Digitale Schiene
Deutschland

2021-11-18 | Capella Days 2021

An example of model-centric engineering environment with Capella and CI/CD

Viktor Kravchenko / Systems Engineering Toolchain Team / DB Netz AG, I.NAT22

Index

- → What Digitale Schiene Deutschland does (brief intro)
- → Our MBSE challenges (some of)
- → What is CI/CD in MBSE context?
- → Model-centric engineering environment
  - Automation of repetitive tasks via rule-based **model modifiers**
  - Artefact generation (**mddocgen**)
  - Workflow automation – wiring it all together with CI/CD
  - Sandbox for research & discovery of automation opportunities (talking to Capella models from **Jupyter**)

Government and society expectations towards the future of the railway system

Digitale Schiene Deutschland

- **2x growth in passengers** by 2030
- **Increasing** modal split of **rail freight** up to 25 %
- Measurable contribution to **CO_2 reduction goals**
- Significant **improvements** in **punctuality**
- Preparing for **Deutschlandtakt**

- In order to meet the expectations with regard to shifting traffic to rail, we need **to increase rail capacity by up to 35%.**
- Along with the physical expansion, **technological innovation** and the **digitalization of the railway operations** are the biggest factors when it comes to increasing capacity
- Making this lever **available to the rail system** is the mission of Digital Rail for Germany

Faster, greener, smoother – with Digital Rail for Germany

Digitale Schiene Deutschland

More trains, same tracks – Digital Rail will increase **network's capacity by up to 35%**

Standardised technical components and optimal control of all processes will **increase reliability**

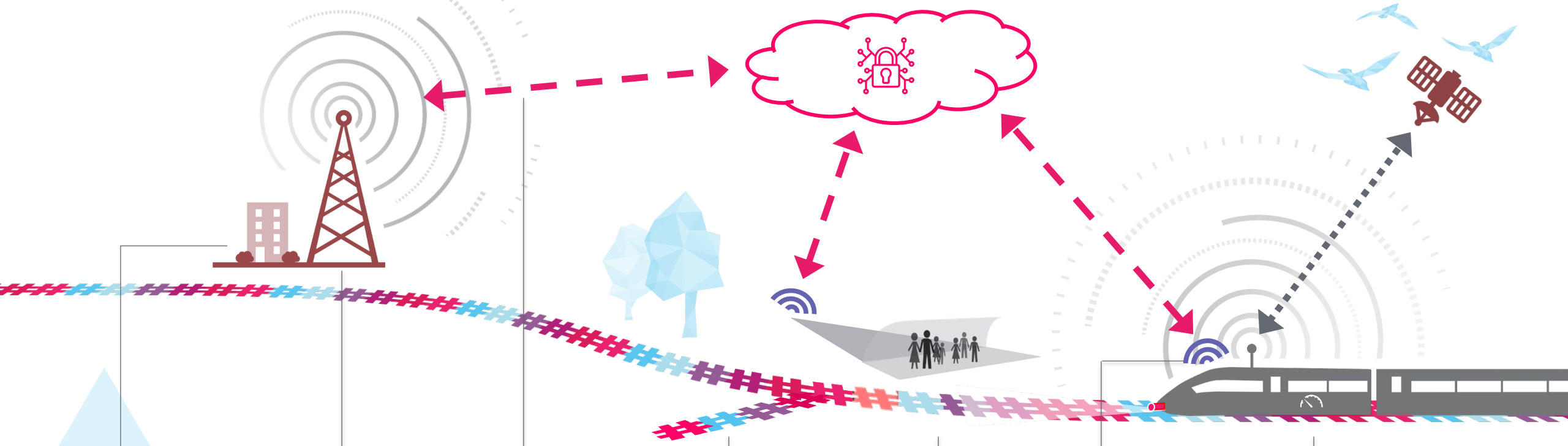
Virtualization of **'hardware'** will **reduce infrastructure to be maintained**

Digital Rail will **boost new tech** in German rail industry and promote **future-oriented skills**

While carrying more passengers and goods, Digital Rail will **emit less CO_2 by 1.6 million tons p.a.**

Major challenge is the development and interaction of the essential future technologies

Digitale Schiene Deutschland



AI in traffic and incident management

FRMCS/5G & fiber-optic networks

Cloud und IT-Security

Digital Signalling System (ETCS based)

Sensor based environment monitoring

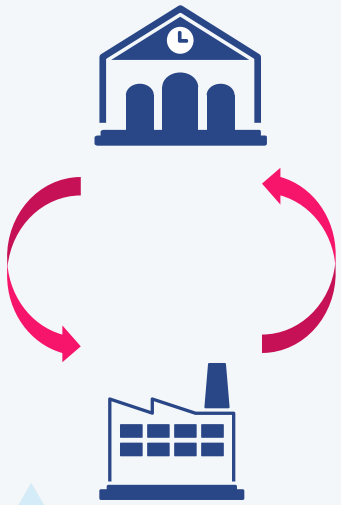
Sensor based localization & digital map

Automatic Train Operation

Digitizing the railway system requires new partnership models with operators and industry

Digitale Schiene Deutschland

OPERATORS



INDUSTRY

Basic principles of Digitale Schiene Deutschland

- **Fast validation** of requirements and **testing** in **collaborative projects** with industry partners
- **Standardization** and **harmonization** at **European level** with other rail operators

TODAY		FUTURE
From technology user	→	To active co-developer
From closed systems	→	To open platforms

Learn more here: https://digitale-schiene-deutschland.de/en
(or follow the QR link)

System Engineering Challenges

Digitale Schiene Deutschland

Create a valid **Reference Architecture** of a railway system and **specify the** key **building blocks**
Ensure interoperability with international partners by aligning on core concepts and interfaces

Define Stakeholder Needs and Environment

Capture and consolidate operational needs from stakeholders
Define what the users of the system have to accomplish
Identify entities, actors, roles, activities, concepts

Formalize System Requirements

Identify the boundary of the system, consolidate requirements
Define what the system has to accomplish for the users
Model functional dataflows and dynamic behaviour

Develop System Logical Architecture

See the system as a white box
Define how the system will work so as to fulfill expectations
Perform a first trade-off analysis

Develop System Physical Architecture

How the system will be developed and built
Software vs. hardware allocation, specification of interfaces, deployment configurations, trade-off analysis

Capture the domain needs in a **solution-agnostic** way, agree on abstract concepts with international partners

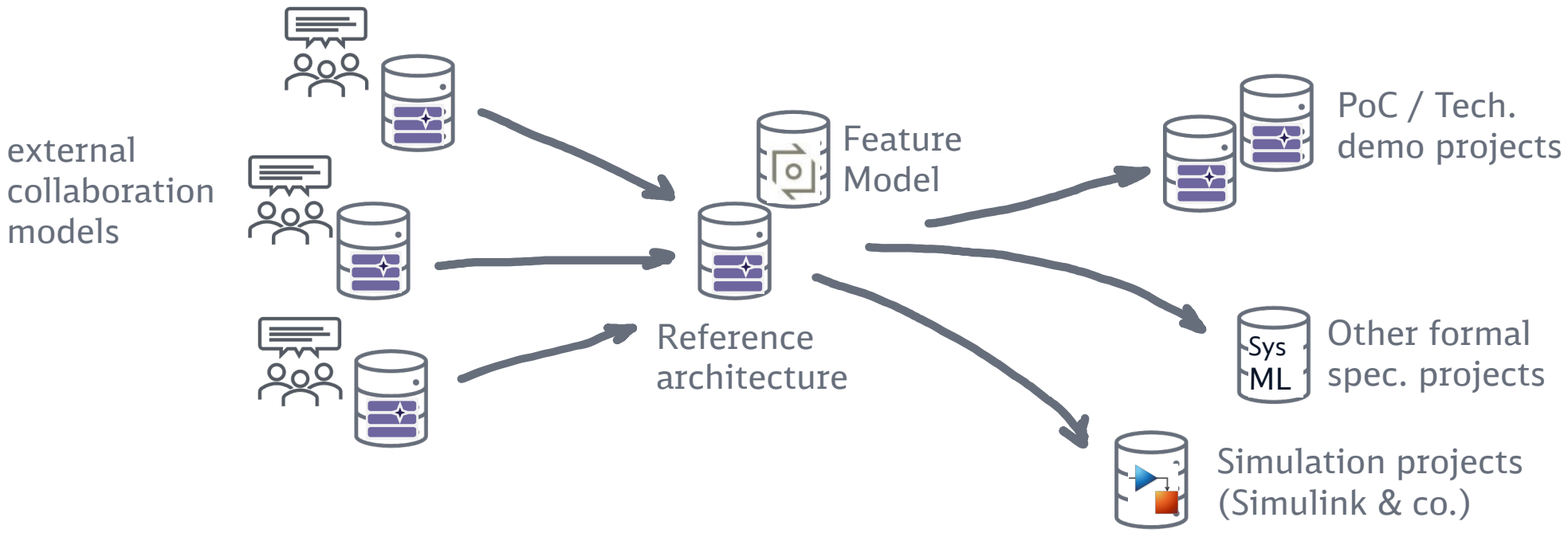
Validate key assumptions on technologies and architectural concepts via real **prototypes** and demonstrators / achieve TRL

Produce **subsystem** specifications and blueprints so that industrial partners could develop end products

Challenge: model together with external partners

Digitale Schiene Deutschland

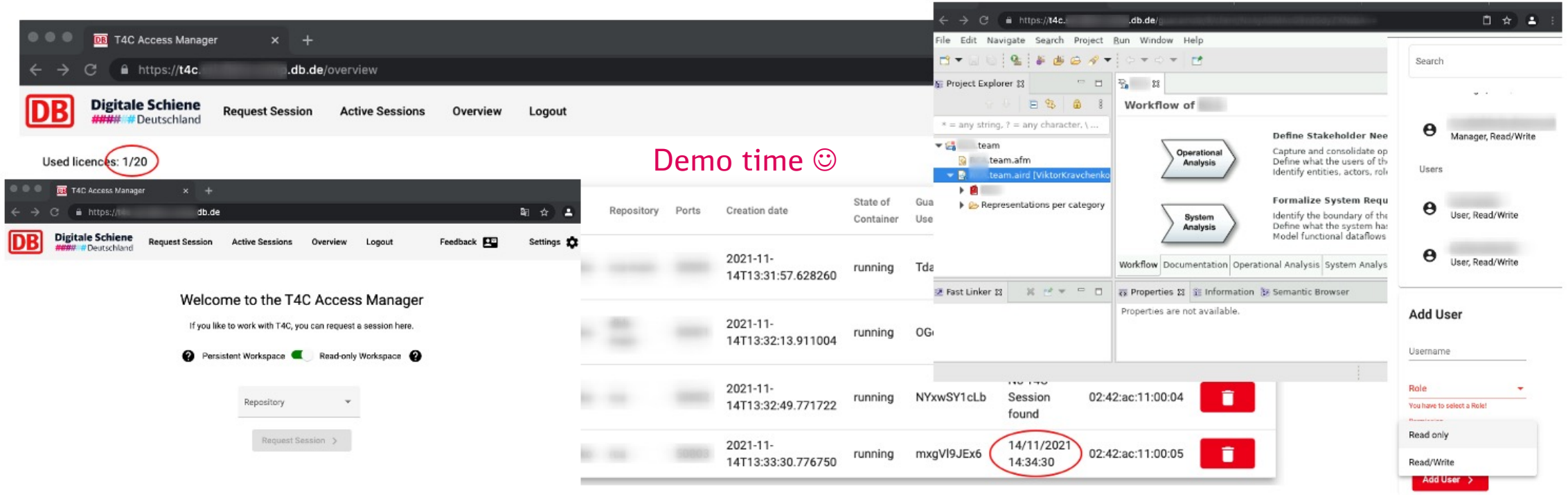
- We have a number of **models for external collaboration and alignment** around our reference architecture
- We work on those models with external partners that have **own IT security constraints**
- The **pace of modelling needs to be fast** enough for workshop environment / no time for merge conflict resolution
- The teams are big and it is difficult to manage access in a “classical” way → need for **self-service access management**



Solution: managed TeamForCapella (T4C Web Access)

Digitale Schiene ##### Deutschland

- Allows read or read/write access to **TeamForCapella models in a browser** (without overhead of shared VMs)
- Runs **cloud-native in a Kubernetes cluster**, every user session gets an own HW resource allocated, recycled on closure
- **Model owner can grant access** or delegate access management to other users; no local accounts thanks to OAuth2
- **Read only users** (for review) **use no** TeamForCapella **licenses** – we auto-clone the latest model from the git mirror



Challenge: develop tech demonstrators at a pace

Digitale Schiene Deutschland

To validate a reference architecture **concept** / assess technology readiness, **where simulation** or theoretical analysis **is insufficient,** more difficult to setup or requires initial field data,
we create technology demonstrators / proof-of-concept projects

→ Projects usually require very high pace – no time or people for classical “paperflow” engineering

→ Systems design team on a project is usually tiny when compared to other engineering functions

→ need to integrate large number of non-SE stakeholders with a very small SE team

→ Yet when things go on public tracks we need to be compliant with EU and national regulations.

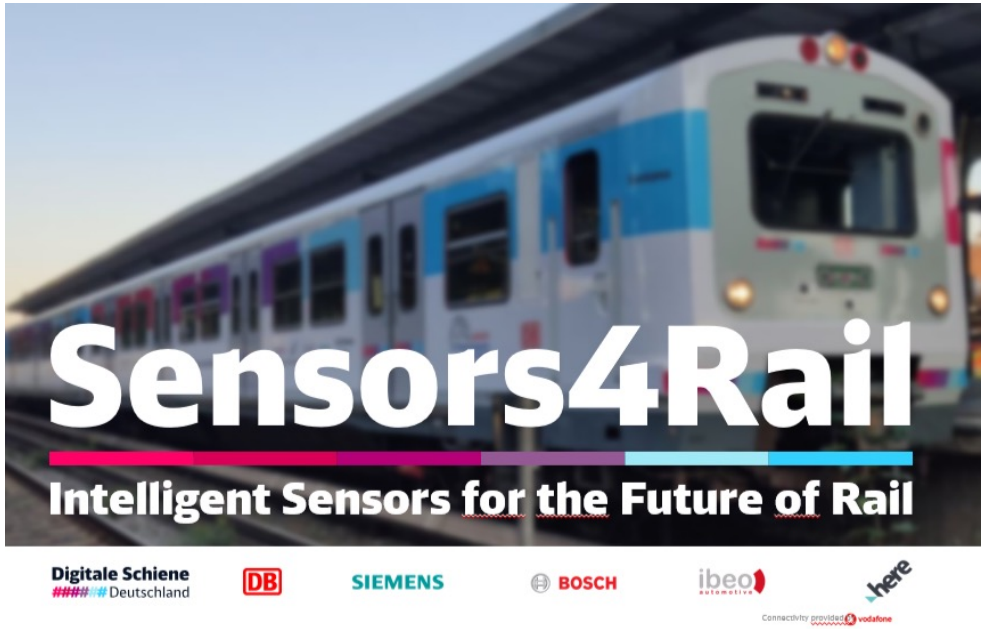
In short that also means:

- **System requirements and solution were developed to a valid process**
- **The requirements for the system were identified and are valid**
- **Solution satisfies the requirements**
- **Problem and solution documentation is consistent / no uncontrolled changes broke it**
- **The resulting system will do no harm**

→ need for rigorous analysis, SE, V&V workflows and Change Control

Application examples: Sensors4Rail toolchain

Digitale Schiene Deutschland



Sensors4Rail: Intelligent sensor technology has everything in view

- **Environment detection**: Equipment of a train of the S-Bahn HH with current camera/radar/lidar and localization technology
- **Demonstrated** landmark-based localization, train detection, pedestrian detection, track detection, localization consolidation, occupancy detection
- **Real-time data streaming** to cloud for further processing
- **Outlook:** will collect operational data / perform 4-seasons evaluation until 2023

Architected with Capella, change-managed in Jira

Model configuration controlled in git:

- 990 commits
- 12 model release tags
- 90 CI/CD artefacts

Model metrics:

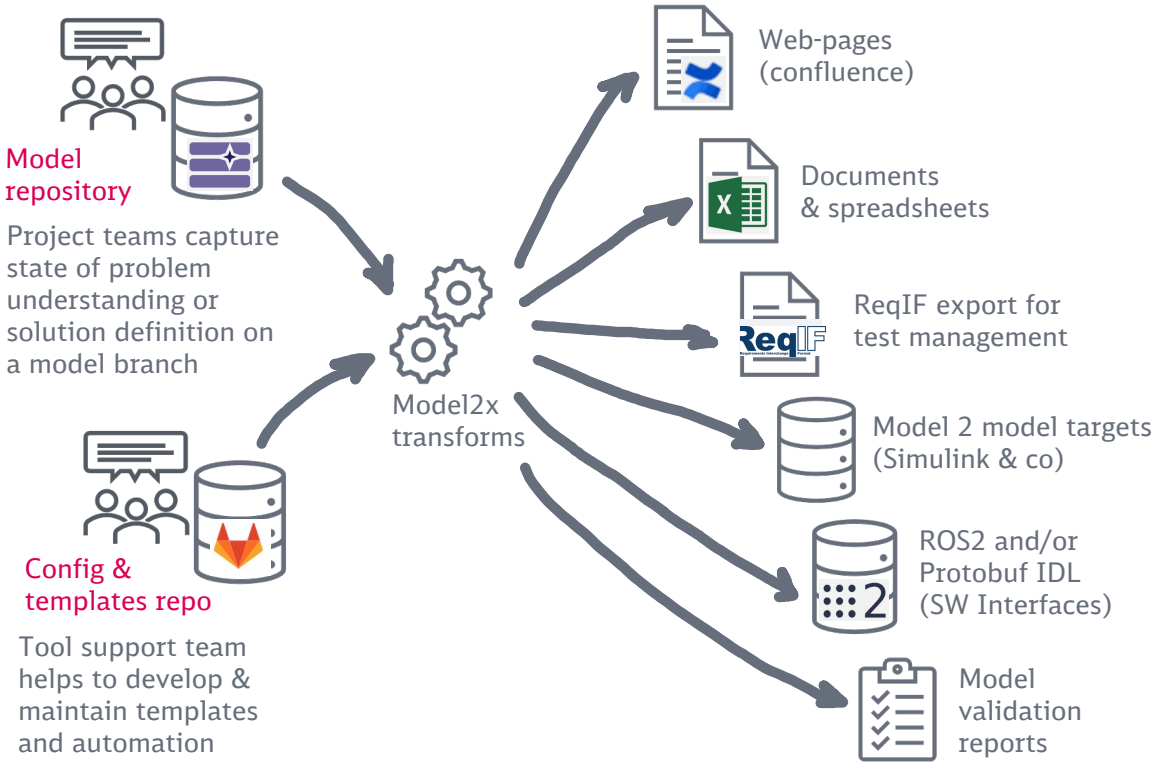
- 23 system capabilities, about 110 logical functions
- 170+ physical parts (LRUs), 64 unique software interface definitions
- 243 wiring definitions (part.port to part.port + cable type and other metadata)

You may learn more about the project here: https://digitale-schiene-deutschland.de/en/Sensors4Rail-ITS2021

Solution: model-centric engineering environment

Digitale Schiene Deutschland

- Model is placed in the middle of the project life.
- Most of the aspects of problem and solution definitions are captured in the model
- Model configuration is managed via git, changes are worked on feature branches and refer to development tasks
- Project documentation and technical artefacts are derived from the model



Non MBSE stakeholders review derived **work products in the target form** and close the loop by providing feedback to the MBSE team

Model as a source for artefact generation **guarantees consistency of all** model-**derived artifacts** and reduces the workload of the entire project team

What is CI/CD in the MBSE context?

Digitale Schiene Deutschland

In software development, CI/CD is a method to **frequently integrate & deliver** software to end users:

- **Continuous Integration** (CI) is a method that puts a software product together and runs it through quality gates
- **Continuous Delivery** (CD) gets those builds that made it through quality gates and delivers them into operations

In short – it's **a method to automate** build, quality assurance and delivery of software products

Applying CI/CD to an MBSE project may look like this (very simplified):

Apply **model modifiers** – run repetitive tasks that an engineer would do if The Machine wasn't there

Check if the resulting model is good enough to keep going (rules & consistency checking)

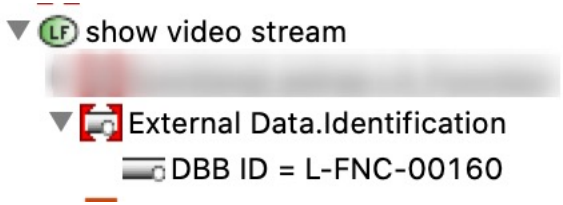
Build & deliver artifacts

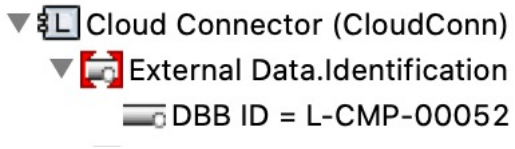
Model Modifiers: unique identification of key elements

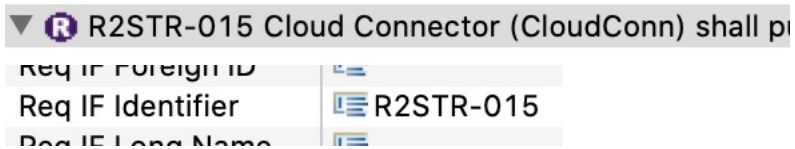
Digitale Schiene Deutschland

Need:
add human-friendly unique identifier to key model elements so that these could be referenced in documentation and project communications

Solution:
apply attribute extension via PVMT (or ReqIF.Identifier) and maintain attribute externally (on git commit);
protect from accidental modification by a redundant attribute storage (adjacent JSON “database”)







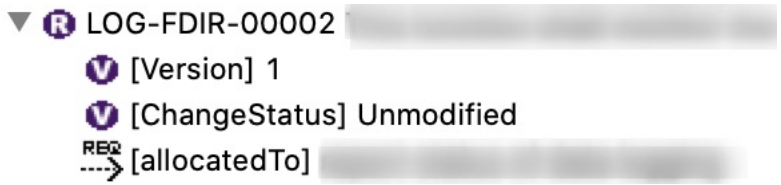
Model Modifiers: requirement change status assessment

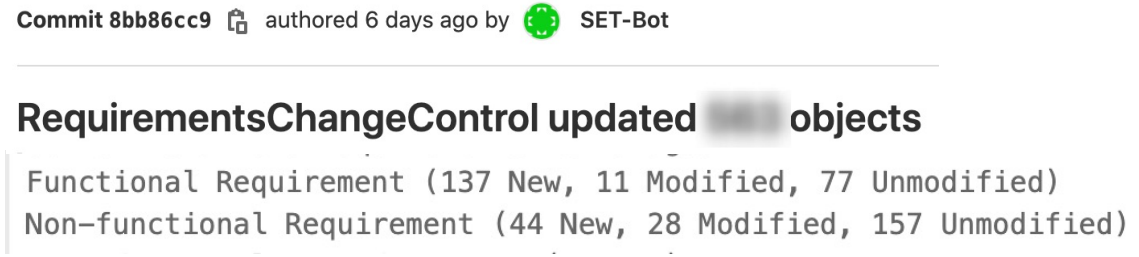
Digitale Schiene Deutschland

Need:
Compare requirements in the current model and the last released model (pre-configured git tag), then:

Condition pseudo-code	ChangeStatus	Version
current.text != released.text	Modified	released.version + 1
current not in released model	New	1
Current.text == released.text	Unchanged	released.version

Solution: compare models versions and update element attributes automatically on commit; store change assessment summary





Model Modifiers: req-bot

Digitale Schiene Deutschland

Need:
As our models evolve we see **modelling patterns** and also how those should be represented in natural text / **formal requirements**. We need a way to describe those patterns and "spell-out model" as natural language requirement objects for non-MBSE stakeholders; Same patterns may apply to synthesis of "rationales" – statements that justify requirements. We also need the solution to **manage lifecycle of generated requirements** – so they are created, modified and deleted in alignment with the **model changes**.

Solution: Requirements Bot (req-bot) that consumes generic and project-specific requirement pattern definitions and maintains natural language representations and traceability of those in the model.

```
@dataclass(frozen=True)
class ReqPatternInstance:
    id_prefix: str  # prefix for the req, like "FSR" - used as a key for abs number assignment / counter increment
    derived_from: T.Sequence[
        GenericElement
    ]  # list of key elements that the req was derived from and can't exist without
    text: str  # natural language representation of the model-based requirement pattern
    target: str  # "OA/Generated Requiremnts/Functional/Persistent behavior" -> layer, module, subfolder, subfolder
    req_type: str  # name of the type, i.e. "Non-functional Requirment"
    allocated_to: GenericElement  # the element that the req instance is linked to
    allocation_link_type: str  # name of the link type, defaults to "allocatedTo"
    rationale: str = None # natural language explanation of why the requirement is needed, optional
```

Artefact generation: mddocgen

Digitale Schiene Deutschland

Deriving things from a model requires some templating and an engine to ”inflate” the template with some contents. We honestly gave M2Doc a go but in short – it wasn’t sticking well.

One of the team members had a thought how the templating pain can be reduced and made a PoC of an own generation engine (in about a week). The concept was so cool that we decided to make it work.

The engine uses **python Jinja templates library** → any text / html / markdown file can be a template. We went on and wired this to Confluence so that we could query models directly from Confluence pages.
Or to .pdf via another off-the-shelf html to pdf engine

```
{% set nfrs = subsystem.requirements.exclude_types("Functional Requirement", "Assumpt
```

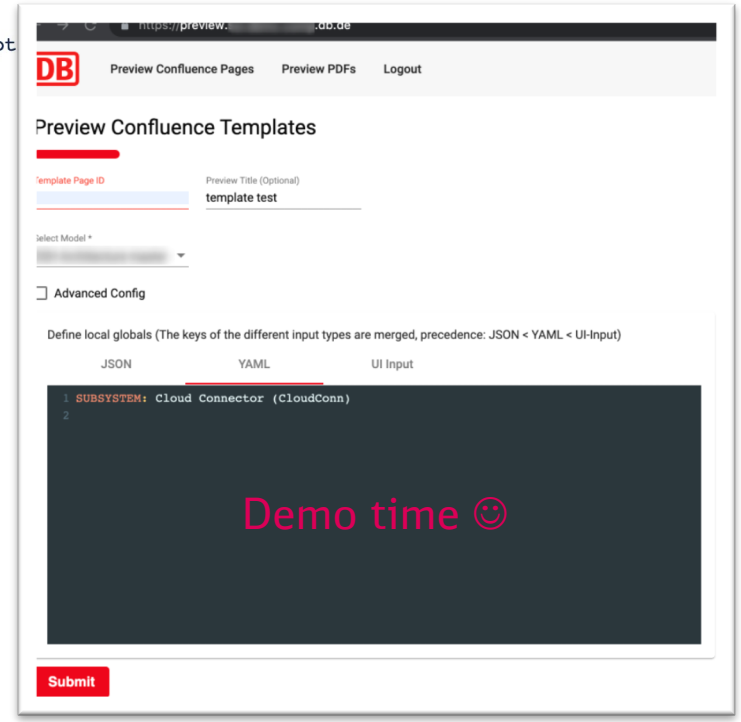
Additional requirements for {{ subsystem.name }}:

{{ req_table(nfrs) }}

```
{% macro req_table(reqs) %}
```

Req.ID	Requirement	Rationale
{% for req in reqs %}		
{{ req_id(req) }}	{{ req.text }}	{{req.attributes["Rationale"]}}
{% endfor %}		

```
{% endmacro %}
```



Artefact generation: mddocgen + Gitlab-CI

Deriving things from a model **requires** some template **design** / **maintenance effort**. This applies to any „thing“ – be it a document, an IDL file or a spreadsheet.

When the templates are in a working state however, generating and delivering derived material is only a matter of wiring things together. That’s where Gitlab-CI comes in.

Machine-friendly „build plan“ acts as an HMI for the generation and delivery process

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # #
#                                                      #
#   SUBSYSTEM SPECIFICATIONS                           #
#                                                      #
# # # # # # # # # # # # # # # # # # # # # # # # # # # #

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
- title: Mission Control (MCTR) Subsystem Specification
  doc_id: [redacted]
  abbrev: Not Applicable
  description: Definition of subsystem border, interfaces and functionalities of
    Mission Control (MCTR) subsystem.
  version: '0.1'
  source:
    engine: confluence-template-getter
    page_id: [redacted]
  globals:
    SUBSYSTEM: Mission Control (MCTR)
    REVIEWERS:
    - Mission Control Lead SW Engineer
    - SW Engineering Lead
    - Project System Architect
  build_targets:
  - engine: confluence-page-maker
    target_page_id: [redacted]
    channel: release-subsys
  - engine: confluence-page-maker
    target_page_id: [redacted]
    channel: preview
  - engine: pdf-maker
    filename: [redacted].pdf
    profile: dbs
    channel: preview
```

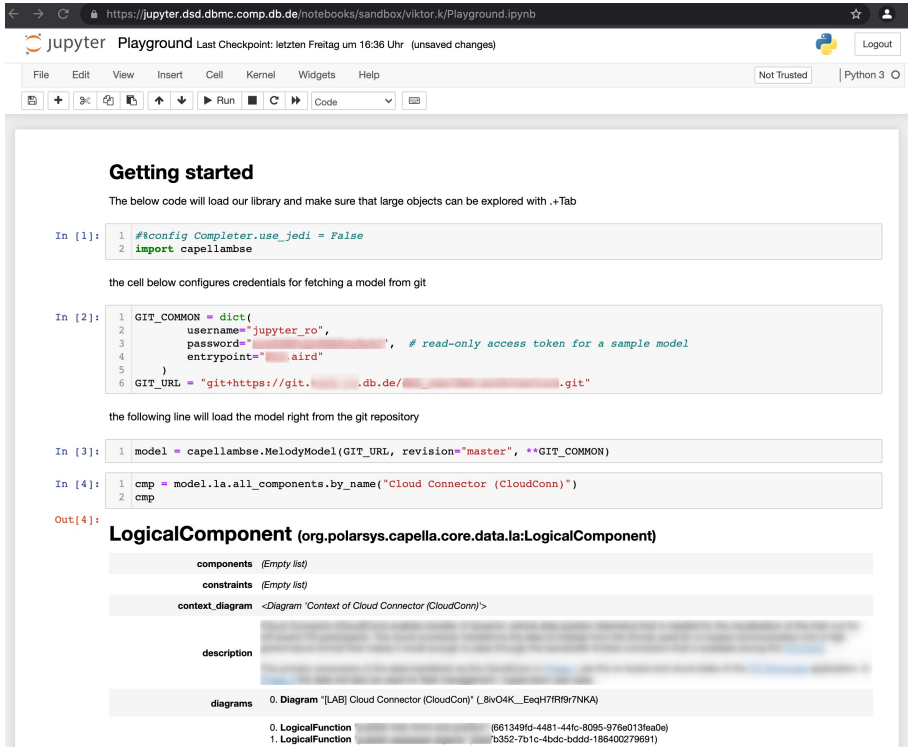
```
$ python3 -m mddocgen.workers jobs/04-export -c ${JOB_CHANNEL} --log-level=$MDDOCGEN_LOGLEVEL || r4=$
2021-11-05 16:07:58,404: INFO     Processing 72 jobs in jobs/04-export
2021-11-05 16:07:58,734: INFO     [1/72] Job GENERATED_0000.job.json finished, status: SUCCESS
2021-11-05 16:07:58,737: INFO     [2/72] Job GENERATED_0001.job.json finished, status: IGNORED
2021-11-05 16:07:58,752: INFO     [3/72] Job GENERATED_0002.job.json finished, status: FAILED
2021-11-05 16:07:58,767: INFO     [4/72] Job GENERATED_0003.job.json finished, status: FAILED
2021-11-05 16:07:58,768: INFO     [5/72] Job GENERATED_0004.job.json finished, status: IGNORED
2021-11-05 16:08:00,258: INFO     [6/72] Job GENERATED_0005.job.json finished, status: SUCCESS
2021-11-05 16:08:00,259: INFO     [7/72] Job GENERATED_0006.job.json finished, status: IGNORED
2021-11-05 16:08:00,285: INFO     [8/72] Job GENERATED_0007.job.json finished, status: FAILED
2021-11-05 16:08:00,311: INFO     [9/72] Job GENERATED_0008.job.json finished, status: FAILED
2021-11-05 16:08:00,346: INFO     [10/72] Job GENERATED_0009.job.json finished, status: FAILED
2021-11-05 16:08:00,347: INFO     [11/72] Job GENERATED_0010.job.json finished, status: IGNORED
2021-11-05 16:08:01,318: INFO     [12/72] Job GENERATED_0011.job.json finished, status: SUCCESS
2021-11-05 16:08:02,691: INFO     [13/72] Job GENERATED_0012.job.json finished, status: SUCCESS
2021-11-05 16:08:03,939: INFO     [14/72] Job GENERATED_0013.job.json finished, status: SUCCESS
2021-11-05 16:08:03,940: INFO     [15/72] Job GENERATED_0014.job.json finished, status: IGNORED
2021-11-05 16:08:05,186: INFO     [16/72] Job GENERATED_0015.job.json finished, status: SUCCESS
2021-11-05 16:08:09,558: INFO     [17/72] Job GENERATED_0016.job.json finished, status: SUCCESS
2021-11-05 16:08:12,524: INFO     [18/72] Job GENERATED_0017.job.json finished, status: SUCCESS
2021-11-05 16:08:13,386: INFO     [19/72] Job GENERATED_0018.job.json finished, status: SUCCESS
2021-11-05 16:08:13,388: INFO     [20/72] Job GENERATED_0019.job.json finished, status: IGNORED
2021-11-05 16:08:14,366: INFO     [21/72] Job GENERATED_0020.job.json finished, status: SUCCESS
```

Sandbox for research & discovery of automation opportunities

Digitale Schiene Deutschland

For doing all that templating / generation we need a place where we could “**talk**” **to the model**.
And what could be better than **Jupyter Notebooks** with all the features that Python ecosystem brings?!

We run a **hosted Jupyter instance** with shared spaces so that people could open the link and start “playing” without any installation overhead + learn from each-other’s work



Demo time ☺

The below code shows how a PVMT attribute can be retrieved

```
In [5]:
rows = []
for fnc in model.la.all_functions:
    rows.append({"Fnc ID":fnc.pvmt['External Data.Identification.DBB ID'], "Fnc. name": fnc.name})
pd.DataFrame(rows)
```

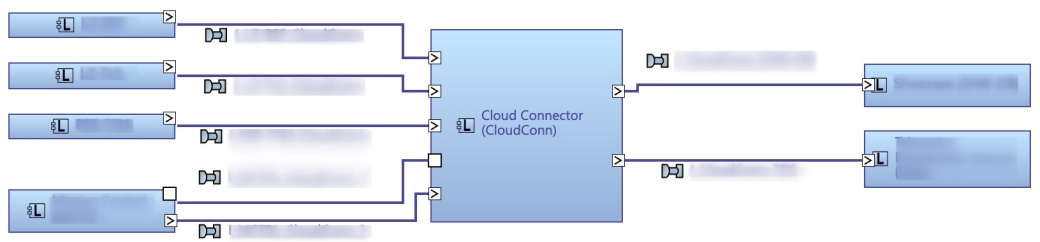
Out[5]:

	Fnc ID	Fnc. name
0	L-FNC-00144	
1	L-FNC-00145	
2	L-FNC-00146	
3	L-FNC-00147	
4	L-FNC-00148	
...	...	
102	L-FNC-00247	
103	L-FNC-00248	
104	L-FNC-00249	
105	L-FNC-00250	
106	L-FNC-00251	

107 rows × 2 columns

```
In [5]:
cmp.context_diagram
```

Out[5]:



Thank you!

We thought some of the toolchain tech we developed may be helpful for others, so we made it public:
https://github.com/DSD-DBS/py-capellambse

Also, feel free to get in touch:

Viktor Kravchenko
Circle Lead – System Engineering Toolchain
viktor.kravchenko@deutschebahn.com

Or PM via Linkedin (QR-code below)