Rail4Future

Project title:	**Resilient Digital Railway Systems to enhance performance**
Start date:	**01/04/2021**
Time duration:	**42 months**
Project number:	**882504**
Announcement:	**8. Ausschreibung COMET Projekte 2019**

Deliverable D1.3.6

Report – Automated Model Integration

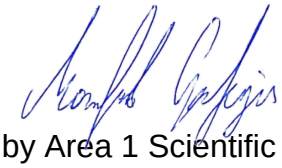
Due date	
Submission date	
Submitted by	TU Wien MIVP

Version	Date	Edited by	Description
0.1	31.07.2024	Ozan Kugu	Creation
0.2	07.08.2024	Manfred Grafinger	Proof reading
0.3	20.08.2024	Stefan H. Reiterer	Edition

Deliverable released

by Area 1 Manager
Dr. Michaela Haberler-Weber

Deliverable released



by Area 1 Scientific Lead
Dr. Manfred Grafinger

1 Executive Summary

In this deliverable, we show and describe how we designed and developed the methodology to automate and manage the entire asset integration and processing, based on different interface standards (e.g., FMI, SysML), by using different key technologies such as graph DB, CI/CD pipeline, artifactory repository management, VCS, CAT and CLI. This is enormously important to gain insights about automation techniques helping to reduce time effort and resource consumption. The assets (model + data) of different railway use cases, which are successfully implemented into the R4F Platform (see Deliverable D1.1.5 and D1.3.5), are automatically built, tested and then deployed to visualization in the platform for the Rail4Future project.

2 Table of Content

3 Abbreviations and Acronyms

Abbreviations / Acronyms	Description
R4F	Rail for Future
MBS	Multi body simulation
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
RLT	Residual Life Time
VTI	Vehicle Track Interaction
ViV	Virtual Vehicle Research GmbH
SSP	System Structure and Parametrization
JSON	JavaScript Object Notation
CSV	Comma-separated values
ML	Machine Learning
PID	Proportional-Integral-Derivative
CLI	Command Line Interface
CAT	Code Analysis Tool
VCS	Version Control System
SCM	Source Code Management
DevOps	Development and Operations
CI/CD	Continuous Integration / Continuous Delivery
PC	Personal Computer
VM	Virtual Machine
DB	Database
DCP	Distributed Co-Simulation Protocol

4 Problem Description / Objectives

4.1 Problem Description

After implementing all railway assets (model + data) into the R4F Platform, it is important to manage and organize the entire DevOps process, including data management (data storage, collection and exchange), result generation, visualization, model integration and exchange, in the platform. For this, relevant know-how and experience about DevOps practices are required to apply these to the platform. Besides, automation is a remarkable aspect to significantly reduce the technical effort and time burden spent for the integration and processing of the assets in the platform. Therefore, the automation of the asset integration and processing would help to make the R4F Platform more time- and cost-efficient for its users.

4.2 Objectives

Based on the problem statement above, this deliverable aims to cover two main subjects: 1) The entire prototype of the R4F Platform; 2) Automated integration and delivery testing of simulation assets in the R4F Platform. In the first subject, the entire IT architecture is proposed, which is used to build the real R4F Platform in future. This architecture includes graph DB, version control, pipeline, code analysis and artifactory repository management technologies, which are used to automatically build, test and deploy all the assets of different railway use cases. The second one shows and demonstrates what kind of approach can be followed to automate and manage the asset integration and processing task in the R4F Platform by using the pipeline, version control and code analysis technologies with the CLI as an alternative (for more information see Kugu et.al. [3]).

5 Significance for the overall Project

For the Rail4Future project, it is important to follow an approach significantly helping users (railway infrastructure managers and train operators) to maintain and monitor their holistic large-scale railway infrastructure system time- and cost-efficiently. For this, the automation needs to be applied to the R4F Platform. Specifically, the available models, belonging to different railway use cases, can be automatically integrated and then visually deployed in the platform with their data for the users. Surely, the automated integration and deployment process needs to be controlled based on the users' feedback as well, which would significantly contribute to the feasibility and resilience of the Rail4Future project.

6 Description

6.1 Graph-DB based Automated Model Integration (R4F Prototype)

For the automation a graph-based approach is used which uses co-simulation process graphs (see [4] and references there). The benefit of co-simulation process graphs over traditional process graphs is that complex co-simulations can also be represented within the process (which is normally not possible due to possible cycles within the graph), and it allows for autoconfiguration and is compatible with common standards like FMI, DCP or SSP (see [4] for further details).

The co-simulation process graph can be stored within a graph database, and it is possible to auto-generate execution pipelines for automation services like Jenkins. This also allows to easily keep track of the processes and co-simulations and allows to combine several processes into one (see [5]) and allows for easier re-usability and reproducibility. This makes it well suited to automate the auto integration and deployment of complex (co-)simulations.

The automation itself is executed by a service implemented in Python and deployed in Docker, which can translate the graphs into pipelines that can be processed by Jenkins.

6.1.1 An Example: The surrogate UC

In Fig.1 we see a simple example of a co-simulation process graph representing the surrogate model use case. The pipeline simply pulls the source code, builds an FMU for the model and starts the simulation. Every stage contains the necessary information for execution like the link to the version control system (in this case Git) or parameters to run the simulation.

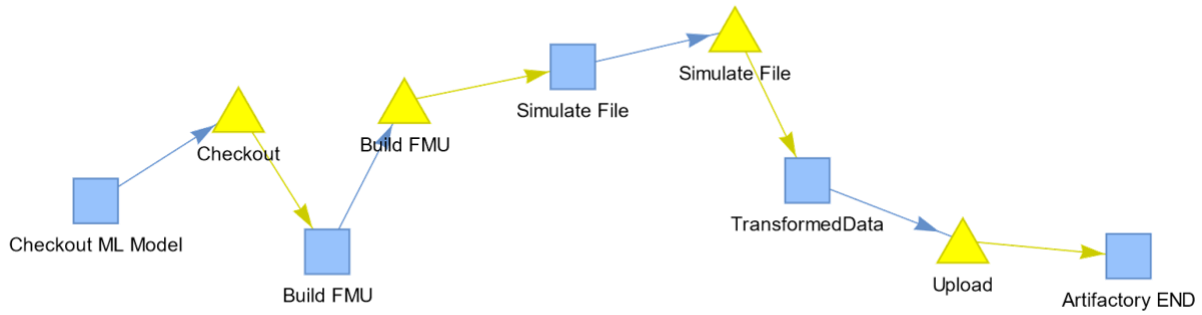


Fig. 1: Example Co-Simulation Process graph for the Surrogate Use Case. (Squares stand for data that has to be transformed or was transformed and triangles for the transformation of this data)

6.2 Automated Integration and Delivery Testing

In this subsection, a simplistic demonstration approach is shown and described, which would help to easily manage and automate the integration and processing of railway simulation assets (parameters, model configurations, model files etc.) in the R4F Platform. These assets belong to the use cases, which are successfully implemented into the platform (e.g., anti-slip co-simulation of a railway vehicle, RLT calculation of a steel bridge). Besides, the approach is successfully addressed into the conceptual R4F Platform consisting of different layers (for more details see Zhou et.al. [2]). During the integration and processing, the assets are built, tested and then delivered to the Infrastructure Layer by using the open-source FMI & SSP interface standards (for more information about the model standardization and simulation approaches see Kugu et.al. [1]), CLI, mailing function, VCS and CAT technologies. For further information about the entire approach, the readers can take a look into the Kugu et.al. [3] to get comprehensive insights about the management and automation of the asset integration and processing task with a use case example (anti-slip traction and vehicle speed control system, co-simulation model).

After succeeding the entire simulation of the assets on the PC, we test the simulation in a Linux VM, which is located in the Integration Layer of the R4F Platform. As first step, we integrate these assets, imported from the Asset Layer, through the use of the FMI and SSP interface standards. Then we simulate them by using the Python tool in Linux bash, where we can directly verify the platform compatibility of these assets. After that, we build and test the asset simulation in our Jenkins pipeline, which is a demonstrative reference of the R4F Platform (see Fig. 2). For this, we as Asset Integrator put all these assets into a software repository in a Git VCS tool, helping us to store and track these assets, and therefore to work with stakeholders collaboratively. This tool can directly connect to our CI/CD Jenkins pipeline through the use of the SCM, so that the pipeline can read, store and run these simulation assets in the Jenkins server. Besides, we preferred to use a CAT, which assists us to detect potential vulnerabilities, bugs, errors and redundancies by analyzing the source codes associated with these assets. After testing the entire simulation of these assets and manually approving the entire process, the asset application, containing these assets, their relevant source codes and configuration files (e.g., a properties file specifying the version of the CAT), is automatically delivered to the VCS repository of the Infrastructure Layer in the platform. After that, the application is used in a simulation pipeline, having particular functionalities (e.g., predictive maintenance), and finally brought into visualization to the users of the platform. It is also worthy to note that the manual approval step is successfully automated in terms of auto-check by Reiterer et.al. [4].

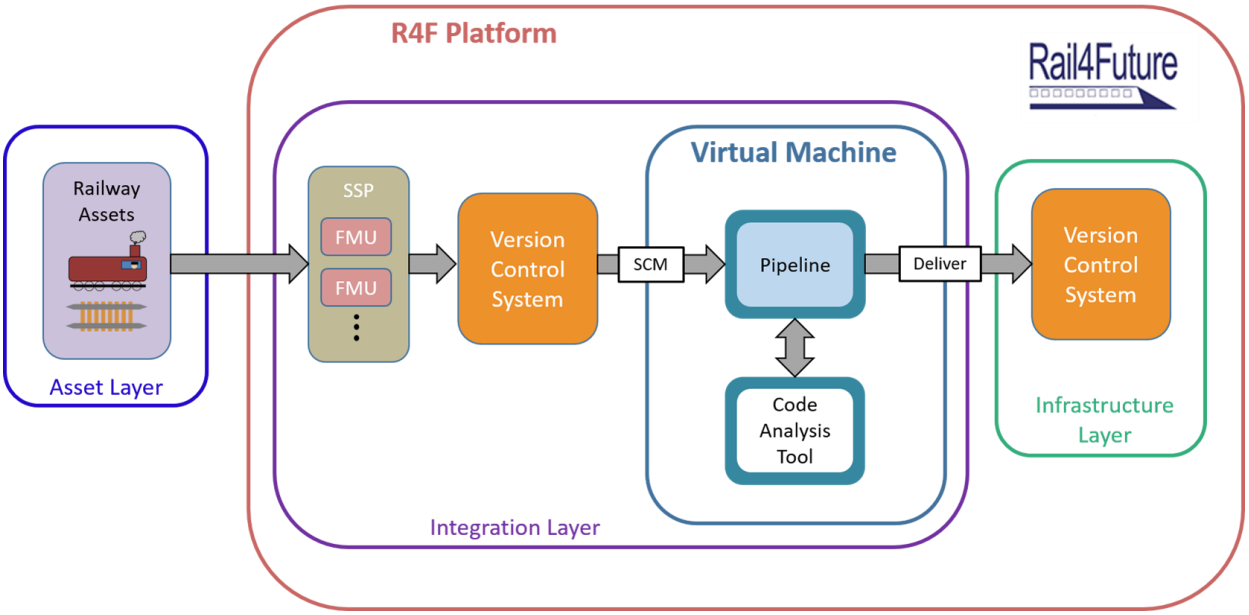


Fig. 2: Automated asset integration and processing in the R4F Platform. [3]

Fig. 3 shows how we demonstrate the entire automation and management of the asset integration and processing task in the R4F Platform. In this case, we build, test and then deliver the railway simulation assets in our Jenkins CI/CD pipeline. First, we as Asset Integrator start a shell script in the CLI of our VM. Thus, we can push the first version of these assets (pre-assets), pipeline code, describing the pipeline workflow, and configuration files (pre-configs), helping us to automatically configure these assets and the pipeline environment with necessary software package managers and FMI / SSP standards, into our VCS software repository. For the pushing process, we preferred to use the Git command (see https://git-scm.com/docs/git). Besides, we preferred to use an open-source mail transfer agent software tool called postfix (see https://www.postfix.org/) to notify all the stakeholders about progresses such as pipeline execution start during the asset integration and processing. Then, we remotely trigger the pipeline by executing the shell script, by which the entire asset simulation is officially built and tested in the pipeline. After automatic code analysis of the CAT, the FMI-based SSP asset simulation is started, during that input JSON is parsed and then output JSON and a LOG file, containing the pipeline console outputs, are generated. When the entire simulation is finished, the Asset Integrator receives an email from the pipeline, which uses the Mailer plugin (see https://plugins.jenkins.io/mailer/) installed in the Jenkins server, in order them to be notified about the end of the simulation for their manual approval to release the assets to the VCS repository of the Infrastructure Layer. This email includes valuable information such as the web links of the Jenkins pipeline, pipeline console output, Asset Integrator's CAT and VCS repository so that the Asset Integrator can easily check all the results only by clicking these links. After the Asset Integrator further optimizes these assets and their simulation occurring in the pipeline, they decide to deliver them as final assets, and also the generated LOG file. In the meanwhile, all the involving stakeholders are briefly informed about the succession of the asset delivery through another email notification coming from the pipeline. For the delivery, the Git command is used in the pipeline again like in the shell script execution.

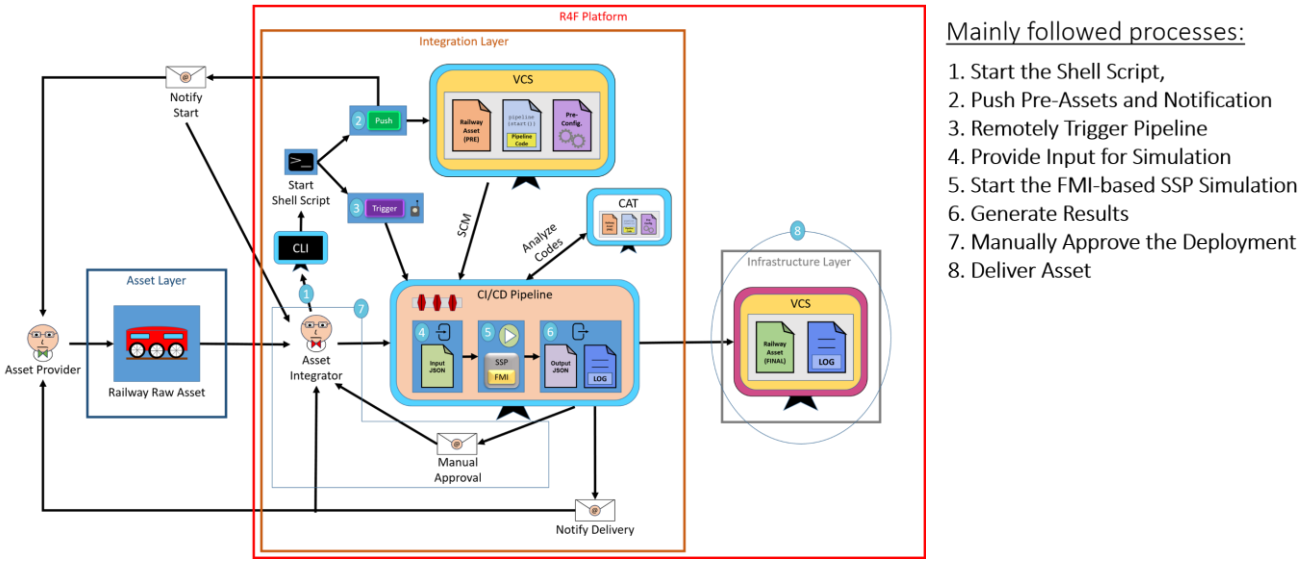


Fig. 3: Approach followed for the automated asset integration and delivery testing. [3]

In Table 1, we defined and described the main processes and subprocesses implemented in our pipeline code to apply them to our Jenkins CI/CD pipeline. First, in the Build step, most of the pre-configurations, needed for the asset simulation, are aimed to be automatically done by the pipeline. For example, we used the SCM functionality of the Jenkins server to connect the pipeline to our GitLab VCS repository. As second example, we preferred to use the pip freeze function in order the pipeline to install all the necessary Python packages and libraries, which are listed in a requirement.txt file. Besides, we succeeded to automate the FMU packaging process of all the railway simulation models incl. the PID-based MATLAB Simulink model, MBS Simpack model, RLT bridge Python model and ML-based surrogate Python model.

Main Processes	**Subprocesses**
Build	1) Checkout SCM: Connect the pipeline to the VCS repository.
	2) Pre-Install: Automatically install all the necessary dependencies.
	3) Build FMU: Auto-pack the simulation models into the FMU.
Test	4) Analyze Codes: Check all the source codes of the assets with the CAT.
	5) Validate FMU: Check FMU by using fmpy.
	6) Gather Info FMU: Parse metadata from the FMU(s) by using fmpy.
	7) Update SSP: Auto-upload the FMU(s) into the example SSP file.

	8) Simulate SSP: Start the FMI-based SSP simulation.
	9) Generate Results: Publish outputs incl. validation curves, CSV and JSON.
	10) Manual Approval: Asset Integrator approves the asset delivery.
Deliver	11) Deliver Asset: Assets are officially released to the Infrastructure Layer.

Table 1: Designed Workflow of the Jenkins CI/CD Pipeline. [3]

In the Test process, we aimed to automate and manage the entire code analysis with the CAT, FMU validation and information parsing processes, SSP updating, FMI-based SSP simulation, its result generation and the Asset Integrator's manual approval step. Therefore, we can be sure about the qualification of the asset integration and processing task while saving time thanks to the automation.

In the Deliver process, the final version of the assets is officially released to the Infrastructure Layer's VCS repository after the Asset Integrator manually approves the asset delivery in the pipeline. In the meanwhile, all the stakeholders are notified about the delivery as previously mentioned.

7 Conclusion

Based on the suggested R4F prototype and alternative approach above, this deliverable aimed to provide information about how to successfully automate and manage the integration and deployment of the assets belonging to the use cases successfully implemented into the R4F Platform. The automation would help the users of the platform to visualize and control the use cases for their predictive maintenance and condition monitoring tasks more time- and cost-efficiently.

In future, the automated model integration will be directly applied to the real R4F Platform together with the successfully implemented use cases. In addition, more use cases can be implemented into the platform, which ensures greater fidelity, resilience and availability for the Rail4Future project.

8 References

[1] Kugu, O., Zhou, S., Nowak, R., Müller, G., Reiterer, S. H., Meierhofer, A., ... & Grafinger, M. (2023, December). An FMI-and SSP-based Model Integration Methodology for a Digital Twin Platform of a Holistic Railway Infrastructure System. In Modelica Conferences (pp. 717-726). https://doi.org/10.3384/ecp204717

[2] Zhou, S., Dumss, S., Nowak, R., Riegler, R., Kugu, O., Krammer, M. and Grafinger, M., 2022. A Conceptual Model-based Digital Twin Platform for Holistic Large-scale Railway Infrastructure Systems. Procedia CIRP, 109, pp.362-367. https://doi.org/10.1016/j.procir.2022.05.263

[3] Kugu, O., Zhou, S., Reiterer, S.H., Schwaiger, M., Wurth, L. and Grafinger, M. (2024). Pipeline-based Automated Integration and Delivery Testing of Simulation Assets with FMI/SSP in a Railway Digital Twin. American Modelica Conference 2024. (paper accepted)

[4] Reiterer, S. H., Schiffer, C., & Schwaiger, M. (2023, December). A graph-based meta-data model for devops: Extensions to ssp and sysml2 and a review on the dcp standard. In Modelica Conferences (pp. 159-166). https://doi.org/10.3384/ecp204159

[5] Reiterer, S. H., Schiffer, C., & Benedikt, M. (2022). A Graph-Based Metadata Model for DevOps in Simulation-Driven Development and Generation of DCP Configurations. *Electronics*, *11*(20), 3325. https://doi.org/10.3390/electronics11203325