and the software platform is still a monolith the promised agility and pace cannot be achieved, because the teams are usually highly dependent on enabler work done by other teams. SAFe supports here with processes helping to identify dependencies that constrain development of new features, but the actual solution to the problem is avoiding and eliminating these dependencies in the first place with an architecture made for team autonomy.

[20] Picture taken from https://dzone.com/articles/a-transition-from-monolith-to-Microservices, visited July 23rd 2021

[21] Scaled-Agile Framework

The Microservices themselves are operated in containers[22] where orchestration systems take care of scaling, deployment and management of these containers. SAFe, DevOps Teams, and an open architecture around microservices does not necessarily have to be hosted in the "Cloud". It is just a matter of fact that a lot of basic services to operate such architectures are already available in existing cloud offerings. Besides the fact that many enterprises will benefit for business reasons from "the Cloud" – lower operating costs, lower invest or capital expense – many classical PLM vendors will move and cooperate with cloud companies just because there is infrastructure in place that they cannot and don't want to develop in house.

Different capabilities of selected public cloud service providers are shown in Figure 7. Besides the core capabilities to provide services for networking, storage and compute, there are many other more specialized and use case driven service categories. Generally, the market leaders are able to cover most of the service categories within their corresponding cloud platforms. The major difference lies in the way the services are implemented and made available to users. *Amazon Web Services* (AWS) for example, is known to provide the most cloud services for the most specialized needs and new services are introduced in the fastest pace. The Google cloud platform (GCP) in contrast has slightly less but more curated services to offer, with the major design goal to be the most developer friendly cloud platform. Microsoft's cloud platform (Azure) has its strengths in being well integrated with the windows ecosystem (e.g. MS SQL, Windows Server) and therefore making a migration of such resources more approachable.

Networking	Persistency, Storage	Compute Services	Database Services	Enterprise Integration	Security, Identity and Access Services		Management & Automation Services		Development Services		App Services	Mobile Services	Analytics and Big Data	Artificial Intelligence Services	IoT & VR/AR Services	Enterprise Applications
Virtual Networking	Attached Disk Storage	Bare Metal / Dedicated Compute	Cloud-native RDBMS	Data Import Export	Account Management	Authentication Authorization	Network Monitoring	Best Practices Advisor	App Deployment Automation	Cloud IDE	Application Hosting	Mobile Apps Development	Data Orchestration	Speech Recognition	IoT Edge	Content Management
Hybrid Connectivity	Object Storage	Virtual Servers	Non-native managed RDBMS	Data Integration & ETL	Compliance	Security reporting	Resource Monitoring	Fleet Deployment Orchestration	Distributed Apps	API Management	Business Apps	Mobile App Testing	Data Discovery	Vision Recognition	IoT Gateway	Managed Email Services
Dedicated Connectivity	Shared File Storage	Container Compute	No-SQL	VM Import Export	Directory Services	DDoS Protection Service	Configuration Change Management	Instance Management	App Testing	API Apps	Send Email Notifications	Mobile Identity & Data Sync	Big Data Processing	Language Recognition	IoT Events	Unified Comms
Network Load Balancing	Archiving – Cool Storage	Container Registry	Graph Database	Data Migration	Information Protection	Application Firewall	Cloud API Log Service	Job Scheduler	DevOps Pipeline	Build Automation	Media Transcoding	Mobile Targeting	Stream Analytics	Text-to-Speech	Virtual & Augmented Reality	Application Streaming
Web Load Balancing	Hybrid Storage	Event-based Compute	In-memory Database	Integration Services	HSM Security Module	Firewall Management	Operations Analytics & Insights	Infra-as-a-Code Templates	Microservices	Code Analysis	Messaging Service	Mobile Device Management	Visualization	Translation		Desktop as-a-Service
Domain Name Service	Backup	Batch processing	Data Warehouse	Server Migration	Certificate Management	Key Management	Exception Reporting	Application Discovery	Managed Source Control	Continuous Delivery	Workflow Service	Mobile Analytics	Data Quality Service	Machine Learning		Marketplace
Internet Traffic Distributor	Disaster Recovery	Scalability	Caching	Enterprise Application Integration		Secrets Management	Service Catalog	Governance and Cost Controls	Web Apps	Game Development	Push Notification		Predictive Analytics	Machine Learning Training		Portal
Content Delivery Network		Simple Compute	Master Data Management								Managed Search			Intelligent Bots		

Figure 7 – Accenture Public Cloud Capability Canvas[23]

[22] For containers and orchestration systems the tools Docker and Kubernetes are currently the de facto standard.

[23] https://www.computerwoche.de/a/wann-und-wie-sich-der-weg-in-die-cloud-lohnt,3544678, visited 03/06/2021.

The Accenture Cloud Capability Canvas captures these capabilities in available services and categorizes them in the following way:

Networking
Services to build reliable and secure virtual private networks within cloud environments. Dedicated or hybrid connectivity to on-premise networks can be enabled and webservice related functions like content delivery or load balancing can be implemented.
Persistency & Storage
Services for different storage types like object storage, shared file storage or backups with high data availability and durability.
Compute Services
Services to provision virtual servers or containers on cloud infrastructure with built-in scalability features. Serverless technologies like Function as a Service (FaaS) are also included in this category of cloud services.
Database Services
Services to provision, configure, and manage different database types like RDBMS, NoSQL or Graph DBs.
Enterprise Integration
Services to help large enterprises with public cloud data migration, ranging from simple data imports, to VM imports and complex ETL scenarios or application integration. This includes also extensive support and solutions for large-scale migrations of mainframe environments (lift-and-shift scenarios).
Security, Identity & Access Services
Services to protect and secure a cloud environment and associated applications including account management, authentication and authorization, key and certificate management and firewall services.
Management & Automation Services
Services for logging and monitoring cloud resources as well as automate the same with e.g. infrastructure as a code (IaaC) capabilities.
Development Services
Services to support cloud developers with specialized cloud IDEs, build and deployment automation tools as well as Continuous Delivery (CI/CD) or application testing.
App Services
Services to build and support integrated application services like media transcoding, push notifications or messaging services.
Mobile Services
Services supporting specifically mobile application development ranging from mobile analytics to mobile application testing.
Analytics and Big Data
Services to build analytics and big data use cases, with data discovery and processing services as well as orchestration and stream analytics functions.
Artificial Intelligence Services
Services for out-of-the-box AI use case support with speech, vision and language recognition but also more sophisticated machine learning training models.

IoT Services
Services for IoT use cases covering edge computing, gateway services and event handlers.
Enterprise Applications
Services for enterprise application use cases like content management, managed e-mail services or desktop as a service.

Cloud offerings are usually accompanied by Managed Services and DevOps Teams. For a working DevOps process it is key to establish efficient pipelining where development of new features, compiling, deploying and testing is effectively managed and the necessary traceability is achieved. That requires a CI/CD[24] process which helps to automate the distribution of additional features to the public cloud and private clouds as well. The mentioned "Twelve-factor App" methodology helps developers to build SaaS-Apps with 12 guiding principles efficiently – it can basically be seen as "lessons learned" from developing Apps in modern Web architectures.

What can be identified here again is the fact that cloud offerings are based on a subscription model. Customers pay a certain amount monthly, quarterly or yearly and receive the service they subscribed with the guaranteed service level, basic enhancements, administration and operation. What they do not pay for is licenses of the developed software. This raises an interesting point: As the vendor of the software is not payed for licenses, he has no incentive to code a lot. In other words he is not generating more revenue by developing more software he has the IP for. If a vendor wants to maximize his profits – and that of course can be assumed, as otherwise he would not be an actor in a free market – he will try to maximize revenue and simultaneously minimize costs. In this scenario this means, he will do everything possible to attract more users to his service (more customers), having less users/customers churn and offer that service in the most efficient way – with a minimal set of resources as an input. Again as he is not paid for the IP he owns, this creates a huge incentive to use open source software. This change in current business model brings objectiveness in software development – software is not better just because a vendor owns its IP. Software is better because it is better satisfying user requirements – even if e.g. the library that has been used is based on an open source license.

3.3 The craft of scoping and implementing Microservices

The term „Microservice" has already been introduced. The question when looking at the PLM domain is what kind of Microservices can be identified and can there be any kind of guidance on how to scope Microservices[25]. In other words what functionality should be implemented in one Microservice and which functions should be separated into two or more Microservices. It is important to

[24] Continuous Integration / Continuous Deployment
[25] Fig. 14 shows the defined Microservices for what is today „PLM" in an IT-project at Mercedes-Benz.

reiterate that Microservices are independently deployable services modeled around a business domain or so-called bounded context.

There are various approaches and methods for defining Microservices and domain contexts. However, in all these approaches, the business capabilities as well as the user-driven use cases are two of the most important criteria when creating domain contexts. Naab et.al. introduce in [9] an additional important criterion for defining Microservices for a particular domain – not just the consideration of what business logic belongs to one bounded context, but also data needs to be part of the consideration. This means: What kind of data is captured, stored and worked with in a particular domain? How frequently is it changed? Does it need to flow between domains or is it specific to one domain? How big is the conflict potential – the risk – when an inconsistent state appears?

The initial goal is here to minimize the movement of data between domains by mindfully defining contexts and their boundaries, thereby reducing the potential for domain data inconsistency. Newman [8] suggests, thus, keeping the data in the same database, thus in the same domain, if the ACID principles are mandatory for the dataset. In other words, the criticality of the data transactions affects the decision about the definition of domain contexts.

However, if for some reason the data has to be split or partly duplicated between different domains and the ACID principles are still to be retained, approaches such as Sagas [10] can be used. Here, transactions are divided into sub-transactions based on the respective business process. Although the atomicity of the business transactions in their entirety is no longer guaranteed, each sub-transaction in different domains is still atomic. In this way, the sub-transactions are committed to different domain databases after each business process step succeed. In case of a process failure, the rollback scenarios are different. In a classic ACID transaction, a rollback is performed before a commit. Thus, no trace exists in the database after the rollback. A Sagas-rollback, however, includes many already committed transactions in different domain databases. This requires the elaboration of rollback scenarios for each business process, ensuring consistency of data in all domains involved. Although a Sagas-rollback leaves a trace in the touched databases, the rollback is semantically equivalent to the rollback of ACID transactions. This helps to satisfy the requirement for semantic replication between Microservices.

Communication and data flow between Microservices and domains can be both synchronous and asynchronous. The simplest synchronous communication is if one Microservice calls the REST endpoint of another Microservice based on an HTTP method. In this case, the calling Microservice waits for a response before sending the next message. This communication method is indeed a flexible way to access data on demand. In asynchronous communication, on the other hand, messages are sent without waiting for a response. This is usually suitable for writing operations and requires a message broker[26] to manage the messages. Thus, each domain publishes its data, which can be accessed by other domains.

[26] For example, Apache Kafka, see [28].

4 Knowledge Graphs and Link-Enabling APIs

The initiative "Plattform Industrie 4.0" underlined in its recommendations from 2013 (See [1]) the demand for end-to-end digital integration for engineering across the entire value-chain. From an IT perspective this addresses the lack of interoperability of engineering tools and especially PLM systems. There have been many different attempts to solve that problem or at least improve the situation. Today's most promising cornerstone for a solution to this problem can be identified with Linked Data – a concept initially developed by the W3C[27].

The automation of design and verification processes is key to improving product development speed and risk reduction. The challenge is that automation workflows can only be established once the relationships between engineering data are described formally. For example, it is necessary to know which simulation result created by which simulation model is used to perform a specific test to verify a requirement. Once these relationships are captured, then it is possible to perform automated workflows to check if new simulation results still satisfy the requirement after a change to the simulation model. Existing automated workflows are currently discipline-specific, and not holistic. For example, continuous integration and continuous deployment workflows for DevOps are specific to software engineering. Impact analysis of requirements on model elements is specific to systems and requirements engineering. Computing the product structure and performing new simulations based on a new lifecycle state of a 3D model is specific to mechanical engineering (e.g. PLM). Creating a new machine learning model based on new data is specific to data analytics as used in IoT. An important next step for improving productivity in engineering is to perform automated workflows at a global level, whereby workflows would execute based on relationships connecting data from all the various engineering disciplines including Application Lifecycle Management (ALM), Product Lifecycle Management (PLM), Internet of Things (IoT), Simulation Data Management (SDM), Enterprise Resource Planning (ERP).

4.1 Linked Data for Engineering

The relationships between engineering data from different sources can be described in many different ways. A common idea would be to describe all the relationships in a central database, as in one of the databases typically used for PLM or ERP. However, this approach does not scale. This approach would be similar as to describing all the links between Web pages in one central database. A distributed approach for saving links has advantages in terms of flexibility and effort. A distributed approach for saving links allows each data source to evolve independently of the others. Each data

[27] World Wide Web Consortium

source can save its relationships on its preferred computer hardware, as long as the links are accessible through the same HTTP protocol. Furthermore, the links can be created for each data source using a different mechanism (e.g. simple text editor, custom HTML editor, WYSIWYG editor, etc.).

The same benefits of a distributed approach for saving links between Web pages also apply for describing relationships between engineering data. For example, requirements will be edited using a specific requirements management application using a specific user interface dialog to create new relationships, such as between requirements or between a requirement and a test case. These requirements-related relationships will be saved on a specific computer in a specific format. Another application, such as a PLM solution will offer different user interface dialogs to create new relationships and save the relationships on a different computer in a different database and in a different format.

The critical question is not in which central database the relationships need to be saved, but how these relationships can be made accessible in a standard way through HTTP. As long as the relationships can be accessed and discovered through HTTP, then additional applications can collect these relationships and decide to store the relationships in different "central" databases for different purposes. For example, the relationships could be saved in an Online Analytical Processing (OLAP) database to run the PageRank algorithm and determine the most relevant full-text search results in the same way that Google indexes Web pages on the Web. Another application could collect the relationships and save them in a graph database, or in a knowledge graph, to enable graph queries and new graph-based applications. Another application could save the relationships in a matrix for machine learning. A distributed approach to saving the relationships and to making them accessible according to a standard protocol enables a scalable data integration strategy, as demonstrated by the World Wide Web.

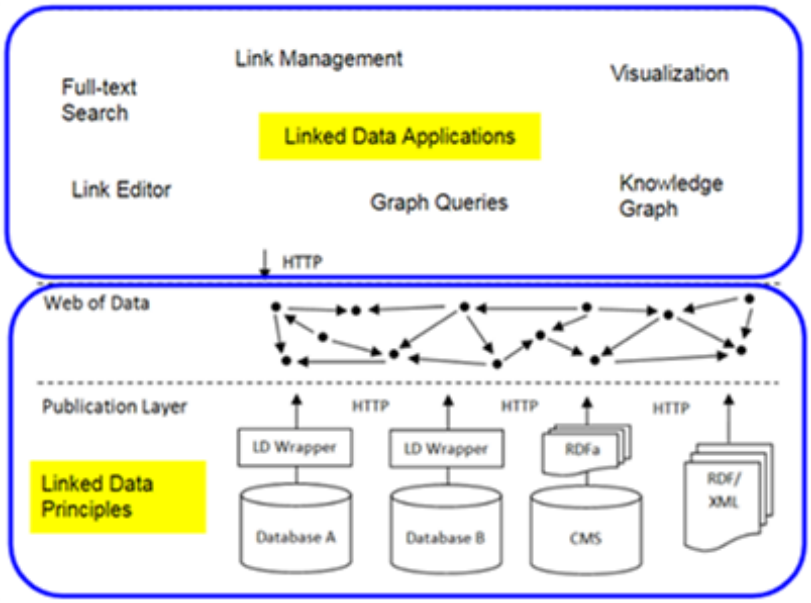


Figure 8– Linked Data Applications and Linked Data Principles

Knowledge graphs and Semantic Web standards are often considered to be important technologies to define and manage relationships between engineering data from different sources. It is important to know how these promising technologies can be used to their fullest potential in engineering. The Semantic Web can be considered an extension of the Web for machine readable data. Its purpose was to allow a new generation of smart applications which could find answers to complex questions such as for finding the cheapest flights, or automatically scheduling a doctor's appointment. This vision was not realized as the public Web contains data which can be vague, inconsistent and false. Automatic reasoners therefore could not make sense of bad data on the public Web.

The fundamental ideas to describe links between machine readable data on the Web, independent of any advanced reasoning capabilities, were summarized by the inventor of the Web Tim Berners-Lee in 2006 as the Linked Data principles. These principles actually help Web search engines to better understand the content in Web pages and to offer improved search results. Schema.org defines a taxonomy for many common concepts such as movies, restaurants, recipes. Web page creators add machine-readable data to their HTML pages according to the Linked Data principles according to schema.org, so that Web search engines can in turn aggregate data from multiple Web pages and improve search results. For example, a Web search for a movie will return the movie ratings from multiple sources such as IMBD and RottenTomatoes, in addition to returning the normal search results. The principles of Linked Data have been adopted at very large scale, however primarily in the context of Web pages pointing to the standardized schema.org concepts.

The adoption of Linked Data principles within the enterprise with private data is limited. Many applications consume RDF data by importing RDF in the form of files (e.g. through bulk import) and by offering a SPARQL endpoint. Very few applications consume RDF data through HTTP requests, as is required by Linked Data applications and even less applications expose RDF data as dereferenceable HTTP resources as required by the Linked Data principles. The main exception are applications complying with Open Services for Lifecycle Collaboration (OSLC) which are used primarily for requirements, test case management and task management. This is unfortunate as private enterprise data is of high quality, and private enterprise Linked Data could be the basis for many enterprise Linked Data applications as shown below.

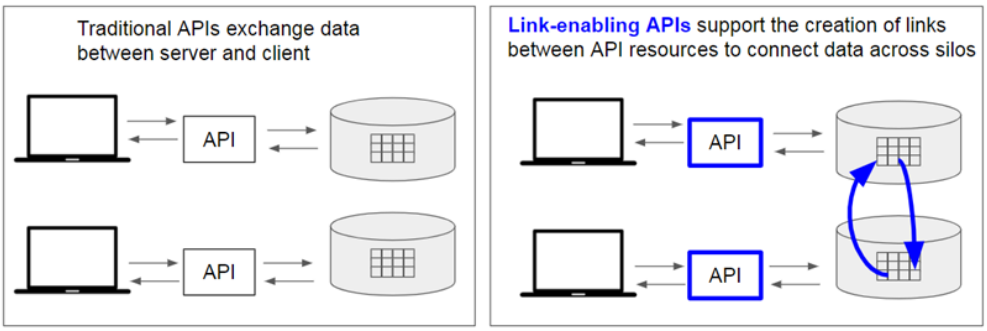


Figure 9 – Traditional vs. Link-enabling APIs

While traditional APIs support the exchange of data between a server application and clients, an API complying with OSLC supports the creation of links between API resources from different APIs. As a result, links between data from different silos can be created ad-hoc by API clients, in other words by engineers themselves. OSLC relies on the Linked Data principles and provides additional standards to support easy ad-hoc linking of Web resources by human users, as well as data interoperability standards for important engineering data aspects such as versions, change events, access rules, schemas, and containers.

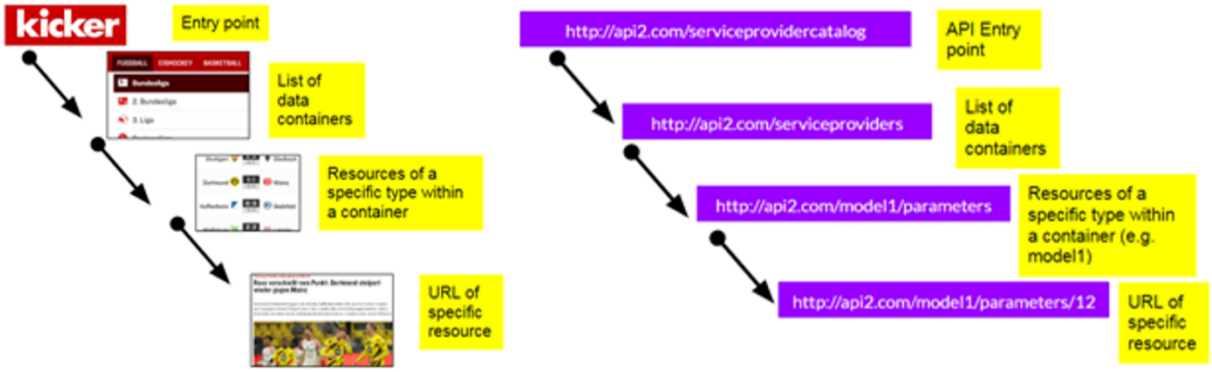


Figure 10 - Example for Hypermedia

However, the creation of links between engineering data, such as between a specific requirement and a test case, can only be done by engineers manually. The engineer needs to define the link by choosing a link source, a link type, and a link target. Even though the link source and link target are typically created in different applications, the engineer wants to create the link without switching applications. Ideally, from a user experience perspective, the engineer can create the link by staying in one application. A challenge in this scenario is for the engineer to discover the HTTP URL, in other words the unique global identifier, of the other API resource he wants to create a link to. Human users can easily discover the URL of Web pages by using a search engine. However, API resources in the context of an enterprise are private and not indexed by public search engines. The engineer therefore needs to discover the HTTP URL of the resource he wants to link to by using a different technique called hypermedia. We use it all the time when browsing the Web. The idea is that the user only needs to know the URL of an "entry point" resource which will help the user discover other available resources linked to the entry point resource. For example, I know that a website like kicker.de will have a Web page containing the report on a specific soccer game but I don't know the specific URL of that Web page. I can discover that URL by first accessing the entry point kicker.de resource, then selecting a category like Bundesliga, and then selecting the specific Bundesliga game I'm interested in among the games that have been reported.
Similarly, an API can provide an entry point resource, which an API client can use to discover on his own available resources provided by this API such as API resources describing categories (e.g. simulation model) and specific resource types (e.g. parameter) and specific resources (e.g. parameter123). OSLC APIs support this notion of hypermedia and enable resource discovery based on an

entry point resource called ServiceProviderCatalog. An application #1 can then communicate with another application #2 having an OSLC API and discover possible link targets to be displayed to the user in application #1.

In order to find answers to complex questions, voice assistants are currently in part realizing the vision of the Semantic Web. These voice assistants use in the background knowledge graphs composed of trusted data maintained by private organizations such as Google and Amazon. The term knowledge graph was coined in 2012 by Google. Existing knowledge graphs are currently mainly used to answer questions about general knowledge such as who the president of a certain country is. General knowledge is found typically in Wikipedia articles or in documents resulting from a Web search. These documents contain natural language, which is then converted automatically into knowledge graphs, which in turn can be queried to find the answer to a complex question. The same approach could be used in an engineering context in which an engineer would ask his voice assistant on his smartphone which parts satisfy a certain requirement which costs less than 100 EUR. As a result, the engineer would then see the answers to his question presented on his smartphone. This scenario can be achieved if the engineering data originating from different sources, as well as the relationships between the data from different sources, can be accessed easily, for example through a common HTTP protocol as it is the case with REST APIs, in order to build a knowledge graph.

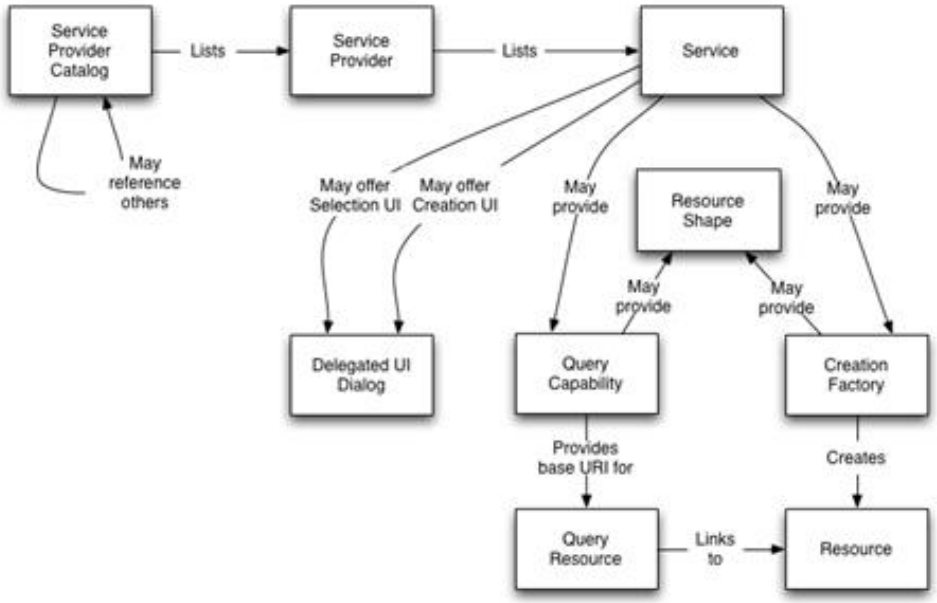
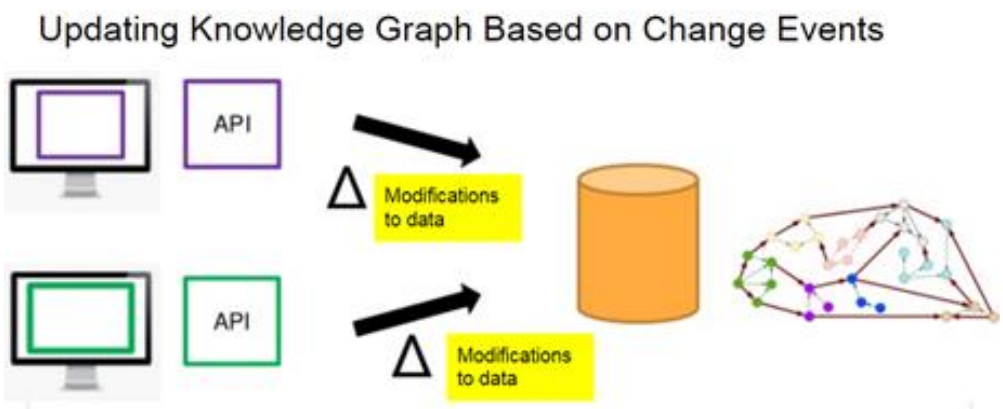


Figure 11 - OSLC Core

The following aspects make the adoption of knowledge graphs for engineering data more challenging than for general knowledge: data changes, access rights, versions of data, and the definition of relationships. Engineering data changes much more often than general knowledge. This means that each data provider needs to expose changes to data in a standard way according to a standard protocol (see e.g. OSLC TRS[28]). Engineering data is only visible to certain users whereas general knowledge is visible to everyone. This means that each data provider needs to expose its data access rules in a standard way according to a standard protocol (see e.g. OSLC TRS). Engineering data is very often versioned whereas general knowledge is not. This means that each data provider needs to expose the different versions of data in a standard way according to a standard protocol (see e.g. OSLC Configuration Management).

Relationships between general knowledge is often described through links between Wikipedia articles. When content from Wikipedia is transformed into a knowledge graph, links between Wikipedia articles are transformed into edges of the generated knowledge graph. Relationships between Web pages or relationships between Wikipedia articles are defined in a markup language using for example HTML anchor elements, or doubled square brackets in the Wikipedia editor. Each link points to a target Web page or Wikipedia article identified by a unique global identifier in the form of an HTTP URL. It is easy to find the URL of a Web page using a Web search engine. It is easy to find the URL of a link target. On the other hand, engineering data is not public and it is not indexed by public Web search engines. There is no widely used equivalent of a Web search engine for private company data. As a result, it is much harder for engineers to find the identifier of engineering data to be used for link targets, and thus to define relationships between engineering data. In order to overcome this handicap, each data provider needs to facilitate the discovery of its data. A lot of engineering data belongs to data containers in the form of projects or repositories. And engineering data is typically associated with one or several data types. So, each data provider needs to expose in a standard way its data containers and its type-specific data access methods (see e.g. OSLC Core). Then engineers can discover possible link targets using the same approach independent of the specific data provider.



28 OSLC Tracked Resource Set

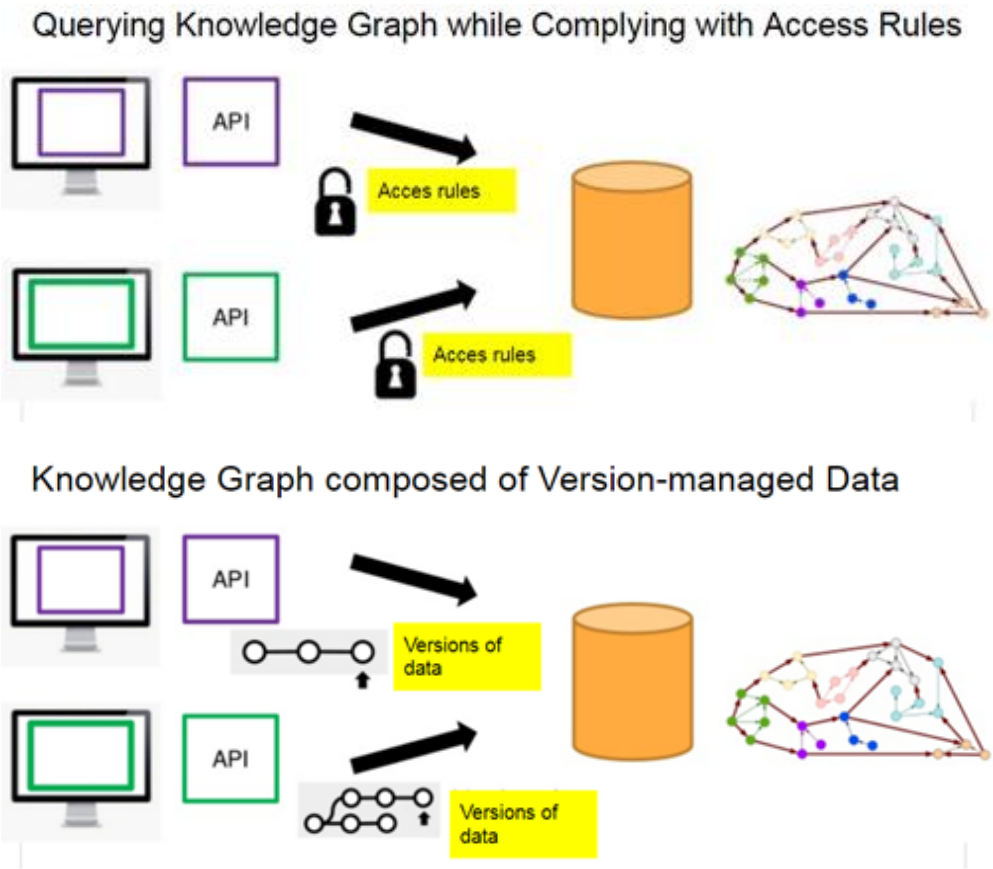


Figure 12 – Knowledge Graph with Change Events, Access Rules and Version Managed Data

Overall, the use of knowledge graphs in engineering is very promising. However, engineering data is not general knowledge data. Engineering data is private data which changes often until released and whose access is restricted to certain users. By adopting standards to expose engineering data, each data provider can facilitate the linking of engineering data, and in turn the creation of an engineering knowledge graph. APIs which adopt standards to facilitate the linking of data can be called link-enabling APIs, in contrast to the traditional producer-oriented or consumer-oriented APIs. Right now, link-enabling APIs are already used in engineering in the form of Open Services for Lifecycle Collaboration (OSLC) APIs.

4.2 Semantic Web technologies

The origin of the WWW can be traced back to solving the interoperability issue between Hypertext editors at CERN[29] where Tim Berners-Lee (see [11]) basically invented the Internet as we know it today. Back then, research results were captured in a variety of hypertext tools and distributed amongst many computers. To exchange research data faster and more efficiently Berners-Lee used the idea of existing Hypertext editors where linking meant relating passages or lines inside of a hypertext document to allow to jump back and forth. He combined that with the idea of retrieving a document from another computer connected to the network. To achieve this, he invented the protocol HTTP[30], the global address room URL[31] and the standardized markup language HTML[32] to exchange hypertext documents between server and browser.

Since then the WWW has of course developed further. From Web 2.0[33] that refers to websites that emphasize user-created content, Semantic Web as a critical component of what is understood to be Web 3.0, to the newest result of the Internet's evolution: Linked Data. Linked Data builds upon existing Web technologies but emphasizes the point that Semantic Web should not just help a human being to better make sense of a website, but extends it to make websites be readable by computers as well. With Linked Data not just HTML documents are shared between server and browser as it has been the case for Web 1.0, now it is about exchanging data – fulfilling the vision to make the internet itself a global database allowing computers and users to retrieve any captured information from it.

[29] Conseil européen pour la recherche nucléaire (former name), today: Organisation européenne pour la recherche nucléaire. It is based in Geneva, Switzerland.

[30] Hypertext Transfer Protocol

[31] Uniform Resource Locator, in more general terms URI is used, which stands for Uniform Resource Identifier.

[32] Hypertext Markup Language

[33] Also known as Social Web

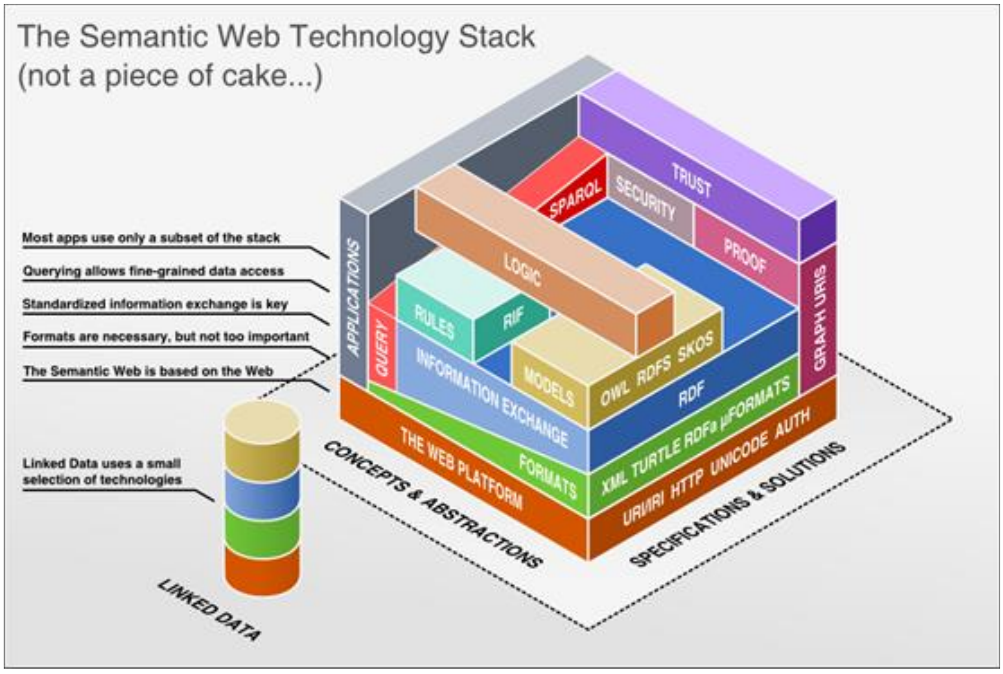


Figure 13 – Semantic Web Technology Stack, see [12]

The Semantic Web attempts to make the Web "intelligent" by using various technologies that build the Semantic Web Stack (see Fig. 13). As illustrated in the figure, the established Web technologies form the basis of the Semantic Web. Linked Data is seen as a subset of this stack. Communication mechanisms such as HTTP, encoding standards such as Unicode, identification mechanisms such as IRI/URI, and authentication mechanisms are implemented in this layer. On top of that, representation syntaxes and formats such as TURTLE, JSON-LD, RDFa and µFormats as well as the Resource Description Framework (RDF) are built. RDF is indeed one of the main building blocks of the Semantic Web, standardizing the description and exchange of data across the Web. RDF also makes the data queryable via SPARQL[34], which is another important building block for both querying and handling data. The technologies described so far are truly essential for standardized data exchange and data integration on the Web. However, the real Semantics is enforced only by the introduction of modeling languages (such as OWL and RDFS) and rule languages (such as RIF and SHACL). These languages are crucial for creating sophisticated models of real-world phenomena and systems, in order to breathe the Semantic soul into the data and make it machine-interpretable. The core promise of the Semantic Web is achieved by using the aforementioned highly standardized technologies as we create applications. Top layers of the stack (including Logic and Trust) are not yet standardized and are mainly still in academic stages.

[34] SPARQL Protocol And RDF Query Language

During work on the initial HTTP protocol – the foundation of the internet – Roy Fielding identified a certain architectural style all interactions on the Web are based on (see [13]). The so called Representational State Transfer (REST)[35] describes a set of constraints to be used for creating Web services. REST defines a set of rules and constraints on how to serve and operate on resources available on – but not restricted to – the internet. Those resources are uniquely identifiable through a Uniform Resource Identifier (URI) and can be accessed by standard HTTP methods, like any common internet browser offers. Additionally, REST defines a strict separation between the data and its representation. This means that different applications can receive the same data in different formats. For instance, a human readable form might be displayed inside a Web browser, whereas an application trying to integrate the data into its workflow might receive it in an interchangeable format, like JSON. In the context of Microservices one of these constraints „Statelessness" is most critical as it enables loosely coupled systems. It does that by not storing any kind of information relating to a connection on the server side, but keeping it all encoded in the message. This means that only the clients need to keep their state, and the server can handle an indefinite number of requests, without having to worry about internal resources. An added side-effect of not having the server bound to individual client connections is an enhanced scalability. RESTful APIs are therefore an additional prerequisite to enable an IT-architecture based on Microservices.

By design, data on the internet has no given semantic meaning. To be able to describe date encountered on the net, efforts of the Semantic Web resulted in the Resource Description Framework (RDF). RDF takes two central properties of the internet into account: that everything on the internet can be reached under an unique address and that everything is connected through "links". RDF utilizes these two properties by first declaring every "object" as a resource, which can be uniquely identified by its URI. It is then possible to semantically describe a resource, by naming the "link" together with its origin and destination, which results in a so called "triple". By using this approach, structured and semi-structured data on the internet can be semantically described and used between different applications. Not only objects themselves, but also the objects they link to, as well as the relationship itself, can be part of other triples. That way a Web of relationships gets created, which results in a natural graph.

A combination of both technologies – RDF as a semantic description about *what* the data represents and REST as a means of *how* to access the data is thus a powerful and scalable approach to build futureproof applications in the form of Microservices.

[35] See [25].

4.3 To be truly Data-Driven we need Data-Centric Architectures[36]

The fragmented IT-Landscape in enterprises is one of the main impediments of being Data-Centric. The common practice to satisfy the needs of existing and emerging processes and methods in the company is often to build or buy a new IT-Solution, which is often an application with tightly coupled database and a fixed data-schema. Thus, a new silo is created with lots of redundant data and lots of redundant functionalities, and the technical debt continues to grow. This leads, inter alia, to high costs for the management and synchronization of redundant data, high maintenance costs of interfaces for data and tool integration, high manual effort to handle data in different data sources and increasing possibility of using wrong or outdated data. These tightly coupled applications and repositories also make the development of new methods and solutions significantly more cost- and time-intensive.

Coping with the already mentioned industrial challenges needs however a seamless integration of tools and full traceability across the system life cycle no matter where the data is stored. Indeed, a data-centric architecture is a prerequisite for a data-driven enterprise. To achieve this, the applications need to be decoupled from the databases and data repositories. This is especially necessary for reading and consuming data from different data sources with standard Web-based interfaces. As illustrated in figure 14, a data-centric architecture[37] divides the enterprise IT landscape into three layers, namely the Application layer, the Semantic layer and the Data layer. In this architecture, the semantic layer is the key building block in providing a unified view of business objects and their connections across different domains in the enterprise, regardless of where the data is stored. The technologies of Semantic Web play a crucial role both in building and in connecting ontology models (based on OWL, RDFS, SHACL …) as well as in seamless integration of different layers (based on OSLC APIs, SPARQL-Endpoints …).

This data-centric architecture focuses on the knowledge workers and domain experts in the enterprise and tries to provide them with all the data they need to do their jobs. Thus, the users receive their desired data from various data sources in their predefined context and in their favorite application. For example a designer who works with a CAD application, can access and even modify the requirements related to the components in development directly in the CAD environment and does not need to use a second application just because the requirements data is stored in a separate repository. Furthermore, this architecture is an important enabler for a cross discipline impact analysis based on various interconnected and cross-linked (ontology) models both at the Semantic layer as well as at the Data layer. This transforms indeed the fragmented data silos into a live enterprise knowledge graph.

[36] See http://www.datacentricmanifesto.org/

[37] The illustrated architecture is a result of a concept study carried out at Mercedes-Benz AG.

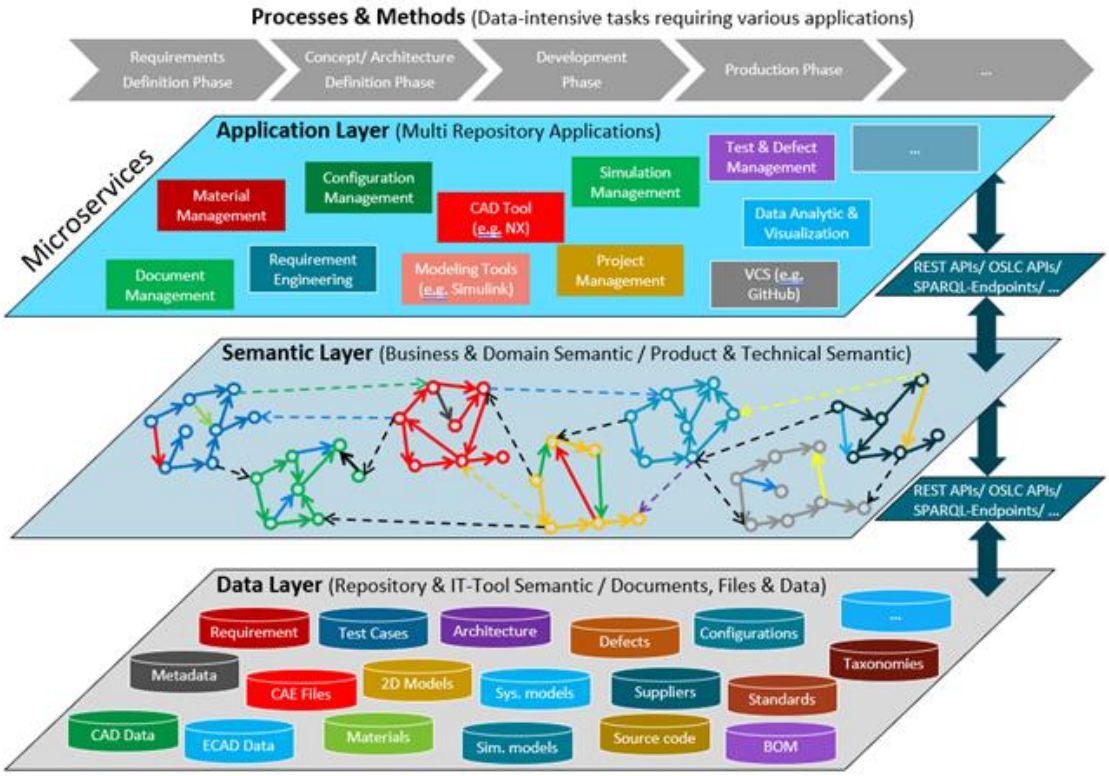


Figure 14 – Data-Centric Architecture for a Data-Driven Enterprise (Source: Mercedes-Benz AG)

The proposed architecture based on Semantic Web technologies enables and fosters significantly the emergence of Microservices in the enterprise. As shown in Figure 14, each Microservice covers only a subset of functionalities and a subset of data belonging to a domain context. The holistic picture can then be built simply by linking the Microservices at the metadata and data levels. This paves also the way for a long desired federalized PLM landscape with data scattered in various databases with theoretically different technologies, necessary to satisfy best the requirements of the corresponding Microservice or use-case. It is only important to use standard semantic tools and languages to ensure inherent interoperability at both the model and data levels. Thus, data in one Microservice can be handled as first-class citizen in other Microservices.

The API schema of link-enabling OSLC APIs adopts the Open World Assumption. API clients can change resources exposed by an OSLC API, as long as they don't violate any constraints. These constraints are used to ensure that resources of the same type have a minimum common set of properties in order to support interoperability. For example, OSLC requirement resources should be described according to the OSLC requirements management specification. An OSLC specification consists of an RDF vocabulary and a separate set of constraints defined as OSLC resource shapes or alternatively as W3C SHACL. The RDF vocabulary defines a set of URLs for identifying resource types (e.g. Requirement type identified by http://open-services.net/ns/rm#Requirement) and resource properties (e.g. verifiedBy identified by http://open-services.net/ns/rm#verifiedBy). The constraints

relate to the multiplicity of properties for example to indicate that a requirement can only have one title property. Domain-specific OSLC specification, such as related to requirements or change management, are likely to evolve over time, especially in order to be better aligned with domain-specific standards such as ReqIF, SysML, or STEP Application Protocols defined by traditional standardization organizations. It is technically simple to convert an existing domain-specific schema into an equivalent Web-compatible OSLC specification. The officially adopted domain-specific OSLC specifications only cover a few domains related to systems and software engineering. Many non-official domain-specific OSLC specifications have been created by developers of OSLC APIs to describe domains related to simulation (e.g. FMI) and MBSE. While the domain-specific OSLC specifications are likely to evolve over time, similarly as domain-specific schemas (e.g. SysML v1, and then v2 etc.), the domain-independent OSLC specifications related to resource discovery, global configuration management, change events, and other generic data aspects are expected to change less over time. The domain-independent OSLC specifications are therefore considered more stable and more important than the domain-specific OSLC specifications.

OSLC as a specific implementation for Linked Data in the engineering context also has some drawbacks that need to be handled, most importantly if there are legacy tools with proprietary APIs that do not allow to develop an OSLC adapter quickly, there needs to be an alternative way of getting data out of its cage. Especially in the MCAD/ECAD space there is little to no alternative than to create connectors that are able to talk to the legacy proprietary APIs. Ideally these connectors are not just point-to-point connectors but all the connectors are part of a bigger framework that helps to neutralize legacy APIs. One example is the XPLM connector framework that supports multiple ECAD, MCAD and PLM-Systems[38].

[38] https://www.xplm.com/our-solutions/

5 There is no Digital Thread without MBSE[39]

The art of product development has gone through tremendous changes over the last century. Where it started with pencil and drawing board we now face a situation where product data is created in an uncountable number of software tools that help to formalize, simulate and therefore automate the development of a product – in order to bring it faster to the market, satisfy more customer needs, improve quality and maintainability.

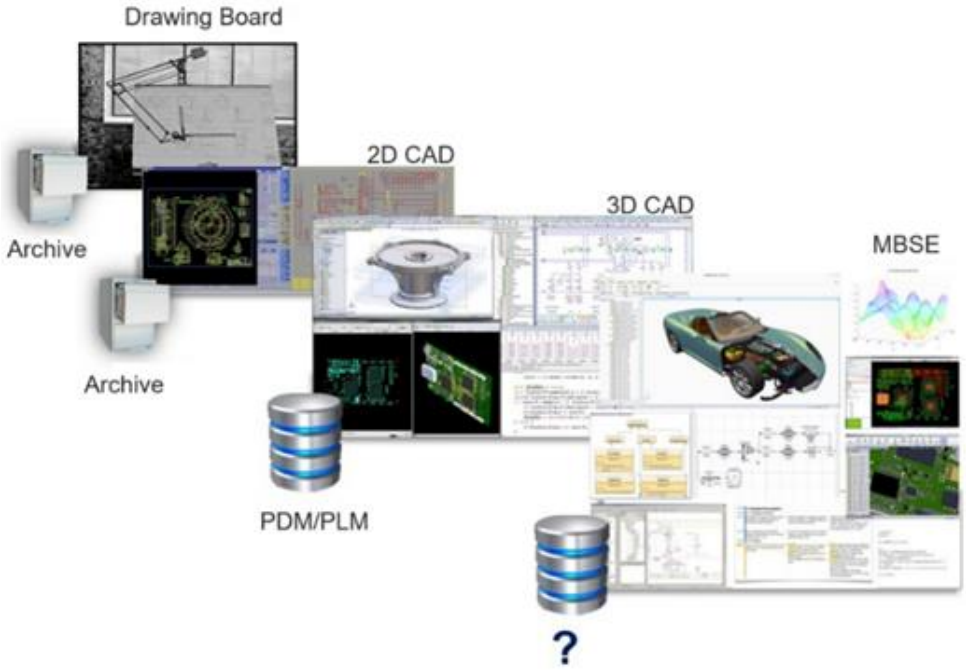


Figure 15 – History of Engineering Tools [14]

Model-Based Systems Engineering (MBSE) is just the latest methodology and tool set that aims to conquer the challenges modern product development is facing: From autonomous driving cars to self-assembling manufacturing systems, from smart cities to electric airplanes. The main challenge is not the introduction of a system modeling tool. It is how the introduction of MBSE methodology and tools can fit into the existing landscape of tools and processes.

[39] Model-Based Systems Engineering

Looking back to the history of product development we can see several phases (see Fig. 15). Before there was any computer support in product development, engineers worked with physical drawing boards. As there has not been much electronics nor software in their products and the complexity of parts in terms of their geometry has been manageable, this tool set was sufficient at times. Drawings were managed in an archive and they received numbers, title and name of creator - essentially *meta data*.

The next phase can be characterized by the introduction of 2D CAD[40]. The first time that computers helped to create drawings. This phase basically replaced the physical drawing board by a computer that ran a 2D CAD tool – a change in authoring system, as they are usually categorized. With the introduction of 3D CAD – phase 3 – the necessity occurred to manage the created models virtually and therefore PDM/PLM tools were invented. In the meantime, of course ECAD[41], EDA and CASE[42] tools were introduced to also help the non-mechanical engineers to develop their respective components more efficiently. Although some of those systems' implementations are using direct connectors to manage their data in PDM/PLM or connect their respective TDM[43] to a PDM backbone, most of PDM/PLM implementations in the world still focus to a large extend on the mechanical domain. ECAD and CASE is seen as separated silos in terms of IT architecture and organization-wise.

From todays perspective it gets clear that the number of tools and models that are used to describe a product under development is overwhelmingly large. Those models have not been implemented to interoperate amongst each other and the corresponding tools do have a lot of overlapping functionality. For instance structural information is described in any of the tools – unfortunately in its own language.

MBSE is aiming to bridge this gap between the engineering disciplines and tear down the walls between those silos. It therefore formalizes established Systems Engineering processes and best-case practices in so called system models. Rather than working with documents and spreadsheets system engineers started to work with formal descriptive models of the system under development (e.g. in a semi-formal language like SysML[44]). Those models capture more information than just structural (what has essentially been the strength of PDM/PLM tools) – in addition system models are able to capture behavioral, parametric information and requirements. That is why the next decade of development is about developing systems rather than products and capture additional behavioral information the existing tool set consisting of CAD/PLM is not able to do.

A very exciting reply to today's challenges is given by the new version of the modeling language SysML v2 (see [15]). A fundamentally new system architecture is not based on UML anymore but on a semantic meta object facility and on the kernel modeling language (KerML). The new architectural approach is essentially more simple and parsimonious. But the main advantage lies in the

[40] Computer-Aided Design

[41] Electrical/Electronical CAD

[42] Computer-Aided Software Engineering

[43] Team Data Management

[44] Systems Modeling Language, standardized by the Object Management Group (OMG).

extensibility and the leveraged model library support. The models are more unified. Structure and behavior models are deeper integrated and the terminology is cleaner and simpler. All this provides an ideal approach how the challenges in today's landscapes should be tackled.

Todays' system models are purely descriptive which means in order to be executed they need to integrate with simulation models. There are many examples of such integration efforts. It can be done on the basis of 0D/1D-simulation tools, e.g. in a language like Modelica or MATLAB/Simulink[45] – it can also be further supported by integrating with 3D-simulation tools, be it CFD, FEM or MBS[46]. In this context especially OMG's SysPISF[47] specification and Modelica Associations' Project "System Structure and Parameterization" need to be mentioned.

One main problem that MBSE currently faces is its adaption amongst non-systems engineers that need to consume and work with data that is initially created in the system model. The steep learning curve that is characteristic for systems modeling leads to the creation of a "MBSE silo" next to the existing engineering silos. In order to make MBSE successful it needs to be connected to the Digital Thread. And because MBSE is aiming to formalize early-phase development processes that were till now only document-driven there is no complete Digital Thread if it does not include MBSE.

The Digital Thread itself is not a tool chain where content from one model is transformed into content of the next model. This could be the understanding of it from an MDA[48]-perspective where Platform Independent Model (PIM) is transformed into Platform Specific Model (PSM)[49]. The Digital Thread purely wants to connect all relevant artifacts that are part of the development of a system, trace between them, ensure their configuration and help organizations to identify what is affected in case of a change.

The Digital Thread runs through an organization and connects the digital artifacts that an organization produces. It is as organization-specific as it sounds and therefore it is just impossible to implement the Digital Thread vision on top of an OOTB, inflexible, clunky monolith. The point for low code platforms which do not ignore the fact that customizations are necessary, but have those capabilities built in and by doing so bring them under control, cannot be stressed enough.

Making MBSE be part of the Digital Thread needs prerequisite conceptual work. There have been a couple of research projects and academic publications on the topic of MBSE/PLM integration that are worth looking at. In [4] Dickopf discusses a holistic system development methodology consisting of process, method and tool. Pfenning describes in [14] a way to map, synchronize and manage system models in a PLM system with his System Concretization Model. Both publications are based on previous concepts by Gilz. He showed in [16] a way to move system model data to PLM with a

45 © The Mathworks, Inc.

46 Computational Fluid Dynamics, Finite-Element Method, Multi-Body Simulation.

47 Object Management Group, SysML extension for Physical Interaction and Signal Flow Simulation

48 Model-driven Architecture, see [27].

49 This may be an interesting area of research for the next years to come: Adapting the MDA for Systems Engineering. The real world implementation of the digital thread is the more pragmatic scenario. The digital thread on the other hand can be seen as an initial prerequisite to work on this so called "Systems-MDA".

focus on profile, methods and visualization of the result in a PLM backbone. All of that lead to the Kaiserslautern' System Concretization Model (KSCM) described in [17] and shown in Figure 16 – see also [18]. Hooshmand et. al. proposes in [19] and [20] a landscape for the integration of PLM and MBSE and the semi-automatic creation of system models based on ontology-based model libraries.

Lessons learned from academia is that integrating MBSE with PLM needs a huge amount of flexibility – not just in terms of the PLM data model, but also in the ability to map and synchronize any element (be it object or relationship) of a system model. This is needed, because used methodologies and moreover used stereotypes in system modeling languages[50] differ tremendously between organizations.

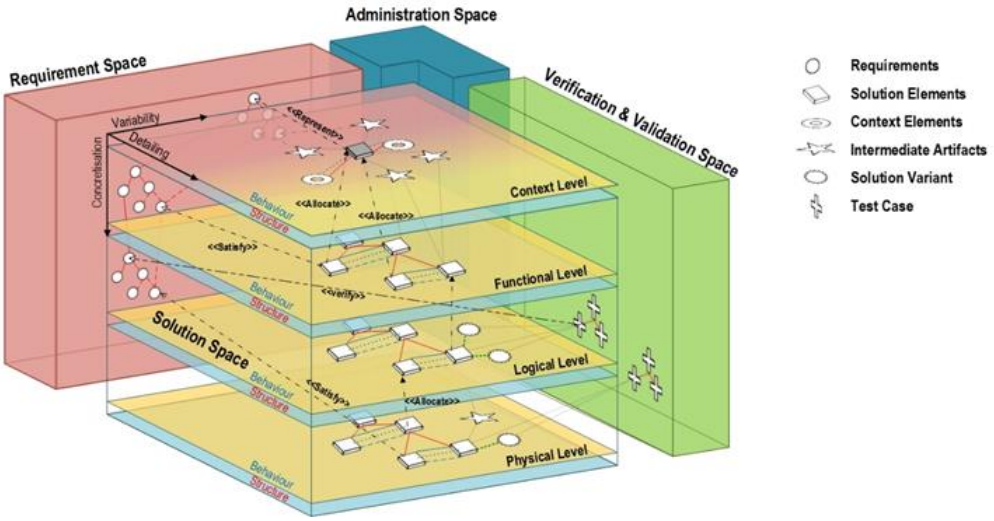


Figure 16 – Kaiserslautern' System Concretization Model (KSCM)

Pfenning introduces in [14] the term *anchor element* to illustrate the problem. As stated before the implementation of the Digital Thread needs to be based on a set of digital artifacts (or *work items*) that are created by an organization to capture and document decisions. This could be for example requirements, use cases, activities, issues, builds, CAD models, parameters, features, libraries, schematics, layouts, part lists, test protocols and so on. Obviously, this can lead to an endless list uniquely defined for each organization. The Anchor Elements concept does not want to restrict that list to a standard set of this artifacts. The concept wants to find a subset of all of those artifacts that are needed to communicate interdisciplinary – meaning between the engineering disciplines. This necessary abstraction needs to be done in order to make interdisciplinary collaboration successful. An initial compromise for a standard set of anchor elements can be seen in the Kaiserslautern' system concretion model.

[50] More specifically in SysML

It consists of several so called spaces: The requirements space that captures the Requirement Model, the verification and validation space that captures the model necessary to test and simulate the developed systems in all its different digital models, the administration space that captures the necessary meta data, processes, rules and so on to manage models and data, and finally in the middle the so called solution space where all models can be found that describe the system under development from every viewpoint – and viewpoint here means again in any necessary type of digital model, be it MCAD, ECAD, or the system model itself.

The solution space knows 4 abstraction layers – 4 ways to describe the solution, getting more concrete from the top to the bottom: from a contextual viewpoint (what is my system's context/environment? Is it part of a bigger system of systems?), from a functional viewpoint (what is my systems functionality without looking at its subsystems, components?) from a logical viewpoint (what are my systems' subsystems? Which system elements can I define in order to lay out its architecture?) and finally from a physical viewpoint where the link to existing product structures is made.

All structures on each of the layers can of course be subject to variation and therefor there is a dimension called "variability" that allows for definition of alternatives and is therefore a prerequisite to support tradeoff analysis. Furthermore, there are several links between elements that help to trace: verify[51] to find the right test case to verify a requirement, derive to break requirements down (cascading requirements), satisfy to show which solution element can satisfy a requirement, allocate to show what element on a lower layer of the solution space is implementing which element of a higher layer.

On each layer also flows can be found that describe behavior. Flows can in general be of type energy, information and matter. Behavior is therefore described by flows between elements, not so much by elements themselves.

From a practical IT-architecture perspective it is obvious that all of this information cannot live in just one software tool. It is necessary to have several tools capturing the described partial models communicate with each other. Linked Data is one cornerstone for this communication and it is obvious that the semantics on each trace link (verify, validate, allocate, satisfy, derive, …) fit quite nicely into a RDF[52]-triple definition.

What is not covered in the KSCM is the instance level, and therefore the lifecycle phases of operation/maintenance and production. This is where Digital Twins also need to be connected to the digital thread.

[51] In addition validate – not shown here, but a necessary additional relationship to trace for validation.

[52] Resource description framework

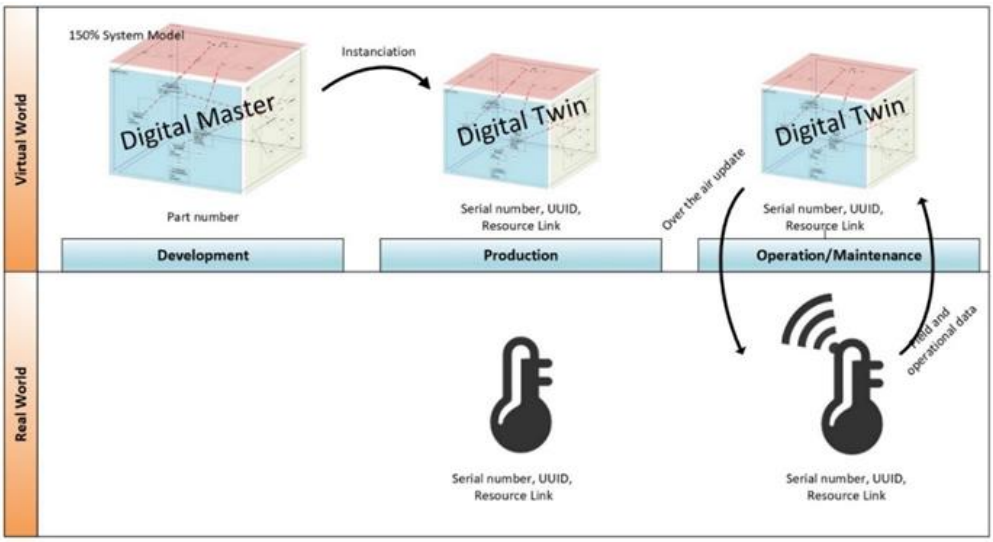


Figure 17 – Digital Master and Digital Twin

The Digital Twin lives on an instance level – as the computer-science community would say. For other engineers the difference between Digital Master[53] and Digital Twin[54] can be explained by just saying the digital twin corresponds to a real twin that can carry a serial number – it corresponds to an actual build system a customer can buy – whereas the digital master is every discipline model and every piece of data necessary to define a type of product and therefore it can carry a part number.

The KSCM (as illustrated in Figure 16) only shows the inner structure of the Digital Master. There has been additional work on bringing this type level together with the instance level, represented by the Digital Twin. In [21] Muggeo and Pfenning show a high-level process on how the Digital Twin can be instanciated from its Digital Master. Fig. 17 shows this process.

There is the real world reserved for real manufactured and operated systems and there is the virtual world that captures all data and models necessary to describe the system over its lifecycle. In the development phase there is usually not yet an existing twin corresponding to the real world – at least when it is about the initial development. The Digital Twin gets instanciated from its Digital Master – which most of the times (but not always) means that the Real Twin gets produced. The Master potentially knows variability – this is why the 150% system model can be found here. After production the real twin is operated, and the digital twin can be used to make sense of collected field and operation data – this may help to optimize the design or even optimize the operated real twin – e.g. less energy consumption by using a better engine control configuration. It is also possible

53 The term digital master originates from the iViP research project. This is also the reason why Prostep iViP Association has iViP as its middle-name.

54 Industry 4.0 Initiative came up with the term „Verwaltungsschale" (eng: administrative shell) to express the need to digitally describe a component in a factory. Both terms can be understood to be very similar.

to virtually first verify software updates on the digital twin before updating the real twin over the air. This can be referred to as *closed loop engineering*.

The main question to answer in this context is potentially what kind of data is needed for the digital twin and what data is only referenced or inherited from the type level. The answer to this question may as well be company specific and cannot always be standardized that easily.

It goes without saying that any implementation of a System Lifecycle Management System (SysLM) must include not just the digital master on a type level with all the artifacts described above, it must also include the Digital Twin concept. That's why IoT platforms must also be part of any IT-architecture aiming to implement a System Lifecycle Management System.

6 The Concept: Cloud-native System Lifecycle Management (SysLM)

The System Lifecycle Management (SysLM) concept isn't really new, but its implementation in the industry has not yet been achieved due to the obvious amount of work necessary to fulfill this vision[55]. The authors of this paper agree that any implementation of a SysLM concept can only be realized by the usage of a decentralized IT-architecture. This means any implementation effort must find answers to the question how the three main building blocks described above – Microservices, Linked Data and MBSE – can be brought together and how those technologies can operate jointly in a holistic concept. On the following pages, a possible concept is presented. This is not the only valid concept for solving this problem, but it can certainly serve as an illustration of the necessary problem-solving process.

As illustrated in figure 14, on the highest layer (the application layer) the involved Microservices are defined. In this example, 18 such Microservices are defined, showing that roughly 1-2 dozens Microservices are required to have the same functionality as in a PLM monolith. Each Microservice is, as stated before, independently deployable and possesses its own persistence layer. The Microservices or bounded contexts cover the entire lifecycle of a product or system. Starting from Requirements Engineering, to (M)CAD Data Management, to ECAD data management, including Meta Data / Master Data, Bill of Material Management, but also of course including the Software domain to include things like source code management[56]. The scope of each Microservice is defined by the underlying business context.

In a more generic concept, all Microservices will have their own endpoints and API to communicate with each other via semantic links. Figure 18 shows a generic cloud-native Microservice. As stated before, it will run in a container (here Docker), it will have its own operating system – as it is running in a container this layer will only have the most relevant capabilities in there. Above we see its own persistence layer, depending on the best data base technology for the business context. It should be noted that when scaling, the Microservices are cloned, but this rarely means that the database is also cloned. In other words, each Microservice in the bounded context has its own database, but the database is shared among the clones of the same Microservice when scaling. Thus, the persistence layer is shared between the clones[57].

To ensure the needed flexibility there needs to be built-in low code capabilities. This makes it possible to easily change data and process model. Repository and modeling engine help to easily achieve customizations without having to hard code on the application layer nor having to directly change the persistence layer. After roughly two decades of experience with data modeling in PLM

[55] See [24].

[56] To read the full list, please see Figure 14.

[57] See also fig. 18.

systems, the low-code data modeling functionality should include additional object-oriented concepts that legacy PLM-systems only support to a certain level. This could mean (multi-) inheritance and instanciation, property libraries, auto-generated forms for the UI. The OMG's modeling languages stack around UML/SysML[58] and for the process model BPMN[59] can serve as a role model how to include graphical modeling into a low-code engine[60]. It does not mean that SysML and BPMN are the only possible standards out there. Most of the time the steep learning curve especially for SysML will hinder its adaption and therefor some light-weight graphical modeling needs to be in place.

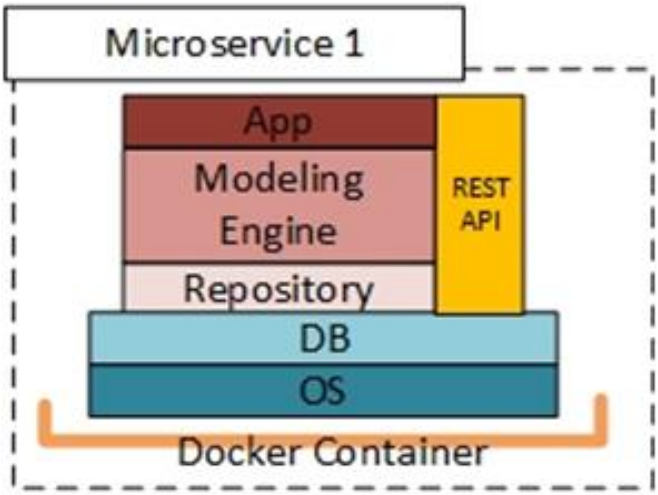


Figure 18 – Generic Microservice

In addition to the definition of the Microservices, one of the most important aspects is the interoperability and communication between these services. Linked Data, as shown in the middle layer of Figure 14, plays an important role here that can be used as a blueprint for data exchange, an obvious example technology as already introduced is OSLC[61]. What needs to happen is that additional domains must be included in the shared ontology. This helps to set up the described knowledge graph and helps to query the whole network rather than each database.

58 Unified Modeling Language / System Modeling Language

59 Business Process Model Notation

60 A good example here is Mendix Studio / Mendix Studio Pro for rapid app development. See www.mendix.com

61 Open Services for Lifecycle Collaboration

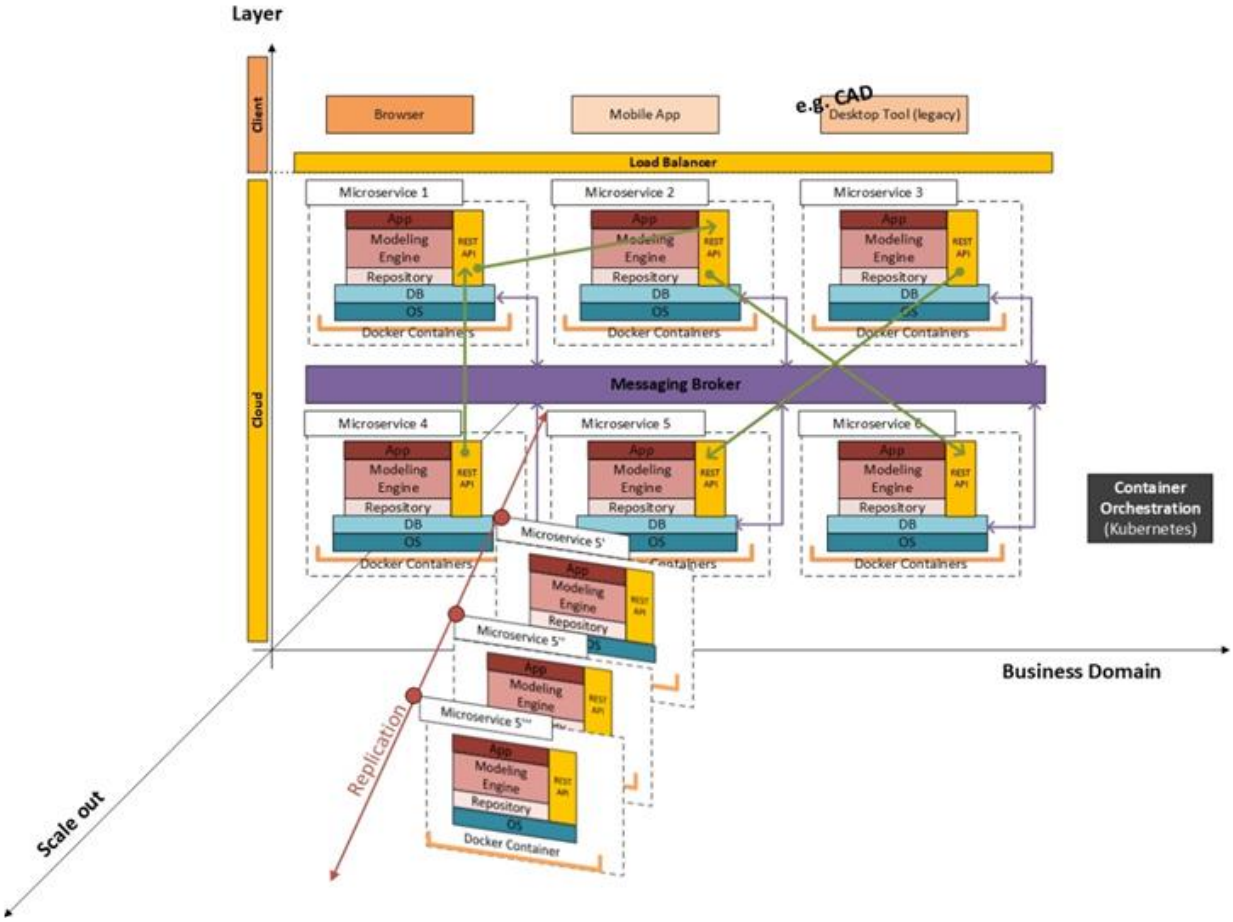


Figure 19 – Generic Cloud Architecture

As shown in Figure 19, Microservices are cloud-native, which not only increases the scalability of services, but also facilitates flexibility in addressing new requirements at different lifecycle stages of a system and in different business contexts. By breaking down a monolithic software into smaller bounded contexts, the scalability of each Microservice can be increased independently and as needed. In addition, the release cycle of Microservices can be handled pretty much independently, allowing to respond more quickly to new needs and requirements and giving the DevOps team the necessary level of autonomy.

To ensure that data is synchronized between the many Microservices the messaging broker will help to distribute data and changes through the network of connected data bases. This is especially important when datasets are needed at more than just one Microservice. Linking via REST-APIs [62]and semantic technologies such as OSLC helps to build the knowledge graph. Brokers are needed to help distribute data sets among the network in order to make more detailed operations running

62 See also [30]

on that data work sufficiently. It should be understood that it is never a decision about linking or synchronization, it must be ensured that both concepts are used where they are needed in combination.

Depending on the persistence strategy there can be multiple different types of databases (one per Microservice) – this of course also means that the principles they run on can differ. Where for many classical PLM tasks ACID principles (see also Sagas in [10]) apply, BASE principles may play an important role when it comes to IoT use cases with a tremendously higher amount of data and update frequency.

Docker Containers are managed by Container Orchestration systems such as Kubernetes (see [22]). There are of course multiple possible clients imaginable: Web clients and Mobile Apps. But it may also be required to connect to existing desktop tools using proprietary connectors. This is needed as long as not all authoring systems have moved to a cloud infrastructure. There is also the necessity to balance the load coming from the clients – e.g. via Load Balancers.

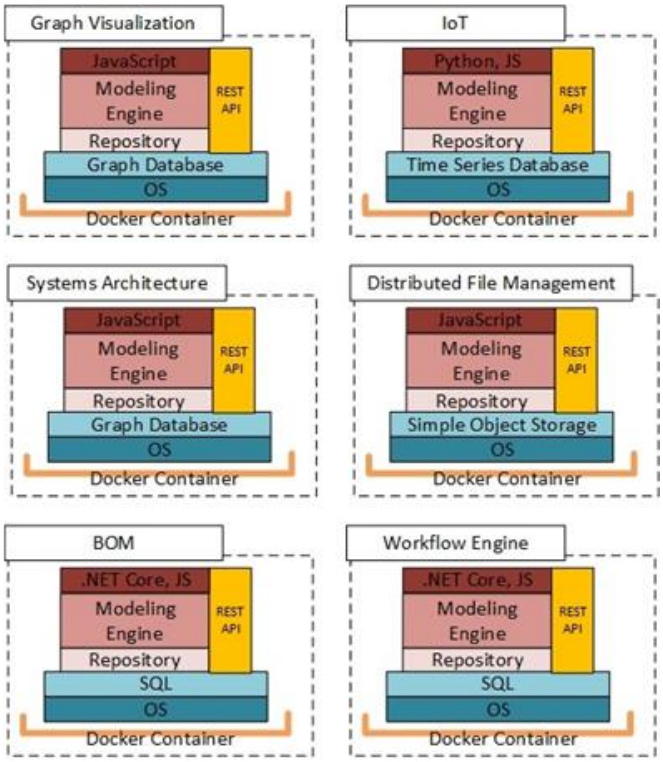


Figure 20 – Technology stacks for certain example Microservices

Fig. 20 gives an overview of six example Microservices and their corresponding technology stack. Where the managed data structures are complex and not easily to be brought into a hierarchy,

graph databases are best suited. Querying a complex graph with such a database will deliver answers fast because it will not need to join tables as regular SQL databases would need to do. This applies to a so called graph visualization Microservice[63] as well as to a systems architecture Microservice. There are already PLM start-ups making use of graph databases[64].

When there is a lot of data to be collected and analyzed Time Series Database are a possible best fit. The most obvious use case in the engineering domain is around IoT. Collecting data from the field to find correlation in it to drive system enhancements is probably the most cited use case[65].

File Management will also probably never stop playing an imported role. Moving existing PLM-concepts for distributed file management (DFM) to a cloud infrastructure will mean making use of object storages such as AWS' simple object storage service Amazon S3[66]. This is a cloud platform service that can be adopted to fulfill the need to distribute large files (such as CAD models) globally.

For other pretty well-established services such as BOM Management and defining and executing workflows it may just mean to stick to regular SQL databases.

An optional component and something that needs additional discussion is the introduction of a central link repository. Currently the shown concept does not manage links centrally but rather lets users create, update and delete those links manually. It is imaginable that in the future there will be the need for a central instance that manages links in order to achieve a higher level of consistency. This discussion originates in the initial creation of the WWW, where the introduction of a central link database wasn't agreed on. Back then consistency has been sacrificed in order to create a highly scalable system (see [11]). For an enterprise use case it may be necessary to rethink that concept as configuration management also for product liability purposes plays a tremendously more important role in the engineering domain than it plays in the public internet.

In addition, the assumption is that a lot of needed capabilities that are not engineering specific may just come from cloud platform services. The most obvious examples would be file management, access control, authentication, reporting and a couple of other capabilities. These may differ from cloud-to-cloud provider but will in general be nothing specific to engineering use cases and can therefore be seen as available.

63 See [29].

64 Ganister PLM is a good example of graph-based PLM, see www.ganister.eu

65 Predictive Maintenance and Closed Loop Engineering should be mentioned here.

66 In other cloud environments there are similar services available.

7 Outlook

Nothing is certain, except for change itself. The enterprise IT industry is changing rapidly. The introduction of Cloud Computing and its adaption constitutes a major shift, not just in software terms but more importantly in the way business is done. The authors of this paper wanted to illustrate what challenges are currently faced by the industry and what solution concepts are being discussed. The introduced concept of cloud-native Systems Lifecycle Management has been discussed to illustrate the ongoing struggle to find a solution and how different components of a possible solution may fit together. It is with no means the only possible outcome. Additional things need to be evaluated, dependencies to existing legacy systems will probably play also a huge role. All eventualities can just not be foreseen in totality. Still, we wanted to leave the reader with a couple of statements we were able to condense out of the intense discussions we had when working on this paper.

10 Statements for the future:

1. Microservices are an architectural prerequisite for agile DevOps processes due to the fact that team autonomy can only be achieved if dependencies within a software's architecture can be brought under control.
2. Without openness of the involved IT applications digitalization cannot be scaled to an enterprise level. Most use cases currently discussed for AI, Data analytics and IOT will just not work if this openness isn't ensured.
3. MBSE needs access to engineering data and mature lifecycle, configuration and data management capabilities. Configuration management is way more than just versioning and baselining.
4. Microservices give freedom to choose the best technology – be it UI, persistence, or programming language – to solve the domain problem at hand in the best way.
5. Decentralized Environments need semantic technologies to query not just one database, but a network of databases. Linked Data in contrast to the original Semantic Web is a pragmatic approach focusing on easy to understand and therefor easy to scale principles.
6. A Microservice-based Architecture needs in addition to Linked Data technologies also asynchronous data exchange for example by using message brokers. Linked data and messaging brokers may serve as the necessary middleware for cloud-native SysLM .
7. Model-Based Systems Engineering and the system model can serve as an interdisciplinary metamodel. It cannot achieve anything beyond a pilot-project if it is not connected to existing engineering data that lives within distributed engineering tools.
8. Systems Modeling itself will eventually also move to a cloud infrastructure, making it necessary to support the authoring of a system model within the web client.
9. Data Protection and Privacy laws will require the establishment of local data centers as moving sensitive (and personal) data between jurisdictions will be inhibited. For example between the US and the EU.

10. The open API philosophy will be strengthened in the years to come also by legislation and precedent cases. Closing APIs to prevent competition can already be seen as distortion of competition. Legislation is slowly but effectively moving into a direction where openness and interoperability is more important and of higher economical and social value than copyright protection of API terminology.

8 References

[1] P. D. H. Kagermann, P. D. W. Wahlster and D. J. Helbig, "Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0," acatech, Frankfurt, 2014.

[2] M. Künzel, J. Schulz and P. Gabriel, "Engineering 4.0 - Grundzüge eines Zukunftsmodells," VDI/VDE Innovationen + Technik GmbH, Berlin, 2016.

[3] A. Alharthi, V. Krotov and M. Bowman, "Addressing barriers to big data," *Business Horizons,* vol. 60, no. 3, pp. 285-292, 2017.

[4] T. Dickopf, A holistic methodology for the development ob cybertronic systems in the context of the Internet of Things, Kaiserslautern, 2020.

[5] M. E. Porter and J. E. Heppelmann, "How Smart Connected Products are transforming Competition," *Harvard Business Review,* November 2014.

[6] E. A. Lee, S. Bateni, S. Lin, M. Lohstroh and C. Menard, *Quantifying and Generalizing the CAP Theorem,* ArXiv, 2021.

[7] R. Vettor and S. Smith, *Architecting Cloud-native .NET Apps for Azure, v 1.0,* Redmond, WA, 2020.

[8] S. Newman, Monolith to Microservices - Evolutionary Patterns to Transform your Monolith, Sebastopol, Ca, 2020.

[9] M. Naab, S. Braun, T. Lenhart, S. Hess, A. Eitel, D. Magin, R. Carbon and F. Kiefer, "Why data needs more attention in Architecture Design - Experiences from Prototyping a Large-Scale Mobile App Ecosystem," in *IEEE/IFIP Conference*, Rio de Janeiro, 2015.

[10] H. Garcia-Molina and K. Salem, "Sagas," *ACM Sigmod Record,* vol. 16, no. 3, pp. 249-259, December 1987.

[11] T. Berners-Lee, Weaving the Web, the original design and ultimate destiny of the world wide Web, New York, 2000.

[12] B. Nowack, "The Semantic Web – Not a Piece of cake…," 08 07 2009. [Online]. Available: http://bnode.org/blog/2009/07/08/the-semantic-Web-not-a-piece-of-cake.

[13] R. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Irvine, CA, 2000.

[14] M. Pfenning, Durchgängiges Engineering durch die Integration von PLM und MBSE, Dissertation, Kaiserslautern: Technical University Kaiserslautern, 2017.

[15] T. Weilkiens, "SysML v2 – The Next Generation. REConf 2019," in *REConf*, München, 13.03.2019.

[16] T. Gilz, Integrated Interdisciplinary System Models in the Conceptual Design Phase Based on Model-Based Systems Engineering, Dissertation, Kaiserslautern, Germany: Technical University Kaiserslautern, 2014.

[17] T. Dickopf, M. Eigner and H. Apostolov, "The KSCM as part of a holistic methodology for the development of cybertronic systems in the context of engineering 4.0," in *International Design Conference* , 2019.

[18] M. Eigner, T. Dickopf and H. Apostolov, "Interdisziplinäre Konstruktionsmethoden und -prozesse zur Entwicklung cybertronischer Produkte – Teil 2," *Konstruktion,* 01 02 2019.

[19] Y. Hooshmand, D. Adamenko, S. Kunnen and P. Köhler, "An approach for holistic model-based engineering of industrial plants," in *21st International Conference on Engineering Design (ICED17)*, Vancouver, 2017.

[20] Y. Hooshmand, D. Adamenko, S. Kunnen and P. Köhler, "Semi-Automatic Creation of System Models based on SysML Model Libraries," in *INCOSE EMEA Sector Systems Engineering Conference*, Berlin, 2018.

[21] C. Muggeo and M. Pfenning, "Durchgängigkeit des Engineerings," *Computer and Automation,* 2016.

[22] G. Sayfan, Mastering Kubernetes: Large scale container deployment and management, Packt Publishing, 2021.

[23] M. L. Abbott and M. T. Fisher, The art of scalability, Crawforsville, In, 2015.

[24] M. Eigner, System Lifecycle Management, Digitalisierung des Engineering, Springer VS, 2021.

[25] S. Tilkov, M. Eigenbrodt, S. Schreier and O. Wolf, REST und HTTP; Entwicklung und Integration nach dem Architekturstil des Web, dpunkt, 2015.

[26] S. Edlich, A. Friedland, J. Hampe and B. Brauer, NOSQL: Einstieg in die Welt nichtrelationaler web 2.0 Datenbanken, vol. 2, Hanser, 2011.

[27] R. Petrasch and O. Meimberg, Model-Driven Architecture: Eine praxisorientierte Einführung in die MDA, dpunkt, 2006.

[28] N. Narkhede, G. Shapira and T. Palino, KAFKA – the definitive guide. Real-time data and stream processing at scale, O´Reilly, 2017.

[29] R. Brath and D. Jonker, Graph Analysis and Visualization - Discovering Business Opportunity in Linked Data, 2015: Wiley.

[30] T. Eickhoff, A. Eiden, J. C. Göbel and M. Eigner, "A Metadata Repository for Semantic Product Lifecycle Management," in *CIRP Design*, 2020.

Document Information

Title:	Killing the PLM Monolith – the Emergence of cloud-native System Lifecycle Management (SysLM)
Date:	May 30, 2022
Status:	Final
Classification:	Public