FMU Best Practice Guide

Best Practices for Applying the FMI Standard in Virtual Commissioning

Created on: November 19, 2025

Version: 1.0

FESTO

Table of Contents

Foreword

Virtual commissioning is a key process for validating control software for machines and systems. Typically, modeling the individual components requires significant effort. Against this backdrop, the present guideline was developed as part of the funded DIAMOND project. Its objective is to accelerate the creation of digital twins and enhance their quality through a consistent and reusable modeling approach. This guide offers practical recommendations to support successful implementation and promote industry-wide adoption. Central to this approach is the Functional Mock-up Interface (FMI) standard, which facilitates the creation and integration of standardized behavioral models known as Functional Mock-up Units (FMUs).

This guideline results from the research project "Digital Plant Modelling with Neutral Data Formats" (DIAMOND). The consortium comprises over 20 partners, including component manufacturers, plant builders, plant operators (OEMs), software providers, and research institutions. DIAMOND is funded by the European Union (EU) and the German Federal Ministry for Economic Affairs and Energy (BMWE). We extend our sincere thanks to the project sponsors whose support made this guideline possible.

The DIAMOND Simulation Taskforce, dedicated to standardized behavior modeling based on the FMI standard, played a pivotal role. Experts from the consortium contributed their knowledge and experience through a series of expert interviews that formed the foundation of this guideline. We are grateful to all interviewees for their valuable insights, which helped illuminate the topic from diverse perspectives. This collaborative effort was essential to develop a comprehensive understanding of various processes, thereby enhancing the guide's acceptance across the industry.

We also acknowledge the Modelica Association, which continues to advance the FMI standard through the "Modelica Association Project Functional Mock-up Interface," ensuring its high level of maturity. We appreciate the professional exchange fostered within the framework of the DIAMOND project.

Finally, the individuals involved in creating this guide are listed at the end of the document under Editors and Contributors. We would like to emphasize the excellent cross-company collaboration that made this work possible. Our sincere thanks go to everyone for their discussions, suggestions, and shared experiences, all of which have been thoughtfully incorporated into this guideline.

1 Executive Summary

This guideline, developed as part of the DIAMOND funding project, focuses on applying the FMI standard to create and integrate standardized behavioral models through FMUs for the virtual commissioning of production plants. The goal is to enhance reusability, quality, and efficiency in behavioral modeling for digital twins. This is achieved through collaboration among component manufacturers, plant manufacturers, plant operators (OEMs), and tool manufacturers.

Problem

Virtual commissioning has become essential for securing software development and commissioning machines and systems in industrial automation. A significant challenge arises in creating associated behavioral models for simulation, especially with increasingly complex automation components. Currently, tool manufacturers, plant manufacturers, plant operators (OEMs), and sometimes component manufacturers develop their own models, leading to high effort, redundancies, and limited reusability. The diversity of engineering tools necessitates a uniform standard for seamless, cross-company digitization.

Solution and associated challenges

The FMI standard provides a tool-independent interface for modeling and simulating behavioral models as FMUs. However, integrating FMUs across companies and industries involves challenges, questions, and concerns. This guide offers practical insights on using FMUs in virtual commissioning from modeling to validation, delivery, and integration into existing tool landscapes. It provides concrete recommendations on FMU structure and content, the use of flags, documentation, and integration into common VC processes. The aim is to accelerate the nationwide adoption and use of the FMI standard.

Added value for the industry

The application of the FMI standard allows for:

- ✓ Reduced modeling effort through reusability and standardization.
- ✓ Greater flexibility in tool selection and switching through standardized model integration.
- ✓ Higher model quality, as component manufacturers can directly contribute their knowledge by providing FMUs.
- ✓ New application possibilities through realistic modeling and varying levels of detail due to component manufacturer integration.
- ✓ Reduced effort in building and maintaining libraries through unified deployment and versioning.

- ✓ Long-term acceptance through potential connections to standards such as AAS, AML, and SSP, as well as applications in cloud environments and the use of artificial intelligence.

Conclusion & Importance of the Guide

The FMI standard is a crucial building block for digitization in mechanical and plant engineering. The FMU guideline offers a practice-oriented foundation and addresses key areas in the digital engineering process. In the long term, FMUs are expected to become integral to component delivery, similar to CAD data. Only through standardized behavioral models can the increasing complexity in automation be managed, securing the industry's competitiveness.

2 Definition of the term and state of the art

2.1 Original Equipment Manufacturer (OEM)

In the context of the automotive industry, the term OEM refers to vehicle manufacturers. These manufacturers integrate products from module, component, and parts suppliers, thus earning the designation of original equipment manufacturers. The machines and systems provided by the plant manufacturer are used in vehicle production and operated by the OEM. Therefore, in this context, the terms OEM and plant operator are used interchangeably.

2.2 Behavior Model of an Automation Component

An automation component is a device that interacts with a control system. A basic distinction is made between actuators (e.g., drives, valves, heating rods, signal lights) and sensors (e.g., buttons, switches, RFID readers, light barriers). A behavioral model describes the virtual behavior of these devices or components within a plant model [16]. The virtual component behaves analogously to the real component [20-24]. In virtual commissioning (VC), models are mostly located at an abstract communication level: the input/output (I/O) behavior of the controllers is mapped, but physical effects are rarely modeled [16]. Detailed modeling is deliberately avoided due to the high performance demands it places on software tools and hardware for complex components. Additionally, third-party providers often lack detailed knowledge of the component when creating behavioral models.

2.3 Virtual Commissioning (VC)

A fundamental aspect of the engineering process is VC. In VC, desired functions and processes are tested step-by-step in a virtual system model. This approach aims to enhance the real programmable logic controller (PLC) and robot programs, as well as communication with higher-level IT systems, during processes, interactions, and error scenarios before the real production plant exists [16]. This enables parallel development across various

trades and early validation of the control code. To achieve this, behavioral models of individual system components are required, which are coupled with virtual or real PLC or robot controllers (RC) (cf. Figure 1). Interfaces to the system operator, such as the Human-Machine Interface (HMI) or the Teach Pendant, are connected to the real or virtual control system. Behavioral models interact with visualization systems to enable sensor feedback and material flow. Examples of higher-level IT systems include control systems, production data acquisition (PDA), machine data acquisition (MDE), and IoT servers. For further discussion on VC, refer to the literature [31], [32], and [33].

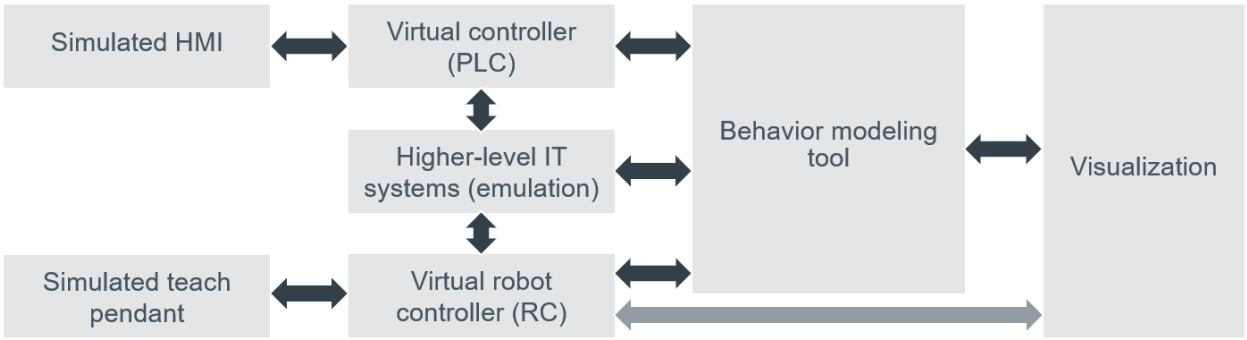

Figure 1: Classic VC Setup: This image illustrates the general architecture of virtual commissioning. Note that the virtual robot controller can also be connected via the behavior modeling tool

2.4 Data basis

The simulation model of the VC comprises models of individual subcomponents, such as PLCs, HMIs, peripherals, and robots. Most manufacturers provide emulators for PLCs, HMIs, and robots, allowing the use of original code. The 3D visualization utilizes data from mechanical design, typically supplied by manufacturers. However, there are no standardized data formats for the necessary behavioral models of real components like drives and sensors. These behavioral models must often be painstakingly recreated independently or by third parties, leading to inevitable behavioral deviations. Consequently, certain tests can only be conducted on the actual plant. Tool manufacturers frequently develop proprietary libraries, resulting in numerous variants of the same component [16].

2.5 Digital twin

There are various definitions of the term "digital twin," which can vary significantly depending on the application area and organization. In this guideline, a "digital twin" refers to the representation of the real production plant that will eventually exist. The scope and detail of the digital twin depend on its intended use in the VC, with focus

given to the automotive industry. Key components of the digital twin include behavioral models, a cinematized CAD model, material flow, virtual controls, and a user interface for performing the VC.

2.6 VC Tool

In this guide, "VC tool" refers to commercial software solutions used to model the digital twin for virtual commissioning. This primarily includes tools for behavioral modeling of automation components, such as SIMIT (Siemens AG), RF::ViPer (EKS InTec GmbH), and WinMOD (WINMOD GmbH). Software solutions for visualization via 3D geometries are usually mentioned separately under "Visualization," although they are also used to model the digital twin.

2.7 Basics of the FMI standard

The Functional Mock-up Interface (FMI) standard [6] was initiated in 2008 during the ITEA 2 MODELISAR project, which aimed to develop a tool-independent, open interface standard [11,14]. The goal was to connect various simulation tools [11]. The FMI standard originated in the context of the interaction of different control units (ECUs) in vehicle technology [15]. Currently, nearly 250 tools (modeling, simulation, testing tools, etc.) support the standard in various versions. The latest update of the standard was in November 2024 with version 3.0.2 [11].
Analogies to plant engineering, such as the cyclical processing of step sequences, encapsulation of individual components, and consolidation via different software tools, led to investigations in the VC context during the ENTOC project in 2018. Currently, common VC tools, such as SIMIT, RF::ViPer, and WinMOD, support the simulation and integration of models created in the FMI standard (Functional Mock-up Units (FMUs)).
In the FMI standard, models can be encapsulated via co-simulation or ModelExchange (cf. Chapter 5.2) and connected to any simulation tool. Various files are provided in a zipped container. The model behavior is described via a binary compiled in C code, enabling know-how protection, and creating acceptance by the FMU creator. An open XML file provides general information and interface signals with metadata, such as names, data types, and default values. The co-simulator reads this XML for integration. In the simulation, inputs are read, a calculation step is performed, and outputs are written. This procedure repeats cyclically with a given time (cf. Figure 2).

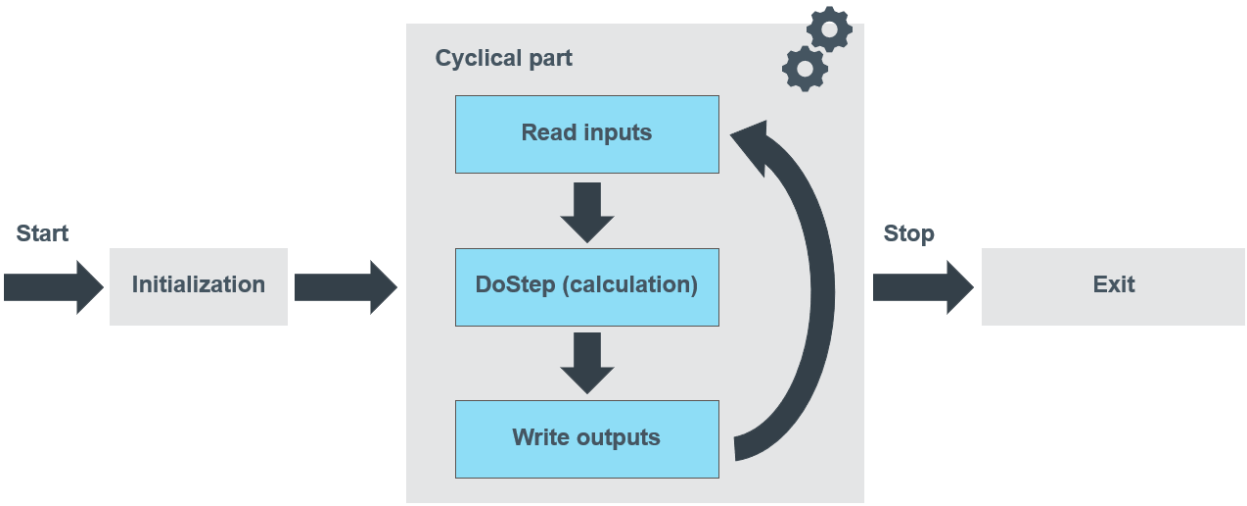

Figure 2: Processing of an FMU

Additional conventions and functionalities can be described via so-called "layered standards" so that the standard can be adapted to the respective application.

3 Introduction and motivation

The FMI standard offers significant potential for improvement in virtual commissioning. Currently, a large portion of the effort in model creation is dedicated to developing behavioral models. Extensive, tool-specific libraries are required to simulate different systems, and these must be created specifically for each simulation tool. For component manufacturers, it is impractical to provide behavioral models for every simulation environment due to the vast number of tools available. Consequently, Original Equipment Manufacturers (OEMs) currently maintain various tool-specific libraries containing behavioral models for the same component. These models must be created and maintained separately, which demands considerable effort (cf. Figure 4). Depending on the tool used, the behavior of the same component may vary for each project.

Figure 3: Logo of FMI-Standard [11]

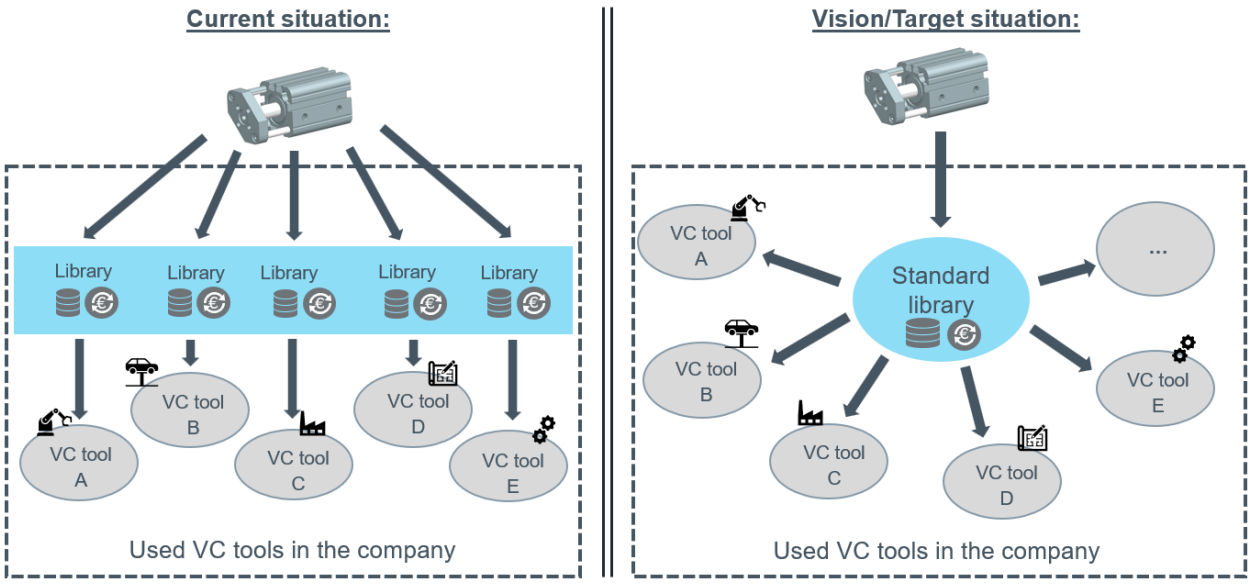

Figure 4: Comparison of the current situation at the OEM and the target situation (vision)

With the help of FMUs, component manufacturers can provide standardized simulation models alongside their real components, which can be integrated by a wide variety of tools. Currently, behavioral models are often based on data sheets, manuals, or PLC function blocks, allowing only an approximation of the real component's behavior. Since component manufacturers have the best knowledge of their products, model quality is improved, and defects are reduced (cf. Figure 5). This means that the behavioral models can be used not only for virtual commissioning but also for the development of PLC modules, which currently rely mostly on real components.

The FMI standard allows the component manufacturer to provide tool-independent models (FMUs).

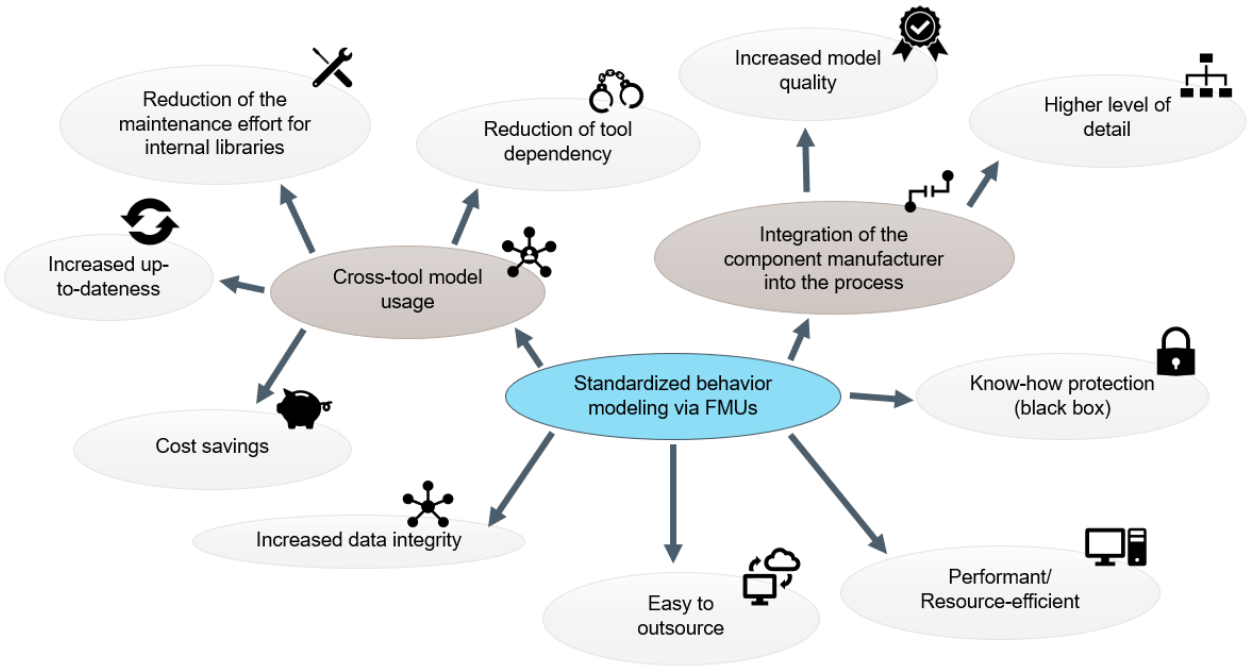

Figure 5: Advantages of using FMUs in the context of the VC

In addition to behavioral models for individual components, it is often necessary to model subsystems. Examples include the sub technology of robots and measuring systems, some of which have their own control systems and communicate with the rest of the system via an interface. Documentation usually focuses on control, while the internal structure and relevant logic are not described, leading to superficial third-party modeling. In this case, the FMI standard provides system suppliers with the opportunity to offer a behavioral model alongside the real module, significantly improving the quality of virtual commissioning (VC).

To enable realistic VC, processes are also simulated in certain areas, which can be modeled as standardized FMUs. Generally, a distinction must be made between device/control technology and process FMUs. In this context, the FMI standard allows for the creation of tool-independent models.

Since the FMI standard is used in various applications and industries, functionalities are diverse and operating modes differ. Starting with the creation of suitable FMUs is often time-consuming and requires familiarization with the standard. Additionally, selecting an appropriate tool for FMU creation depends on individual preferences, necessitating studies to gain experience. The creation of standardized behavioral models involves additional effort, which is why component manufacturers currently offer FMUs only in individual cases. Integration into VC tools has been hesitant in recent years, as the need for an FMU co-simulator was typically absent in productive operations. FMU connections often occur in research projects or demonstrators.

For the use of FMUs as behavioral models, the following questions arise for the component manufacturer [16]:

- What types of FMUs to simulate?
- Which functions/scope of the standard is to be implemented in the context of the VC?
- Which tool is suitable for FMU creation in the respective use case?
- What level of detail needs to be mapped for the model?

The widespread use of FMUs in the VC is only possible through the cooperation of different roles. For example, the creation of FMUs is the responsibility of the Component Manufacturers. The behavioral models are used at the Plant Manufacturer which then merges the models into a digital twin of the system (cf. Figure 6). The VC is then carried out by the plant manufacturer in the presence of the Plant Operator (OEM). To enable the connection of the FMUs in the digital twin, integration into the simulation environment is required by the Tool Maker. This guideline addresses each of the four roles mentioned.

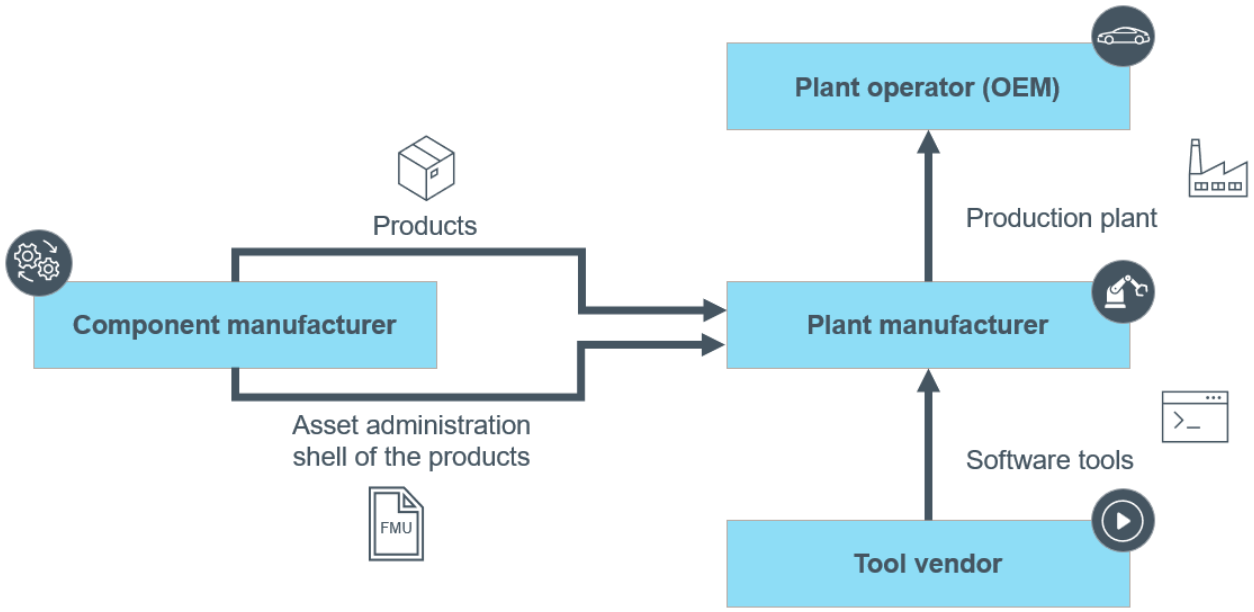

Figure 6: Interaction of the four defined roles

A comprehensive integration of the FMI standard is only possible if the component manufacturers, plant manufacturers, plant operators (OEMs) and tool manufacturers participate.

Currently, the standard is scarcely represented in daily, productive VC business and primarily garners attention through various research activities and pilot projects. Consequently, market penetration is low [16]. The integration of the FMI standard must occur across companies and be accepted by all stakeholders. The Festo and the DIAMOND project offers this FMU guideline to expedite the integration of FMUs into the VC process. The

DIAMOND consortium is ideally positioned to facilitate the introduction of the FMI standard within the automotive industry's VC context. The project partners bring diverse perspectives to the topic.
It is crucial for the document to gain industry-wide acceptance. Each role must relate to the guide's content. Therefore, different roles were interviewed separately via expert interviews as part of the DIAMOND research project. A total of nine partners from the DIAMOND consortium, representing all four roles, participated. The interviewees had prior experience with the FMI standard, enabling them to provide valuable insights and feedback.
This approach helped identify and gather previous findings, challenges, and concerns. This information was used to derive functional and non-functional requirements for the standard. The guide addresses the model creation and simulation of FMUs, offering solutions for implementation and integration. Additionally, it outlines workflows for integrating the standard into the VC process. Given the focus on an existing standard, it is vital to exclude certain functions and areas not required for virtual commissioning. Other significant topics and potential obstacles during implementation should be highlighted.

4 Reading Guide

The purpose of this chapter is to provide the reader with a quick insight into the scope of this guide. Depending on the reader's perspective and objectives, references to specific chapters are provided.

4.1.1 A Guide to Different Roles and Perspectives

As discussed in Chapter 3, four roles can be distinguished (cf. Figure 7) and are addressed in this guide. Depending on the specific role, certain chapters are relevant:

- **Component Manufacturer:**
  Chapter *5;* Chapter *6;* Chapter *7;* Chapter *10;* Chapter *11*

- **Plant engineers:**
  Chapter *6;* Chapter *8;* Chapter *9;* Chapter *10;* Chapter *11*

- **Plant operator (OEM):**
  Chapter *6;* Chapter *8;* Chapter *9;* Chapter *10;* Chapter *11*

- **Tool Manufacturer:**
  Chapter *5;* Chapter *8;* Chapter *9;* Chapter *10*

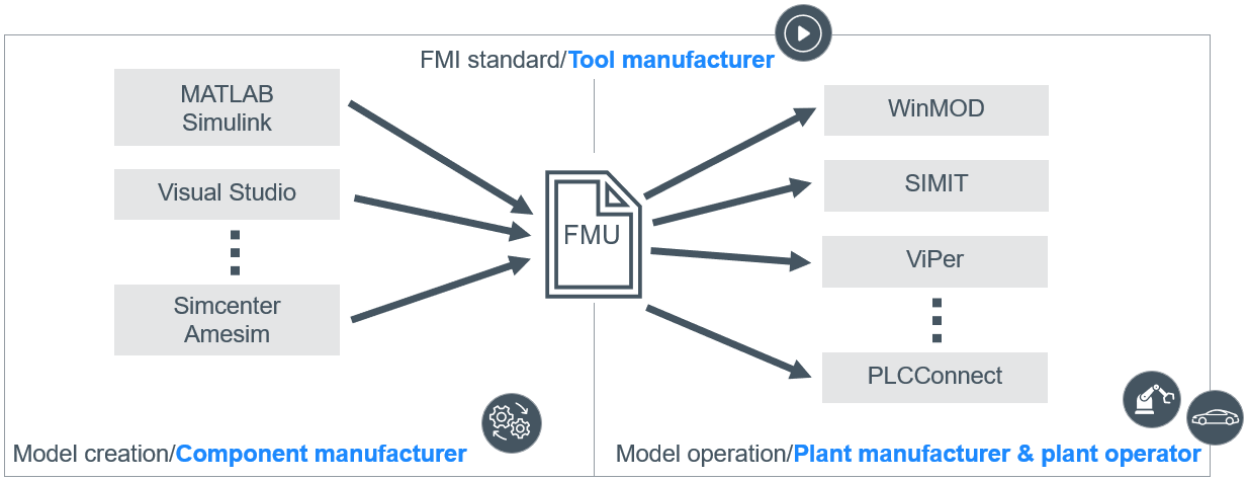

Figure 7: Classification of the four different roles in the overall context

4.1.2 A Guide to Discussing Different Requirements

In the context of the FMI standard within the VC, challenges and requirements often arise, making it difficult to universally implement and utilize FMUs. The paper "FMI in the VC: A Guide to Implementation" [16], published as part of DIAMOND, addresses these requirements, which are detailed below and linked to the relevant sections in the guideline. This enables readers to access information precisely and efficiently:

- **Real-time Capability, Cycle Time, and Performance:** Section 9.11; Section 7.4 FMU simulation is independent of real-time, meaning simulated time and simulation time are not necessarily correlated. In the VC context, very small cycle times are essential [16].
- **Visualization and User Integration:** Section 5.5; Section 7.3; Section 9.6 For the VC, FMUs operate in real-time. The simulation must be influenced by this, necessitating interfaces and user interaction. The FMI standard does not include visualization in the form of a control panel or user interface [16].
- **FMI-VC Libraries:** Chapter 6; Section 7.3; Chapter 8; Section 9.1 Generalizable cross-tool libraries present certain challenges: Component manufacturers must select an appropriate level of model detail, as the behavior is encapsulated as a black box. VC tools must implement the FMU connection accordingly [16].

- **Compatibility & Dependencies:** Section 5.7; Section 6.4; Section 7.6
  FMUs sometimes require external dependencies or licensing mechanisms to execute. FMUs from different manufacturers may be combined [16].

- **Security**: Section 9.9
  Since the FMU contains an encapsulated, compiled binary file, there are certain security risks associated with running the FMU, especially when FMUs are provided via the Internet [16].

The software solutions and tools mentioned in the guide are merely examples. Most of the products listed were from consortium partners, but there are other analogous solutions available on the market. The same applies to the fieldbus systems mentioned; PROFINET and EtherCAT are cited only as examples.

This document serves as a reference framework rather than a prescriptive set of rules. It provides recommendations intended to facilitate the introduction to the FMI standard and accelerate the familiarization process. The primary emphasis is on its application within Virtual Commissioning (VC) in the automotive industry. While standardized behavioral models can be adopted in other sectors, the required level of detail may vary depending on specific industry needs.
The guide concentrates on behavioral models of automation components; however, the underlying principles are equally applicable to process, product, building, and module FMUs. It reflects the state of practice at the time of publication. Future developments, evolving conditions, and version updates of the standard must be carefully considered.

5 FMI standard in the VC

The FMI standard is openly available on the homepage of the Modelica Association. The current version 3.0.2 can be viewed via the following link: https://fmi-standard.org/docs/3.0.2/ [11].
In this guide only versions 2.0 and 3.0 are considered. In general, an FMU describes a model created in the FMI standard. This is in the form of a zipped container (.ZIP File Format Specification in version: 6.3.10 [7]) provided to the user. The structure of the container differs for the different FMI versions (cf. Chapter 5.1) and is shown in Figure 8. One of the most important components of FMU is the modelDescription.xml. This is a standardized model and interface description. The XML serves as a link between the user and the stored model. The interface signals (inputs, outputs, parameters) are listed here and enriched with attributes such as name, description, default value or the unit. General information about the model and the simulation is given. Depending on the mode of operation of the FMU, the model behavior is stored as disclosed source code in the "resources" folder or as an encapsulated compiled file under "binaries" [11]. The remaining files and information are to be considered optional and are discussed in this chapter 5 in some cases. Important components and contents in the modelDescription.xml are also discussed in more detail.

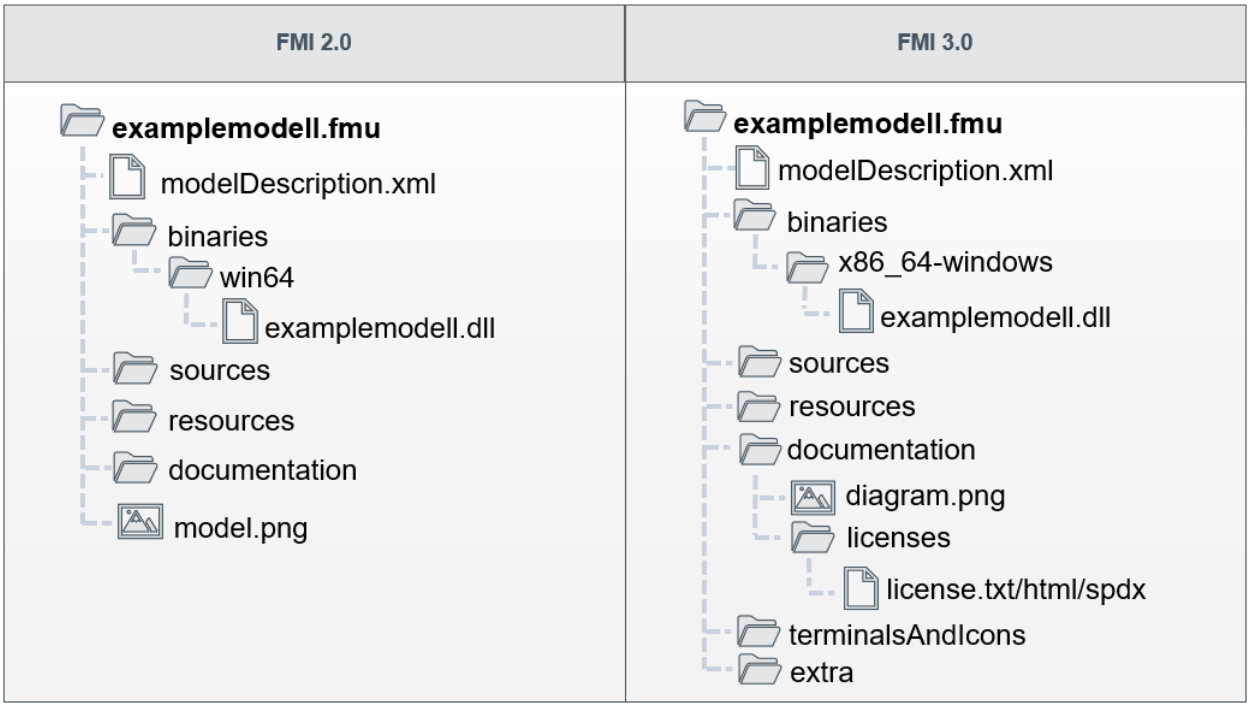

Figure 8: Structure of an FMU container in FMI 2.0 (left) and FMI 3.0 Standard (right)

5.1 Choosing the FMI version

Since it is a standard that is constantly being developed, it is difficult to recommend a specific version. It is advisable to use new versions if possible. The application in the context of the VC has shown that up to version 2.0, a sufficient scope for modeling and connection to the VC tool is included in the standard. From version 2.0 onwards, FMUs can thus be used productively for VC projects. The integration into the VC tool landscape has already taken place. As well as other functions like complex datatypes or arrays (cf. Chapter 9.4) are supported with FMI standard 3.0, which makes this version preferable for the creation of behavioral models in the context of the VC. It is precisely these additional data types that allow the interface to be mapped analogously to the real component. For example, arrays can be used to map the PROFINET interface in its entirety as a single signal. A detour via split individual signals, as in the 2.0 standard, is not necessary. The 3.0 standard is already supported by many creation tools as well as co-simulators.

It is important that backward compatibility is also maintained during a version upgrade. This applies to the co-simulator, which must continue to support old FMU versions (from 2.0) in addition to new ones. Hybrid operations with FMUs of different FMI versions (2.0 and 3.0) must also be enabled in the co-simulator. In principle, the

component manufacturer can also provide FMUs in various standard versions if there are advantages for customers in this context.

The use of the FMI standard in version 3.0 is preferable in the context of the VC.

5.2 Selection of FMI mode

The FMI standard supports different modes of how a simulation model can be exchanged in a standardized way. For example, there is the so-called "ModelExchange", the "Co-Simulation", and in version 3.0 the "Scheduled Execution" mode.

For many component manufacturers, it is important to protect their know-how. Here, the FMI standard offers the possibility to exchange model behavior via compiled binaries. This is particularly crucial as soon as software or source code from the real component is integrated for modeling. Simple logic models are less critical here. The encapsulated, compiled files pass the model in the form of a black box. On Windows, this is done via a .DLL file, and on Linux via the .so file. This is not optimal for the user, as he has no insight into the internal behavior and adjustments to the model are not possible. With sufficiently validated models, these requirements are eliminated, so that the acceptance of these black box models increases.

Optionally, the behavior can be stored in the FMU via a standardized C-code and made available to the target system. The content is thus disclosed across the tool boundaries. A confidentiality agreement (NDA) for the protection of know-how between the creator and the user is urgently needed if source code worthy of protection is also stored in the FMU.

5.2.1 ModelExchange

With the ModelExchange, the model is computed by the solver of the target tool (cf. Figure 9). This achieves good model performance, parallel computation, and high accuracy. Because different target tools use different solvers, model behavior may differ in each use case.

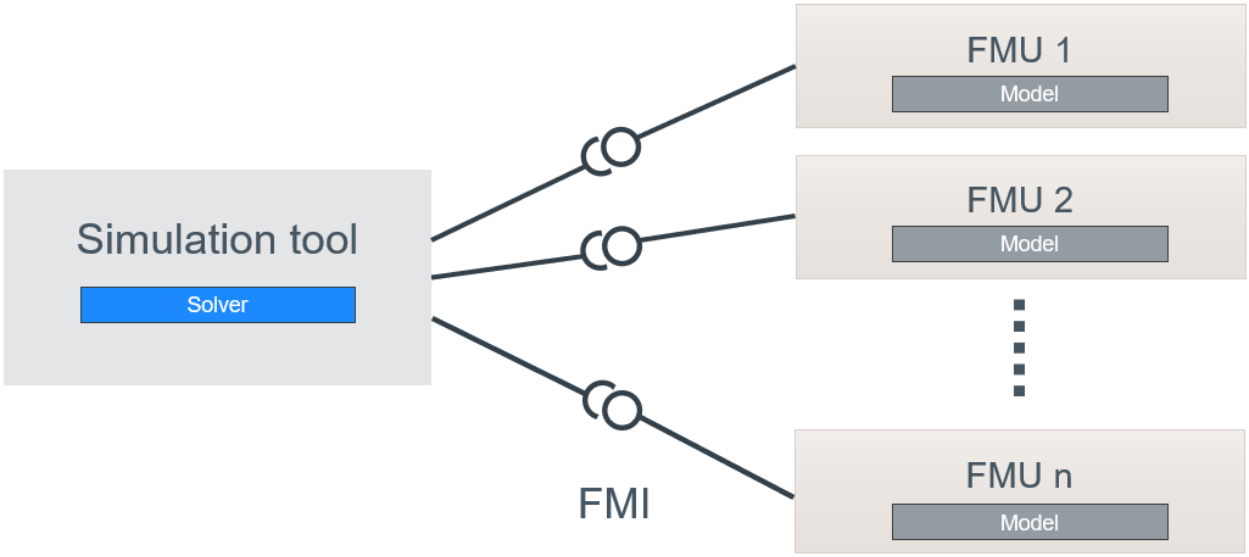

Figure 9: Outlined functionality of an FMU in ModelExchange mode according to [34]

A provision of FMUs in ModelExchange mode is conceivable for the component manufacturer, especially for complex models in exceptional situations. Especially in exchange with other models or FMUs, direct integration into the target system offers better numerical stability because the same solver is used. For this purpose, information such as the continuous states and their derivatives (often also as output signals) is disclosed. The FMUs can then be combined and simulated via a solver as a co-simulation externally or in the target tool. In the context of the VC, the ModelExchange is not the preferred and common way. Thus, special assignments are necessary in such special cases.

5.2.2 Co-Simulation

In contrast to the ModelExchange is the Co-Simulation mode, where a compiled file is passed together with the corresponding solver. Co-simulation is the most robust type of application and delivers the same, reproducible behavior even with a wide variety of tool setups, as the solver is integrated directly into the FMU. In the case of co-simulation, the individual FMUs are simulated via a co-simulation master (without solvers). The FMU contains both the model and the corresponding solver (cf. Figure 10). For this reason, this mode is particularly suitable for use in the VC.

However, it should be noted that due to the encapsulated and independent calculation of the individual sub-models, problems can arise with strongly coupled components (e.g. controllers, physics-based feedback). The FMUs are coupled exclusively via the inputs and outputs, which can lead to numerical errors. In this context, the processing order of the individual models (FMUs) is also important.

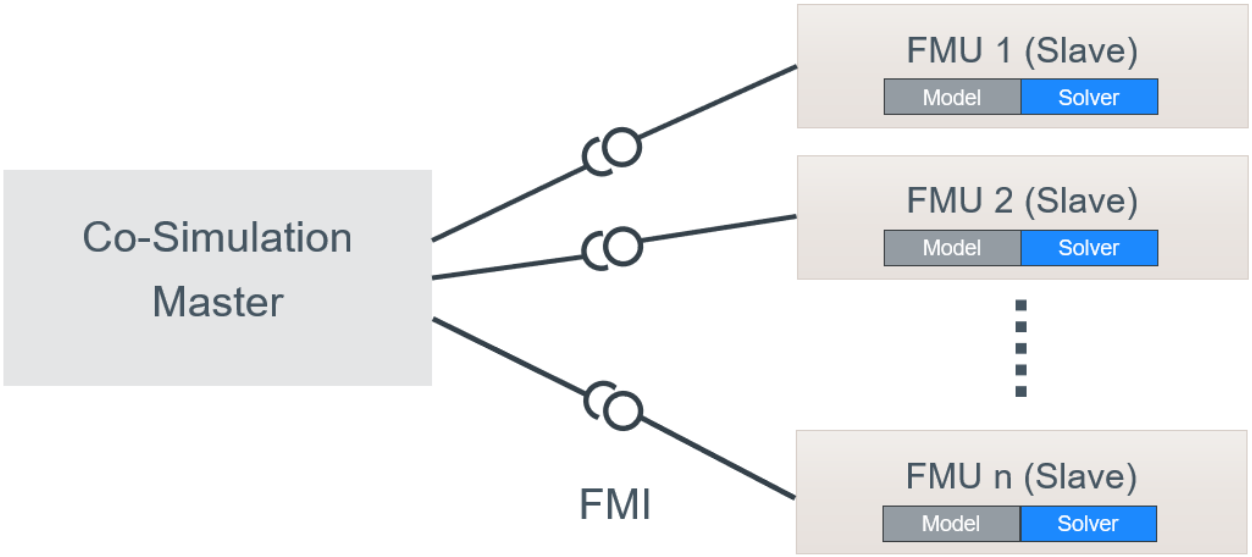

Figure 10: Outlined functionality of an FMU in co-simulation mode based on [11]

5.2.3 Scheduled Execution

In addition to the two modes mentioned, version 3.0 of the standard supports the so-called "Scheduled Execution". Here, different model partitions can be clocked in a targeted manner and executed one after the other. For complex behavioral models, this mode could become relevant in the future.

FMUs can be created and connected via co-simulation.

5.3 Choice of target system

Since most of the VC tools run in **Windows** environment, the variant with the **64-bit DLL** should be chosen. The 32-bit version is only used sporadically, so that a clear recommendation for the 64-bit DLL is made. The standard also allows Linux files to be stored. This will become increasingly important in the future, especially in the context of edge computing, cloud or high-performance computing, and will bring with it a more stable operation of FMUs. For flexible use, FMU can contain both Windows and Linux binaries in the medium to long term.

FMUs can currently be developed and connected as a 64-bit variant via Windows.

5.4 Entries in the modelDescription.xml

The modelDescription.xml defines the interface between the black box (DLL) and the co-simulator. Information and settings are stored when FMU is created. The user or the co-simulator consumes this content. There is no provision for subsequent changes or adjustments to the modelDescription.xml (e.g. by the FMU user).

5.4.1 DefaultExperiment

In the modelDescription.xml there is a section called "DefaultExperiment". Below is information about the simulation. Examples are the total simulation duration for the model, which is defined by the start and stop time, or a possible (meaningful) cycle time (cf. Figure 11). The co-simulator can use and consider the information provided here for the simulation. The information under "DefaultExperiment" can be stored optionally. A well-developed FMU lists the settings to use here. If no information has been provided in this area, the co-simulator must use appropriate default values. The stepSize often uses 0.001s (1ms). The stopTime is set to the maximum possible time ("Inf") so that the FMU runs for as long as possible. Especially for detailed FMUs, it is recommended to specify an ideal stepSize. Especially in differential equations or single-adjustment processes, long cycle times can have a negative impact on the results.

```
<DefaultExperiment startTime="0" stopTime="3" stepSize="1e-2"/>
```

Figure 11: Example excerpt from a modelDescription.xml in the Default Experiment section

5.4.2 Use of different flags

The FMI standard allows certain flags to be set in the modelDescription.xml (cf. Figure 12). These are already defined when FMU is created and are to be regarded as constant. These settings can be used to communicate information about restrictions and settings for the simulation to the co-simulator. The co-simulator has these flags too and to give a notification as soon as implemented functions are not supported. The co-simulator cannot change the flags. As long as a flag is not listed in the modelDescription.xml, it is considered false by default [11]. However, it is a good practice to also list flags with the setting "false" in the XML.

```
canHandleVariableCommunicationStepSize="true"
canBeInstantiatedOnlyOncePerProcess="false"
canGetAndSetFMUState="true"
canSerializeFMUState="true"
```

Figure 12: Example excerpt from a modelDescription.xml in the definition area of the various FMI flags

canHandleVariableCommunicationStepSize:

If this flag is not set ("false"), the FMU always expects a certain constant cycle time. Fluctuating cycle times of different sizes lead to an error. The FMU cannot deal with this. Due to different workloads in the Windows environment, it makes sense to allow a variable cycle time. In the VC context, a fixed step size can also be used under certain circumstances.

canBeInstantiatedOnlyOncePerProcess:

This flag tells the co-simulator that it may only instantiate the FMU once per process. The automation component can therefore only be simulated once per process (tool). The background for this flag are different ways of dealing with internal variables and states (e.g. global variables, singletons). If the same memory areas are used by different instances, this flag must be set, as correct model behavior is not possible with multiple instantiations. To simulate multiple instances of FMU, the models created must be do not use shared resources. In the context of the VC many co-simulators on one process and distribute the FMUs into individual threads. For this reason, this flag often causes problems. If several identical components are installed in a system, complications arise in the digital twin. In the context of the VC, multiple instantiations are therefore essential (cf. Chapter 9.8). This flag (*canBeInstantiatedOnlyOncePerProcess = true*) is therefore to be considered when creating FMU. The ability to multiple instantiations is often dependent on the export tool, If possible, a creation tool should be used that does not set this flag (*canBeInstantiatedOnlyOncePerProcess = false*).

canGetAndSetFMUState:

If it is necessary for an FMU to read the state at a certain point in time, this flag is relevant and must be set to true. The state can also be set and released by the co-simulator. However, the requested state is lost again when the simulation stops. Even if FMU supports this function, it is not mandatory to call the functions in the co-simulator.

canSerializeFMUState:

Once this flag is set to true, *canGetAndSetFMUState=true* must also be defined. The canSerializeFMUState flag enables the co-simulator to obtain states, inputs, and model parameters from the FMU and store them internally. In this way, the values can be set again at the next simulation start, so that the simulation is resumed at a certain point. Even if FMU supports this function, it is not mandatory to call the functions in the co-simulator. This is an important functionality, especially in the field of acyclic bus communication. For this functionality to be used, it must be implemented accordingly in the co-simulator.

Many VC behavior modeling tools offer the option of records or snapshots (part of a system at a specific point in time). This allows a certain signal image to be recorded and played back again. In this way, error states and processes in the VC can be reproduced in a targeted manner. The *canSerializeFMUState flag* tells you whether this functionality is also supported on the FMU side. In this context, it should generally be noted that FMU can also

deal with large jumps in the value of inputs. Crashes or erroneous calculations due to discontinuities and large changes in value within a cycle must be prevented. FMU must also be validated regarding such extreme cases.

The FMU flags must be considered on the creator's side when selecting the tool. The guide makes the following recommendation:
canHandleVariableCommunicationStepSize = true
canBeInstantiatedOnlyOncePerProcess = false
canGetAndSetFMUState = true (relevant in future)
canSerializeFMUState = true (relevant in future)

5.4.3 Entries under Model Info

General information (metadata) must be stored in this area of the modelDescription.xml (cf. Figure 13). This includes, for example, the author, the model's name, the generation tool and the date and time when the model was built. In addition, a brief description of the function and scope of FMU can also be given. Not all this information is urgently needed by the co-simulator for the simulation and is therefore considered optional for FMU creation. The information under Model Info can be read out and displayed to the user via the VC tool (co-simulator) (e.g. FMU block diagram with metadata in the VC tool). With a high level of integration of FMUs, this visualization makes sense. This metadata is also of interest for the storage and integration (e.g. in the asset administration shell, System Structure and Parameterization (SSP), ...) of FMUs. Under these attributes there are indications, which must be deposited urgently in the FMU. These include the modelName and the fmiVersion and a unique ID.

Following Table 1 gives an overview of the various attributes and divides them into optional and mandatory information:

Mandatory attributes (fmiVersion 2.0 & 3.0)	
fmiVersion	FMU created in the 2.0 or 3.0 standard
modelName	FMU-Name analog zum Container-Name
GUID (fmiVersion 2.0) / instantiationToken (fmiVersion 3.0)	Unique ID of FMU
Optionale Attribute (fmiVersion 2.0 & 3.0)	
description	Brief explanation of the model
author	Author's name
version	Model version
copyright	Copyright des Modells
license	Information about licenses
generationTool	Authoring tool name
generationDateAndTime	Creation date and time

Table 1: List of various attributes from the modelDescription.xml with a classification into mandatory and optional information

```
fmiVersion="3.0"
modelName="BouncingBall"
description="This model calculates the trajectory, over time, of a ball..."
generationTool="Reference FMUs (development build)"
instantiationToken="{8c4e810f-3df3-4a00-8276-176fa3c9f003}">
```

Figure 13: Example excerpt from a modelDescription.xml in the definition area of the various model attributes (Model Info).

These attributes generally provide information about when the FMU was created and where it came from. In case of problems or for better traceability, this information can be extremely important. Important management information such as the GUID (fmiVersion 2.0) or the instantiationToken (fmiVersion 3.0) as well as the version and the modelName must be maintained to enable correct instantiation in the co-simulator. This information is of

enormous importance, especially for a model update process in existing projects. In the case of incomplete or poorly maintained attributes, this process is made more difficult or even impossible, which entails a manual update process or the relinking of signals. The indication of the license is also to be regarded as mandatory as soon as a licensing mechanism is used. Specifying the generation tool helps to limit problems during co-simulation at an early stage and makes the handling of FMUs more transparent.

GUID/instantiationToken:

The GUID (fmiVersion 2.0) or the instantiationToken (fmiVersion 3.0) represents a unique ID that is generated during model creation. Different IDs have to be stored for different models. The GUID or instantiationToken is used to check whether the modelDescription.xml and the compiled DLL match. For this purpose, the method for instantiating the FMU is given the ID from the modelDescription.xml. If both IDs do not match, instantiation in the co-simulator fails [11]. In the event of changes in the modelDescription.xml, according to the standard, an adjustment of the ID is mandatory [11]. Due to the adapted "generationDateAndTime", an adjustment takes place in the XML with each build. Thus, in most tools, a new ID is generated every time the FMU is built/exported. The GUID is to be interpreted as a string, so quotation marks or curly brackets are also valid. This string must be globally unique and must not be used more than once.

The modelName, the fmiVersion, and the GUID or instantiationToken must be listed in the modelDescription.xml. The remaining information is considered good practice and should be stored and maintained.

5.5 Handling Image/Icons in the FMU Container

The FMI standard allows an image to be optionally stored in the zipped container. For this purpose, an image labeled "model.png" is stored in the top level (FMI 2.0). This can be put on later during the simulation and used for visualization purposes. Ideally, an image should be stored that provides a recognition value of the real component. Conceivable Figures here are the logo of the component manufacturer, a picture of the real component or a CAD image. It would also be possible to store the logo of the FMI standard (cf. Figure 14). In the case of hybrid model use, the user would be shown whether it is an FMU or a native library element (cf. Chapter 8.1). Some FMU creation tools automatically store images in the container. It should be noted that the image does not lead any unintentional information about the content of the model (e.g. screenshot of the source code) to the outside. When integrating FMUs into the VC tool, the image can be used to make the simulation model of the entire system clearer. This is particularly desirable with a high level of integration of FMUs. The image/icon can be displayed in the respective block diagram (cf. Figure 14). The FMU is shown as a black box with the associated inputs, outputs and parameters.

Figure 14: Exemplary representation of a block diagram with the FMI logo as an icon

However, the image size should be kept small so that the handling of the files is not difficult, and the storage requirements are not unnecessarily large. Optionally, a .html file can be stored in the "documentation" folder, which contains information about the associated image rights. Currently, the integration of an image in many VC tools is not yet supported. In relation to Chapter 10 Called *further standards* there is also the option of storing or linking the icon in a higher-level layer (e.g. in the asset administration shell (AAS)).

Storing an image in the FMU container is currently considered optional. (relevant in the future)

5.6 Documentation integration

In addition, the standard also offers the option of storing the documentation (cf. Chapter 7.3) of the behavioral model. The format for the documentation should be .html or .txt. Thus, a corresponding export from other tools (e.g. Word) is also possible. PDF is not supported by default [11]. Instead of data storage, a link can also be stored that refers to the documentation. If it is provided via the Internet, it should be noted that the VC setup is often isolated in an encapsulated network and does not have access to the Internet. The reference to the document ensures that the FMU is as lean as possible but is not recommended in the context of the VC. However, the file size of the documentation must be considered so that data handling is not unnecessarily difficult. In the VC tool (co-simulator), it is important to ensure that even with extensive documentation, the performance (at the start of the simulation but also during the simulation) is not negatively influenced. In this case, the time for unpacking the FMU is favorable.

Overall, it is good practice to provide the documentation via the FMU container. In this way, the co-simulator (VC tool) can reference the documentation and open it in the same way as the native "help function". This makes it possible to link and open the documentation of FMUs to help with native behavior models. When the FMU is a black box, documentation is an essential part of VC. Especially in a high level of integration with FMUs, it is important for the user to be able to access the corresponding documentation directly via the VC tool (co-

simulator). This procedure also ensures that the documentation fits the FMU at hand and corresponds to the respective versioning. Separate storage of documentation and model can lead to discrepancies. Now, it is still common practice to offer the documentation to the user separately from the FMU as a PDF file.
In relation to the information provided under chapter 10 called *further standards* there is also the option of storing the documentation in a higher-level layer (e.g. AAS). In addition, the Documentation can also be read on the component manufacturer's website. In this context, appropriate references, IDs and links must be stored in the FMU. If the FMU creator (component manufacturer) offers different types of provisioning, the user must take care not to redundantly edit the documentation as much as possible.
The documentation must be prepared at least in English. Especially for use in German-speaking countries, a German version should also be deposited. Other languages that are often used in the context of the VC are Spanish, French, Italian and Chinese.

Documentation from FMU must be provided. Currently, the documentation can also be delivered separately (e.g. as a PDF). (integration into FMU relevant in the future)

5.7 Integration of external dependencies

In the FMI standard, it is possible to access external libraries or compilers (e.g. Python). Such dependencies are available to the user via a readme or via the documentation (cf. Chapter 7.3) so that it can carry out the necessary installations. In some cases, FMUs can only be simulated in connection with the generation tool. However, this is not the goal of a standard. In the context of the VC, FMUs must run independently. In general, external dependencies should therefore be avoided.

External dependencies must be avoided at all costs.

5.8 Using Logging at FMU

Since FMU forms an encapsulated black box, it is of enormous importance to keep log messages external. This can be done on the one hand via a log output, which provides various states (e.g. state machine), errors or diagnostics (often also present in the real component), or on the other hand via logging. The FMI standard has defined a specific procedure for this and defined various log categories. Especially in cases of error (for example, the FMU goes into an error state), the user is dependent on a log message to understand the reasons for the behavior (debugging). Without logging, the user is not offered the opportunity to analyze and understand the misconduct of the FMU in more detail, as he does not have any source code. Especially for complex FMUs, a logging functionality is very useful and helpful. In certain situations (e.g. when reading or writing external data) or when

establishing connections (e.g. TCP/IP), it is relevant to pass on the internal returns of certain functions to the user. Appropriate logging is also essential regarding licensing or time limited FMUs. For models with a temporal behavior, it makes sense to output the simulated time in certain situations. In this way, the user can check whether the FMU is called correctly in the co-simulator. Care must be taken not to log messages in every simulation cycle (doStep). Such an approach leads to the log file filling up and the analysis is made more difficult. Too much logging can cause performance problems. Good and targeted logging is the hallmark of a qualitative FMU.Logging helps the component manufacturer to optimize and further develop its FMUs. For example, he can request log files from the user, which are used for targeted analysis in the event of problems. However, it is important to ensure that the log messages can be viewed by the user. In this context, it should be mentioned that know-how may be disclosed here. Log messages should not reveal any information about the implementation in the real component. On the co-simulator side, the log messages must also be passed on to the user and should not lead nowhere. In most cases, the log outputs are passed on to the tool logging. Alternatively, it is possible to provide the FMU logs in a separate log file. Ideally, the user can decide which log categories (fmiOK, fmiError, fmiFatal ...) he wants to lead to the outside. In this way, targeted logging is made possible. The co-simulator must offer the option to specifically activate and deactivate FMU logging. If the log categories defined by the standard are used sensibly and supported by both sides (co-simulator and creation tool), you can dispense with your own internal log mechanisms. This includes, for example, parameters that can be used to activate logging and define the scope (e.g. debug, VC bit).

Targeted logging is recommended and offers enormous added value, especially when debugging.

5.9 Use of terminals

From FMI version 3.0 onwards, it is possible to define so-called terminals. Groups of variables are formed, and an additional layer is added to the input/output structure. In this way, physical connections as well as communication structures between FMUs can be mapped more easily [11]. It makes sense for VC tools to integrate these new scopes in the future. However, a comprehensive use of terminals requires standardization. For this purpose, the signals used, the associated data types and the respective units must be clearly defined. Especially regarding acyclic communication (e.g. IO-Link), terminals can significantly reduce the effort involved in linking the individual FMUs. Also, in the context of pneumatic or electrical networks, terminals facilitate interaction between FMUs and allow for a higher level of detail.

Communication between FMUs via terminals is currently considered optional. (relevant in the future)

6 Model Scope

6.1 Level of Detail (LoD)

The level of detail of a behavioral model is certainly one of the most discussed points in the context of FMUs. Thus, a simple superficial model is contrasted with a highly complex, detailed replica of the real component (cf. Chapter 6.1.4). In the context of VC, a reasonable compromise is required, as the simulated cycle time must be adhered to in any case. Performance issues due to overly detailed modeling must be avoided. The following sections are intended to support the component manufacturer in finding an appropriate level of detail in the context of VC. Behavioral models in the FMI standard can also be used for the phases of concept development, engineering, and operation of the automation plant. However, this guide is limited to use in VC.

6.1.1 Logical behavior

Especially regarding VC, logical behavior forms an essential part of the behavioral model. Here, FMUs are expected to provide a precise and logical behavior towards the control system. The level of detail must be high enough so that the control system does not notice any difference to the real component. This is the only way to connect the control program to the digital twin without any adjustments. For this purpose, the FMU must implement the complete IO behavior regarding connection to the PLC. In general, parameters of the real component must also be defined in the same way. In addition to directly wired IOs, process data transferred via bus communication represents an essential part of the logic model. In the case of various telegrams (e.g. PROFIdrive), specific input/output words are required for connection to the PLC and thus must be provided by the behavior model. These telegrams must be varied via parameters or different behavioral models. Especially with complex components, it is sometimes sufficient to implement only elementary parts of the logic. Many special functions may only be used in exceptional cases.

Error triggers must be mapped accordingly on a component-specific basis. Diagnostic outputs, which are read in and evaluated by the PLC, must be mapped in the behavior model. In the minimal case, an input must be looped through to the diagnostic output.

Display and input elements of the real component are not necessarily relevant for VC and are therefore (in most cases) to be considered optional. Examples of these elements are indicator LEDs that provide information about the current operating status (cf. Figure 15). Input elements are, for example, buttons that are used for manual control or parameterization. Especially in the context of operator training, modeling of these elements offers significant added value.

Figure 15: Abstract representation of input elements in the form of indicator LEDs and buttons ("+"/ "-")

Regarding IOs (PLC signals), the FMU must behave analogously to the real component in terms of logical behavior.

6.1.2 Temporal behavior

Often, parts of the logical behavior are also enriched with temporal behavior. Examples for this are life bits (*periodic clock signals for functional monitoring of the component*) or time-controlled pulses and edges. The processing time in the component (often in case of acyclic communication) also represents a temporal behavior.

The degree of detail of the physical or mechanical behavior needs to be aligned with the temporal behavior to describe processes realistically. In this way, cycle times can be analyzed and optimized. A simple temporal behavior (e.g. abstracted linear ramps for drives or cylinders) enables better safeguarding in VC. Ideally, travel paths are simulated as realistically as possible instead of making linear assumptions about acceleration and deceleration. In the simplest case, this can be realized via a distance-time diagram without dynamic behavior. Such a time behavior allows movements to be displayed much more smoothly in the visualization and processes to be simulated better. Figure 16 shows an exemplary travel path via position and speed in relation to time. Different levels of detail are considered. In the first case, a jump in position takes place and speed is not modeled. In the second case, a jump in velocity is assumed, which linearly approximates the change in position. In the third case, the movement is described based on acceleration and deceleration. The level of detail rises increasingly. For movements in the context of the VC, the third option is preferable, as jumps can lead to problems in the visualization.

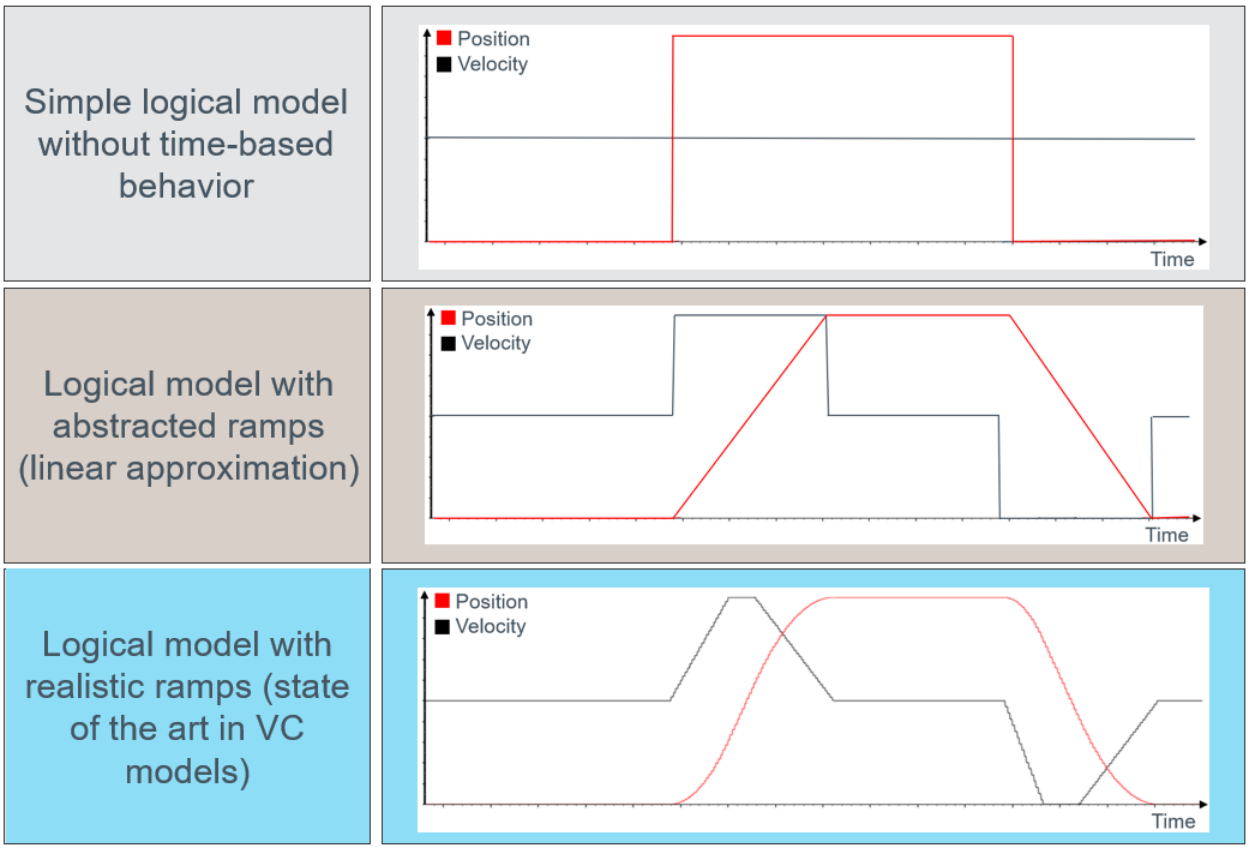

Figure 16: Exemplarily sketched time-behavior of a drive with position and speed in different degrees of detail

Timed signals and processes (movements) must be mapped so that monitoring times and sequences can be validated in the PLC.

6.1.3 Physical behavior

Complex physical behavior and modeling of friction, inertia, forces, moments, thermal and chemical behavior, etc. is not required in most applications. It is also not necessary to simulate detailed electrical behavior and dynamic mechanical processes such as oscillations and vibrations. Such physical aspects are usually analyzed and considered up front in the phases of concept design, dimensioning, and design (e.g. by sizing tools, multi-body simulations, ...). For this reason, VC is often based on a logical, idealized material flow and leaves physics out of the equation.

In a realistic simulation of the material flow using conveyor technology, physics must also be considered. For example, physical relationships must be implemented in the component. Internal inertia, friction, etc. are to be

modeled by the component manufacturer. It is important to note that with a detailed physics simulation in the behavior model, the complexity of the entire simulation model increases. For example, force and torque feedback from the plant mechanics to the behavior model must be provided. For this purpose, correct friction values and weights as well as the inertia must be stored in the visualization. Regarding this, the performance and the associated cycle time in the visualization tool is often a limiting factor. The interfaces between the behavioral model and the visualization must also be performant enough to enable the adjustment process between the drive and the physical feedback. Currently, this is not always the case, which means that highly detailed component models can often not be meaningfully integrated into the digital twin. As a result, simulation results are often not realistic. A time- or clock-synchronous coupling between the VC tools (cf. Chapter 9.11) can support in this context.

The simulation of a non-idealized motion process (see previous paragraph) is rarely necessary in the context of VC in the automotive industry. In other areas, such as special machine construction, such approaches are now playing an increasingly important role.

Currently, the modeling of physical relationships (friction, force, inertia, ...) is usually not required. In the future, the requirements will increase.

6.1.4 Definition of different levels of detail

A concrete definition of a specific level of detail is not possible across the board as different approaches are necessary depending on the respective component and the scope of VC. For example, the level of detail of an inductive proximity sensor must be defined differently than that of a technology system such as a welding gun with subcomponents. By defining different LoDs, a suitable model scope can be selected according to requirements. A distinction must be made between "Basic" and "Advanced" functionalities. Within the framework of this guide, a three-level subdivision is made (cf. Figure 17):

- **S: "abstracted in time, so much that the PLC doesn't complain"**

  Describes the behavior of the IOs in relation to each other. It is only modeled as much as is needed for the operation of the component without the PLC throwing any errors. This level of detail forms the counterpart to the PLC function block and can be developed based on it. In this way, the relevant bus interface (e.g. PROFINET) is mirrored. This level of detail only implements common error cases via error triggers. To make VC possible, temporal sequences in the signal flow as well as simple movements in the form of ramps (if available) must be implemented. This level of detail meets the minimum requirements

and is comparable to current library elements in common VC tools. The LoD in the basic version ("S") should not go beyond the information provided in the component documentation.

- **M: "highly detailed logic model with temporally correct behavior"**
  Represents the logical behavior in analogy to the real component. All cases of error (cf. Chapter 6.6) and parameters described in the component's documentation are modeled. This model provides a higher level of detail than previous behavioral models (in VC tool libraries).

- **L: "behavior analogous to the real component including physics"**
  This level of detail describes a "one-to-one" representation of the real component. Certain parts of the real firmware are present in the model. Important physical relationships are taken into account in the model. This enables a meaningful cycle time analysis. Investigations of air pressure, inertia and other physical relationships are also possible. It should be noted that such detailed FMUs (in terms of the number of parameters) need to remain simple to use and connect.

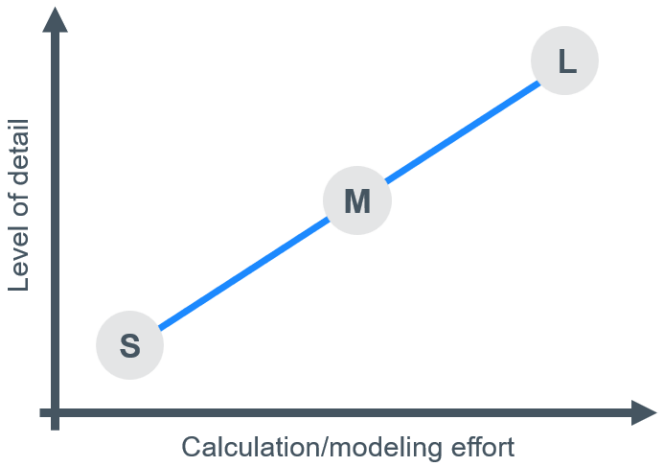

Figure 17: Classification of the different levels of detail ("S"/ "M"/ "L") with regard to the realism of the model and the calculation and modelling effort

Less extensive automation components without mechanical behavior, such as sensors or communication modules, are often limited to the "S" and "M" levels. Here, it is important to map the basic logical functions. Complex modeling scopes for the "L" level may for example be the simulation of the current flow over the respective pins and ports to map overloads. More complex components, such as electric gripping systems or drive systems, are difficult to map in the "S" stage, as many process data and parameters have to be integrated here. It is possible that only the levels "M" and "L" exist for such components. The scope is also strongly dependent

on the component type itself. Electrical components (e.g. drives) have a much larger scope, especially in logic and firmware, compared to purely mechanical components (e.g. cylinders).

In general, different levels of detail can be used in the engineering process for the respective application scenarios. It may be necessary to validate PLC standard modules (building blocks) of the components using the "M" level to test all possible error cases. VC based on standard modules that have already been validated will then take place via level "S". The model "S" must be detailed enough so that VC can be carried out based on it. Ideally, the different levels of detail have the same inputs and outputs (names and data types), so that they are easily interchangeable. The simulation model can thus be easily adapted with a detailed model if required.

The different levels of detail can also be provided and licensed differently (cf. Chapter 7.6). Overall, highly detailed models (cf. Chapter 11) that the FMU (analogous to the real component) could be misused for reverse engineering purposes. Even if the code is encapsulated, logical relationships can be evaluated.

For the respective application, components can be modeled in different levels of detail. The guide distinguishes between three levels of detail.

6.2 Connection to the control system (bus communication)

In general, it should be taken into account that not all automation components are connected via fieldbus systems (e.g. PROFINET, EtherCAT, ...). In particular, non-electrical components (e.g. pneumatic cylinders) sometimes communicate only indirectly (e.g. via valve terminals) with the control system, which eliminates the need for a PLC connection in these cases. In the context of electric drives or other bus participants, this section represents an important basis, also in terms of the level of detail.

In current VC projects, the behavior model is connected to the control system via the VC tool. Connections to the real and virtual controllers are already implemented in these tools. The bus itself is emulated in many cases. In the short to medium term, the FMUs must be connected to the control system via the respective VC tool (cf. Figure 18). In general, the goal of VC is to safeguard the PLC logic. It is assumed that the bus telegram works correctly in the real component. For this reason, the realistic mapping of bus protocols is currently not relevant.

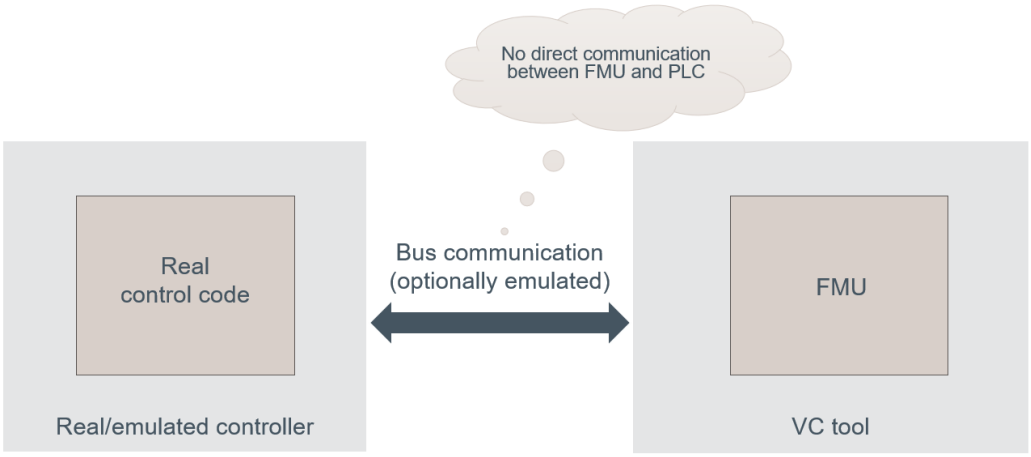

Figure 18: Connection of an FMU to the real control code (PLC) via a VC tool (co-simulator)

In the long term, however, a direct connection to the virtual/real control system by FMU would be conceivable. The model then acts, analogous to the real component, as a participant in the corresponding bus (e.g. PROFINET, EtherCAT, ...). In addition to the complex implementation, there may also be challenges regarding the hardware (Ethernet/network cards, switches, ...) due to bus topologies.

In the context of FMU creation, it is possible to use components of the real bus connection for generation and parameterization. For example, standardized XML descriptions, such as Generic Station Description Markup Language (GSDML) or IO Device Description (IODD), can be used. This enables semi-automated FMU creation and reduces the manual implementation effort. The information obtained from this can also be used for higher-level descriptions and standards (AutomationML (AML), Asset Administration Shell (AAS)) (cf. Chapter 10).

The connection to the respective (real/virtual) controller is currently taken care of by the VC tool and is not relevant for the FMU creation.

6.3 Integration of acyclic communication

In addition to the cyclical communication discussed in the previous section, some automation components use acyclic communication. Such communication is often used to exchange parameters and to transmit less time-critical data. Currently, this functionality is often not considered in behavioral models. Even if the real component supports acyclic services, these are sometimes not used in the PLC program. In the future, however, the integration of acyclic communication will continue to increase, which makes its modelling in FMU unavoidable. This development is already becoming apparent in special machine construction.

In addition to bus-specific protocols, there are also additional acyclic connection options. These include, for example, OPC UA, TCP/IP or IO-Link. The use of acyclic communication continues to increase with the increasingly complex automation components. Currently, acyclic communication is often mapped via the VC tool or bridged via an external tool. This separation between behavioral model and acyclic communication (VC tool) poses a challenge for the interface to the behavioral model (FMU). For example, the imported parameters and settings must be passed on to FMU. However, these are often several hundred variables. Especially in the case of non-standardized protocols and communication mechanisms via TCP/IP or OPC UA, it makes sense to integrate the communication into the FMU. In the current situation, acyclic communication based on bus standards (e.g. PROFINET, EtherCAT) can be implemented via the VC tool. Technically, integration into an FMU is possible, but should not be done at present due to performance and, above all, stability reasons. If the communication takes place outside the FMU, the model must be tested and validated using common VC tools.

Communication methods such as TCP/IP or OPC UA can only be implemented as long as the real component also has these mechanisms (replica). For internal functions, such as storing data, parameterization (if not analogous to the real component), communication between FMUs, connection to the network, etc., the development of such communication methods due to cyber security issues should be avoided.

In the current VC tool environment, the following recommendation is made for the integration of acyclic communication (cf. Table 2). This is a rough guideline, which must be evaluated specifically in the respective context:

Communication	**Comment**	**Integration FMU**	**Integration VC Tool**
acyclic PROFINET (Siemens)	Emulation or connection to virtual control API required	(x)	x
acyclic EtherCAT (Beckhoff)	Emulation or connection to virtual control API required	(x)	x
TCP	Usually very component-specific	x	(x)
UDP	Usually very component-specific	x	(x)
OPC UA	Usually very component-specific	x	(x)
acyclic IO-Link	since no direct communication with the PLC, usually kept abstract	x	(x)

Table 2: Overview of the various acyclic communication channels and their suitability for integration into FMU (x = recommendation; (x) = technically possible, but not recommended)

If acyclic communication is implemented in the VC tool, a cyclic interface to the FMU must be implemented. This interface cyclically forwards the imported parameters to the FMU. It must be designed as simply and uniformly as possible. Analogous to cyclic communication, standardized XML descriptions of the real component also offer the possibility of providing information for modeling and parameterization. For example, in the case of IO-Link, the IODD file can be used for this purpose.

If acyclic services are not present in the provided behavior model, but are required for the VC, the FMU creator (component manufacturer) needs to be contacted. On the user side, it is important to communicate such needs at an early stage so that the create side becomes aware of the relevance of acyclic communication.

In general, the topic of storing FMU-internal states and values is becoming increasingly important in connection with acyclic communication. Many acyclic parameters are written when the controller starts. In the event of an FMU start-stop, these values are lost, which can also cause problems with VC. Saving FMU states can be done via canGetAndSetFMUState and canDe/SerializeFMUState (cf. Chapter 5.4.2).

Acyclic communication plays an increasingly important role in terms of parameterization. Depending on the type of communication, different implementations are possible.
(relevant in future)

6.4 Scope of an FMU (individual component vs. overall network)

In many use cases, the question arises as to where to set the system boundaries of an automation component in the context of a behavior model for VC. One example is a pneumatic cylinder. It is possible to model and provide the cylinder as a combination of valves, end stops, sensors and the actual cylinder (cf. Figure 19). Alternatively, the task of combining individual models falls on the user which increases flexibility. The structure is analogous to the real overall network. Another example is a positioning axis, which is composed of power supply, drive, shaft, frequency converter, encoder and (optionally) sensors.

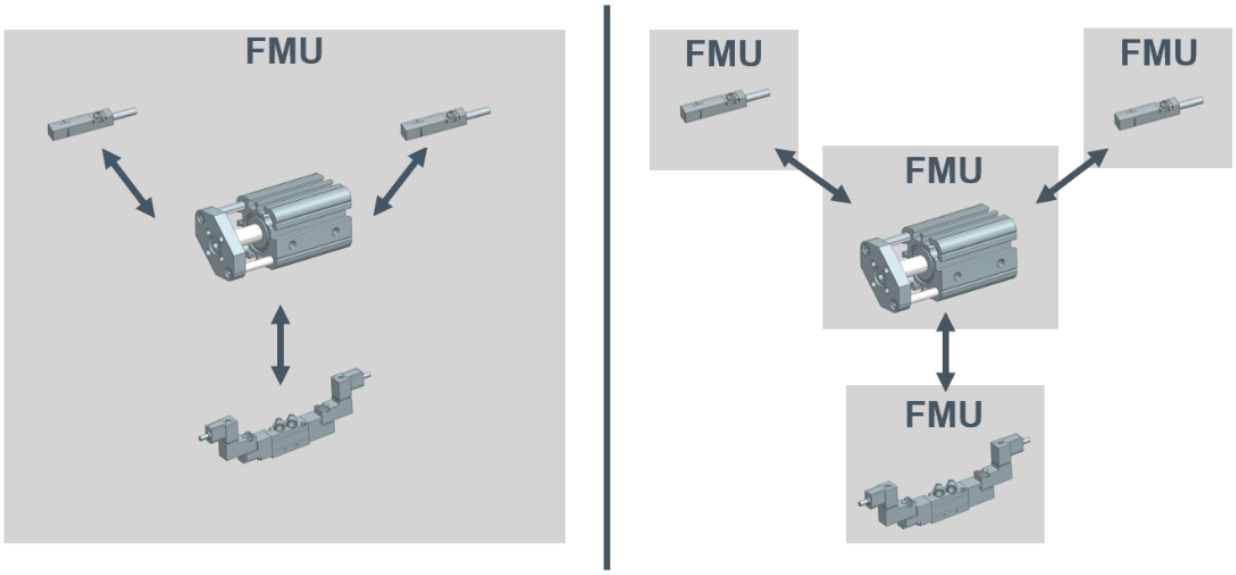

Figure 19: Differentiation between module and single-component FMUs using the example of a pneumatic cylinder

Depending on the scope of the model, existing assistants and automatisms for model creation may also have to be adapted. The combination of several components into an FMU makes it difficult to assign the signals via equipment/reference identifiers, as entire modules must be linked. In many cases, the VC model is generated based on the bill of materials (e.g. EPLAN). Here, module FMUs complicate the process, as several articles/order numbers are mapped via an overall FMU. Thus, the option of a combination of individual components by the user is in most cases the more suitable one.

The more finely granular the individual components are resolved, the more difficult it is to define the system boundaries. To this end, the interfaces of an FMU must be designed to be as generalized and open as possible so that component FMUs from different manufacturers can be linked to each other. The delimitation of the component is analogous to reality. In this context, attention must be paid to numerical instabilities. The FMU must be developed as robustly as possible. Critical situations must be intercepted in a targeted manner. In general, only the IOs for control are of interest to the VC.

If an entire assembly is provided by a single component manufacturer, it makes sense to provide the whole behavior model as a single unit. The behavior of the entire assembly is described by an FMU or SSP (representing several FMUs, cf. Chapter 10.1). The latter would have the advantage that the scope of modelling can be combined depending on the situation. If an assembly or device is assembled from components from different manufacturers

(e.g. motor and converter from different manufacturers), the individual components must be combined with each other. One example is a gripper that is connected to a valve terminal.
In the case of encapsulated applications with their own control system (e.g. welding, bonding or screwdriving applications), an FMU for the entire system can be provided directly by the manufacturer. He knows both the internal behavior and the components used and has relevant documents and specifications at his hand. Today, a behavioral model for this purpose is usually created by the user. Such a model only approximates the system and entails great effort and uncertainty about the model quality.

The scope of the model cannot be fixed across the board and depends on the respective component.
Rule of thumb: Provide a separate FMU for each component with its own reference identifier.

6.5 Individuality of FMU (specific vs. abstract model)

Parameters allow FMUs to be adapted for specific use cases. This makes it possible to map entire product families using a single behavioral model. As an example, a cylinder FMU can be considered, which can be variably adjusted in length via parameters (cf. Figure 20). This behavioral model therefore describes the product family of cylinders. In this way, the component manufacturer must provide fewer behavioral models. The effort is reduced on the creator side. However, the complexity of the connection on the user side increases as many parameters must be set. This also has a negative impact on the model performance. In addition, generalized FMUs require many options, settings, load curves, etc., which may make it difficult to create behavioral models. This approach also requires the definition of a clear connection to the corresponding real components. It must be clear to the user which FMU with which parameter set he has to use for the corresponding real component.

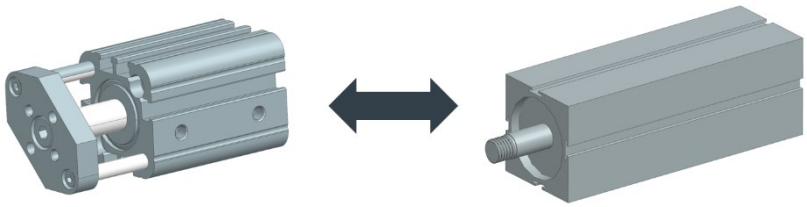

Figure 20: Comparison between an article-specific cylinder (left) and a flexibly parameterizable cylinder (right) analogous to CAD

As an alternative, a separate behavior model could be created for each component (item number). The model is pre-parameterized and can only be used for the specific application. In this way, potential errors of incorrect parameterization are avoided. The assignment of models to the respective real components by the user is also

simplified. In addition and analogy to the article-specific CAD model, a specific behavior model is provided. Using this approach, the effort on the creator side and in model maintenance (e.g. versioning, optimization, ...) is higher. In both cases, the connection to the simulation model can be automated. For example, for a certain article, either a specific FMU or a general FMU must be referenced together with a corresponding set of parameters. Parameterization can be done via a wrapper, which may also be responsible for signal mapping. In general, however, this distinction is strongly dependent on the respective component (e.g. inductive proximity sensor vs. complex drive) and cannot be considered in general.

It is also possible to supply a component specific FMU for each real component (serial number). However, the administrative burden here is enormous. Furthermore, the FMU models are usually required before the order/delivery of the real component. This procedure has therefore proven to be impracticable.

The degree of abstraction cannot be defined across the board and depends on the respective component.
Rule of thumb: As long as the number of parameters remains manageable, a general model can be provided.

6.6 Error triggers in VC

In the context of VC, it is of great importance to test the PLC for possible error cases. Thus, in addition to ideal behavior, certain misbehaviors must also be implemented in an FMU. These can often not be recreated or only with great effort. A distinction must be made between errors that influence the behavior of the component (other error cases) and error messages that are passed on to the PLC via the diagnostic output of the component (if available) (cf. Figure 21). If the automation component has a diagnostic output towards the PLC, this must be integrated into the FMU in any case.

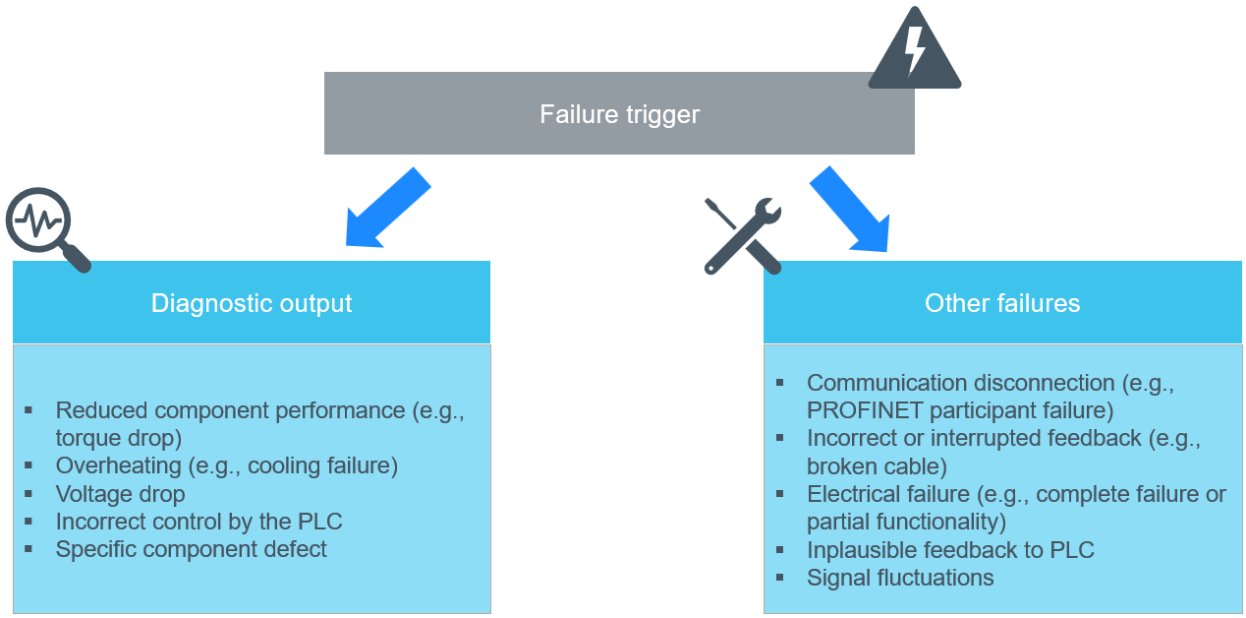

Figure 21: Dividing the error triggers into messages via the diagnostic output and into other error cases with associated examples

Examples of errors include cable breaks in safety (other error cases), overheating of a motor (diagnostic output) or failure of a capacitor (diagnostic output). Another error case is, for example, the separation of the incoming actual position from the visualization (encoder error). As a result, the drive never reaches the desired target position and continues to rotate (other faults). In addition, contouring errors, for example, can be integrated via an error trigger.

In addition to these hardware errors, there are also software errors. These errors are caused, for example, by implausible inputs from the controller or temporal monitoring (delayed feedback). Examples are comprehension problems when switching programs (e.g. parameter change). In an FMU, for example, processing times can be incremented by a factor or delay, so that a monitoring error is triggered in the PLC. For other component-specific fault cases, test tables from the VC are to be used to derive corresponding requirements. For more examples of error triggers, see the following listing:

- Disconnection of communication from/to the controller
- Incorrect or interrupted sensor feedback
- Reduced performance of the component (e.g. low torque, low force, low volume flow, low pressure)
- Overheating or failure of the cooling system

- Voltage drop or complete power failure
- Faulty safety signals (e.g. signals decrease/do not match)
- Incorrect operation of the system

To test such error cases, integration into the behavior model is required. The control is usually designed via so-called "error triggers". These are to be understood as a kind of "pseudo-inputs“ since they do not exist in the real component. For this reason, it is also essential to include the error cases in the FMU documentation.
In the case of a small number of error cases, the triggers can be easily mapped via binary signals. To ensure that the behavior model does not become confusing in the event of many error cases, an integer signal and a trigger bit can also be used to trigger the respective error case. It is often sufficient to simulate the errors at the top level and only provide feedback to the PLC. In this way, desired faults can be defined by the user and passed on directly to the PLC. The error number is to be set by the user via an input.
In addition to individual errors, complete error scenarios and test cases can also be mapped and described. These must be listed in the documentation and possibly triggered via special trigger bits. This means that fully automated tests can also be carried out. Above all, it is necessary to check those errors that occur very rarely in reality. Regarding automated error tests, an associated acknowledgement is required. It is often difficult to put the system back into operation after a fault. In this context, a confirmation bit is required so that the adjacent error can be reset. If the real hardware has a reset/acknowledgement button, this must also be implemented in the behavior model.
In the event of fatal errors, both the PLC and the behavior model may have to be restarted to exit the error case. In this case, it is possible to restart the model via an FMU reset in the co-simulator without having to restart the simulation. These automated tests are to be managed via external software (e.g. Expecto).
In the case of complex components, there is a very large number of conceivable error cases. If not all of these have been thought through and implemented in the FMU, the user can implement some of them via the VC tool. Here, mechanisms are often provided to trigger manual or scripted error and disruption events. The VC tool also enables the connection of auto-test tools.
Since the communication of the component via the bus is currently emulated by the VC tool, a failure of the bus participant must also be simulated via the VC tool. An example would be the failure of a PROFINET participant. In FMU, no trigger and error state need to be provided for this. Related to the failure of a bus participant, there is the connection to a power supply. To simulate a failure of a network of components on the same circuit, the behavior model must be informed via an input whether current is currently available to the device. This can be done either via an "enable" bit or via a real input, which simulates the applied voltage.

In addition to error triggers for components, these must also be stored for processes. Since the processes are usually assumed to be idealized, stored error scenarios are required. For example, individual sensors must be triggered to get into the desired state. Furthermore, the possibility must be created to trigger "NIO" processes (e.g. during bonding or welding). In process technology, it often takes a long time to reach the respective states. Error triggers can bridge long "waiting times" in the VC.

Fault triggers increase the added value of VC and enable tests that can be reproduced on the real system only with a lot of effort. Error triggers increase the quality of FMU.

6.7 Controller-specific adjustments

The connection of complex automation components to the corresponding controllers often takes different forms. In addition to the various manufacturer-specific protocols (e.g. PROFINET, EtherCAT, ...), the components can also be connected to the controller via different structures. In some cases, small adjustments to the FMU are necessary for this. Examples of this are byte flipping or splitting words into individual bits and bytes. This can be done directly in the FMU using suitable parameters so that the connection effort for the user is kept to a minimum. However, this increases the number of interface signals and the parameterization effort. Alternatively, the component FMU can also be connected to "auxiliary FMUs", which prepare the signals appropriately (cf. Chapter 10.1). Furthermore, it is possible to make these adjustments directly in the VC tool.

In general, FMUs should be kept as general as possible. Specific adjustments for individual users should be avoided and implemented via an alternative approach. The behavioral models replicate the real components. If these can be parameterized for different bus protocols, this functionality (e.g. byte flip) shall also be implemented in the FMU via parameters.

The following exemplary recommendations can be made (cf. Table 3):

Usecase	Implementation in FMU	Via Auxiliary FMU	Via Behavior Modelers
Byte-Flip	x	(x)	(x)
Word to Byte	IO Word	x	x
Word to Bit	IO Word	x	x
Byte to Word	IO Word	x	x
Bit to Word	IO Word	x	x
Unit Customization	x	(x)	(x)
Standardization	x	(x)	(x)

Table 3: Overview of various control-specific adaptations and their implementation options (x = recommendation, (x) = technically feasible, but if possible, to be provided for in FMU)

The FMU should be created as neutrally as possible. Control-specific characteristics can be adjusted using auxiliary FMUs or the VC tool. Integration into an FMU only makes sense if the real component can also be parameterized in this respect.

6.8 Role of the Visualization (CAD)

In most cases of VC, the behavioral models (FMUs) are coupled to a visualization (e.g. YAMS, SIMLINE, NX MCD). Among other things, the material flow is mapped, or collision checks are carried out. The visualization offers the possibility of safeguarding additional aspects via VC. Above all, processes and material flow are difficult to map with pure behavioral models. Visualization also makes it easier to ensure safety. For example, light grids or individual beams can be modeled and passed on to the respective behavioral model. In addition, the visualization provides the user with visual feedback. The digital twin is clear and easier to interpret.

Not every component needs an interface for visualization. This includes, for example, communication modules or temperature sensors. For many components, however, a connection to a 3D simulation is important. These include sensors, drives, cylinders and robots.

The interface signals towards the visualization are not defined in a standardized way. For example, the units used (e.g. millimeters and meters) differ. In addition, percentage values may be passed to the visualization instead of direct process values. One example is a pneumatic cylinder that passes a value between 0 and 100% to the visualization. From this percentage value, the resulting travel is derived in the visualization.

In case of a cylinder, the sensors for the end stops ("extended"/"retracted") can be mapped directly in the behavior model and in the visualization (cf. Figure 22). Especially in case of robot couplings or mechanical assemblies, it makes sense to model the sensor feedback through visualization. In case of clamping cylinders or

sensors that are not represented in a design, the connection via the behavior model is often sufficient. In case "1" (Figure 22), the controller receives direct feedback from the behavior model. For this reason, the cylinder can also be operated without visualization. In version "2", the behavioral model provides a position to the visualization, which detects the position of the cylinder via sensors and provides feedback to the PLC. In option "3", the state from the visualization is fed back to the behavioral model. The behavior evaluates the result from the visualization and reports it to the PLC.

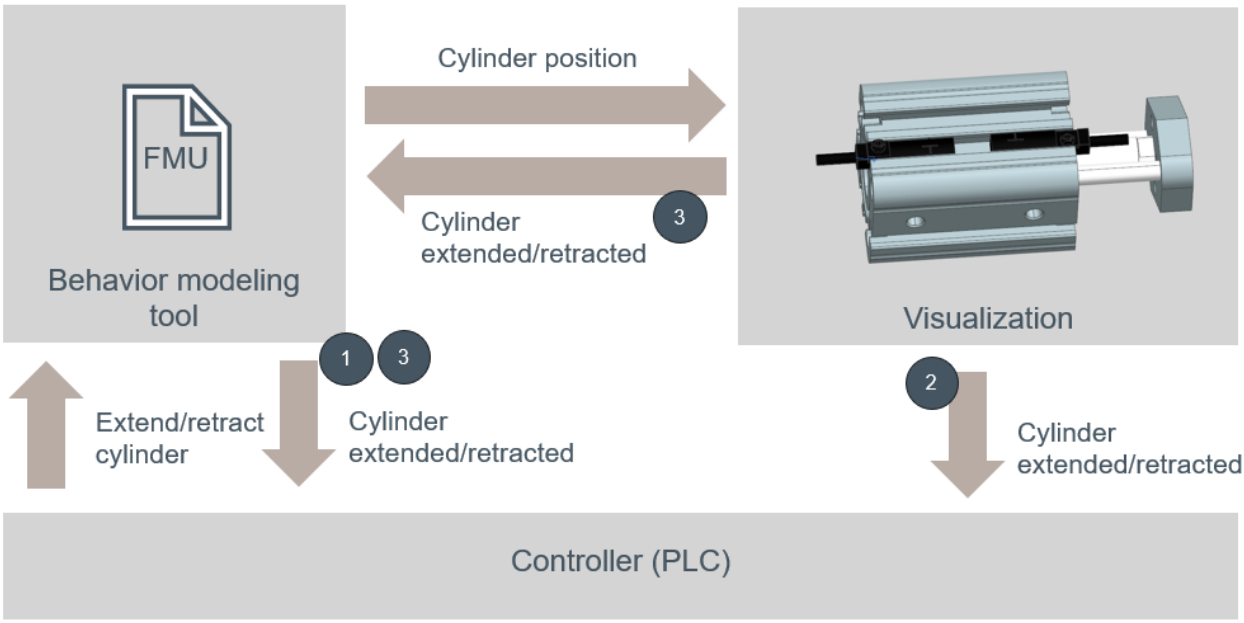

Figure 22: Various possibilities of feedback to the control system using the example of a pneumatic cylinder with end stops (1=without visualization; 2=directly from visualization; 3=via visualization and behavior model)

In general, it is recommended (depending on the area of application) that the FMU can be operated as independently as possible. Essential functions of the individual components should be simulated directly via the FMU. This increases the stability and flexibility of the digital twin. The number of interface signals decreases. In addition, VC (e.g. IO test, simple logics) can be carried out at an early stage without visualization. Later, the visualization serves as support. In the case of mechanical components, a connection to the visualization (e.g. drives with feedback from CAD) can ideally be selected and deselected via a parameter. Depending on the simulation chain and test case, it is possible to activate or deactivate the corresponding functions.
Especially in conjunction with highly complex physical behavior models (cf. Chapter 6.1.3), visualization plays a major role. If, for example, forces and torques are fed back to the FMU, a corresponding physics simulation is

required in the visualization. This involves specifying parameters such as mass, friction coefficients and inertia accordingly. The prerequisite for this is that the visualization tool provides a physics-based solver.

Many behavioral models interact with a CAD model (visualization). For this purpose, appropriate interfaces must be provided in the FMU.

6.9 Outlook: Modelling of processes, buildings, and products

In addition to automation components, processes often need to be mapped in the digital twin. The creation of process FMUs is not in the scope of the component manufacturer. In this context, the process knowledge resides with the machine manufacturer or the OEM. Modules with sub-processes are also often provided (e.g. adhesive applications, welding guns, etc.). In this case, FMU creation by the module manufacturer is recommended. In many cases, simple logic models are sufficient. Ideally, these should be generated automatically so that the creation effort is minimized. In the case of process FMUs, inputs and parameters are used to enable operator training. Parameters can be set and optimized by adjustments during the simulation. Error cases can be easily simulated (cf. Chapter 6.6).

In addition to processes, models of building layout and infrastructure can also be mapped via FMUs. This includes, for example, compressed air supply or the electrical system in a hall as well as external influences such as temperature or humidity.

Especially in the case of complex products, these must be modeled for a meaningful VC. In the automotive industry, for example, vehicles communicate with their test benches in end-of-line test benches. Serial numbers are often read out or other parameters are stored in the product. These use cases can also be described independent of the tool via FMUs.

Similar to automation components, processes, buildings, products or subsystems (modules) can also be modeled and connected using the FMI standard.

7 FMU creation and handover to the user

From the point of view of the plant manufacturer and operator (OEM), FMUs are provided by the component manufacturer for all/new components via a platform in the target image. This has the advantage that the plant manufacturer can download the component models if required and integrate them into the simulation model. Creating FMU on demand, which is often done till date, has the disadvantage that it usually takes a long time to deploy. The model can no longer be used in the corresponding VC project. As a result, the plant manufacturer must generate its own behavioral models (FMUs). In the target image, the component manufacturer makes the

corresponding behavior model available to the plant manufacturer/OEM as FMU during release for new components.

7.1 Data consistency (use of engineering data)

Ideally, behavioral models are exported directly from the component firmware in the medium and long term. In this way, adjustments in the real component can also be transferred directly to the models. Additionally, the costs for model creation are also kept low. A direct adoption and integration of the original firmware brings a high level of complexity in the model and may also have an impact on performance in the co-simulator. Often, a direct adoption of the software is not possible or only with difficulty, as there are various dependencies and limitations from real firmware development. The mechanical behavior (e.g. with cylinders) is not part of the firmware and therefore cannot be derived from it. Similarly, error triggers must be implemented in the FMUs (cf. Chapter 6.6), which are also not part of the firmware.

Since the behavioral models are currently mostly generated without the knowledge of the component manufacturers, the scope is usually limited to the reference manual. Simplified models are currently used in the VC, which only describe parts of the firmware logic. Many component manufacturers are currently developing the logic of the FMU models parallelly with the firmware. To minimize implementation effort, component manufacturers create libraries in their FMU export tool. During FMU creation, individual library elements are interconnected with each other.

A complete adoption of FMUs from the firmware is difficult. Error triggers and mechanical relationships must be expanded in addition to logic. An automated (partial) generation of FMUs is to be strived for in the medium term.

7.2 Tools for creating FMUs (FMU export)

The FMI standard is gaining more and more attention. For this reason, almost 250 tools now support the handling of FMUs. A list can be found under the following link: https://fmi-standard.org/tools/ [35].

In principle, there are no specific tools in the context of FMU development for the VC for creating the behavioral models (cf. Figure 23). A suitable tool must always be found considering the company context. A general statement about the suitability of special creation tools in the form of a ranking is not meaningful. However, it is important that the exported FMUs meet the requirements defined in this guide. This must be checked on a case-by-case basis.

In general, the tools and environments for creating FMUs can be divided into two categories. On one hand, various simulation and modeling tools offer an FMU export (e.g. AMESim, MATLAB/Simulink, OpenModelica). The second option is the direct creation of FMUs based on the FMI standard. Here, various FMU templates are available. The

FMUs are developed using programming languages such as C or C++. In general, it is the task of the component manufacturer to identify the tool that suits them. Since the lifetime of the FMU is linked to the real component, it is important to ensure that the selected tool is also suitable for medium or long-term operation. Otherwise, no adjustments to the FMU models are possible. The requirements for the FMUs from the VC context must be considered when selecting the tool. Otherwise, there will be problems with the integration into the VC tool landscape. An example of this would be the multiple instances of FMU models via the flag "canBeInstantiatedOnlyOncePerProcess" (cf. Chapter 5.4.2).

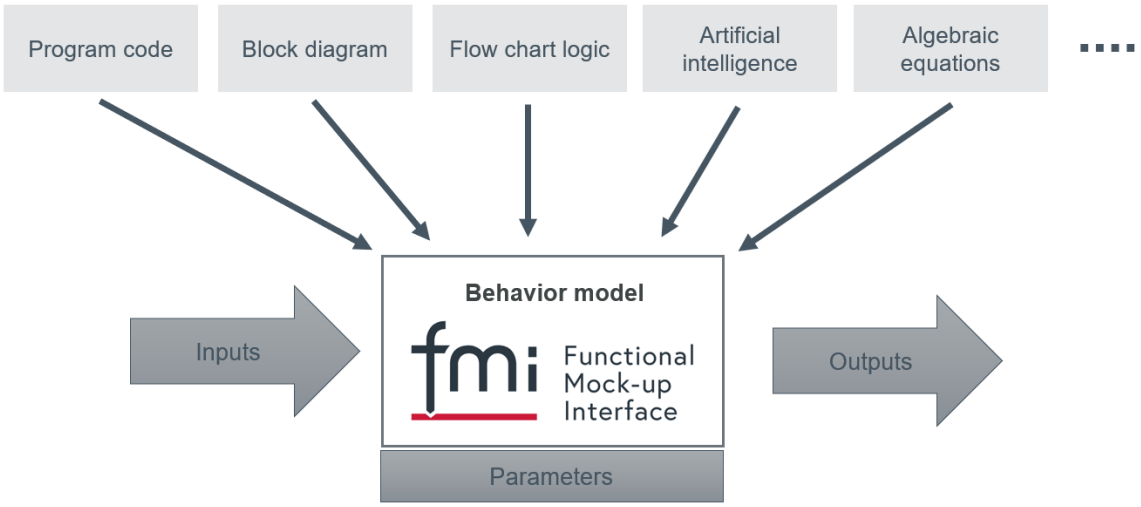

Figure 23: Overview of the different modeling options of FMUs

Simulation and modeling tools (1):

- Currently they are frequently used to provide FMUs for the VC in a less complex level of detail (developed in parallel with the firmware).
- Enable rapid FMU creation since the exporter is already implemented and modeling is often quite intuitive. For this reason, the tools are often used as a short- to medium-term solution. A deep familiarization with the standard and validation of the exported FMUs is not necessary. The simulation and modeling tool is responsible for a conformant and stable output of FMUs. Adjustments by the user are impossible or difficult to implement.
- Especially in the context of physics simulations, extensive libraries are often stored here. The settings can be used to select different solver methods and influence the calculation of the differential equations in the FMU.

- Some of them support both import and export in various operating systems (Windows and Linux).
- They usually also offer a co-simulator with which the FMUs can be validated.
- Set certain flags (cf. Chapter 5.4.2) automatically.
- The user is limited to the scope of the respective native libraries.

FMU-Templates (2):

- They are usually based on a high-level language (C, C++) and are therefore similar to firmware development. An integration of firmware into the behavior model is possible in this way.
- Can be integrated into a development environment (e.g. Visual Studio), which the component manufacturer may have in-house. (no new acquisition necessary)
- Can be extended to the respective purpose. The integration of additional FMI functionalities is possible if required.
- By using C/C++, extensive libraries/methods are available. For example, complex functions such as data storage, an API call or TCP/IP communication can be mapped.
- Version control and management tools from software development (such as Git, SVN) can be used. This creates analogies and synergy effects in firmware development. Examples include conventions in the variable name, code formatting, the use of well-known libraries or code commenting.
- Targeted logging is possible. This makes debugging the FMU easier.
- The stability of FMU is in the hands of the developer. Crashes and incorrect entries can be intercepted in a targeted manner.
- Since the modelDescription.xml is often generated manually or using auxiliary tools, a cross-check using the FMU Checker (cf. Chapter 7.4) is of great importance.
- Attention must be paid to high-quality software development and implementation, as improper handling of resources and methods can lead to the crash of FMUs. The FMUs must be validated carefully.

VC-Tools (3):

- Currently, no VC tool offers the export of FMUs. Only FMU import and simulation are supported. However, it is generally not excluded that VC tools enable the export of natively created behavior models or library elements.
- Not every tool is suitable for FMU export. There are dependencies on the tool's structure, licensing, etc. The effort involved in implementing and maintaining an FMU export is high.

Overall, tools from a wide variety of areas and industries can be used for FMU creation. Due to the FMI standard, no specific VC software is required to create behavioral models for use in the digital twin. In general, it should be noted that some tools allow the export of FMUs, but do not export a pure C code binary file. Here, a wrapper or a shell is placed around the actual model (cf. Python-FMU). These FMUs often require an installed runtime in the background since they are not based on a compiled C file.

The challenge for the tool manufacturer is to find a suitable tool. On one hand, the requirements for FMUs in the context of the VC must be met. On the other hand, the tool must also fit within the company context and offer as many synergies as possible. In the best case, existing models or the firmware of the real component can be integrated into the creation process. Tool selection is a complex process that requires preliminary analysis and testing. The selection should be tailored to the company-specific requirements. Under certain circumstances, a combination of several creation tools for different components (product families) can also be considered.

This guide distinguishes between FMU creation tools: simulation and modeling tools (1), FMU templates (2), and VC tools (3). Depending on company-specific circumstances and the available know-how, a suitable tool can be found.

7.3 Documentation of an FMU

An important component of a behavioral model is the documentation (cf. Chapter 5.6). General information such as the standard used (e.g. FMI 2.0/3.0), author, creation date, status, versioning, or the generation tool used must be listed here. Since an FMU is a black box, this document (except for the modelDescription.xml) is the only way to provide the user with information about the model. Among other things, the inputs, outputs, and parameters as interfaces of the FMU together with their signal names, data types, default values, maximum/minimum values, units and comments should be mentioned. The signal description must clearly state which function is assigned to the respective signal and how it is to be used. The inputs and outputs can be displayed in the form of a block diagram (cf. Figure 24).

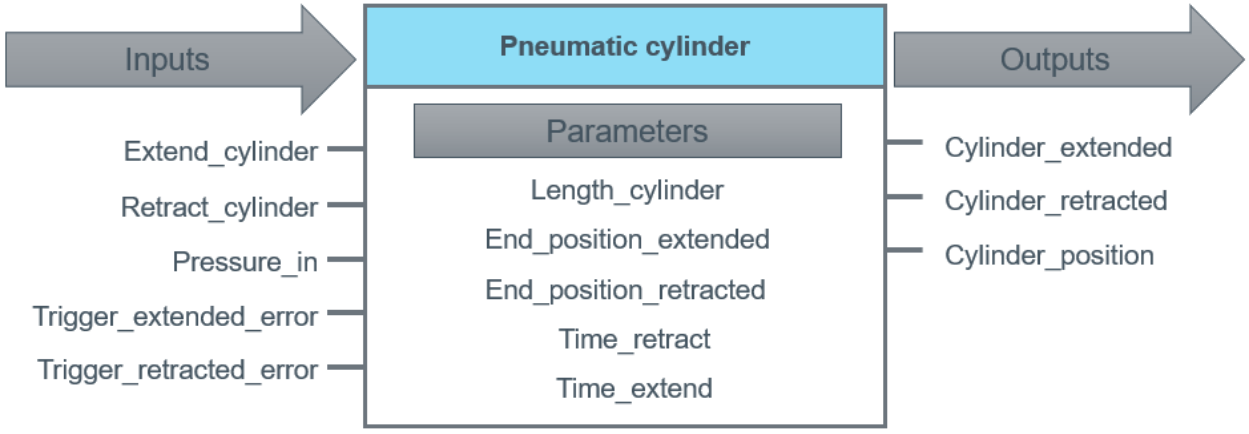

Figure 24: Principal structure of a block diagram using the example of a pneumatic cylinder

The documentation must contain information about which inputs, outputs and parameters must be addressed (basic functions) and which are considered optional (additional functions). Furthermore, it must be described how the data is processed in the black box. This includes, for example, the effects of set parameters on behavior. If error triggers and error cases are mapped in the FMU, they must be described in the documentation. The user must receive detailed information about what is going on in the background during the calculation. Error codes and corresponding log messages must also be described. This creates a basic understanding of the behavior, enabling the user to perform debugging in case of malfunctions and issues.

The default values for parameters and inputs must be selected so that the user can quickly test the model. Basic functionalities should be operational without extensive parameter settings at simulations startup. A small tutorial must be listed in the documentation on how the FMU can be put into operation (e.g. start sequence for state machines, setting of parameters, inputs such as "Enable") and what results can be expected (test functions). In this way, the behavioral model can be tested in the respective co-simulator before the VC.

Another important part of documentation are traces and diagrams (cf. Figure 25 and Table 4). Especially when a temporal behavior (ramps, signal sequences, etc.) is mapped, it is important to document the modeled progression. Based on this reference, the user can identify problems in the co-simulator (possibly performance, utilization, ...). In the VC, this is relevant because monitoring times are often used, so that error messages and malfunctions in the PLC program can affect the VC. In this context, the documentation should also describe the validation procedure. (cf. Chapter 7.4) Statements on model performance (cf. Chapter 9.11) and conformity (according to FMI 2.0/3.0) must be met.

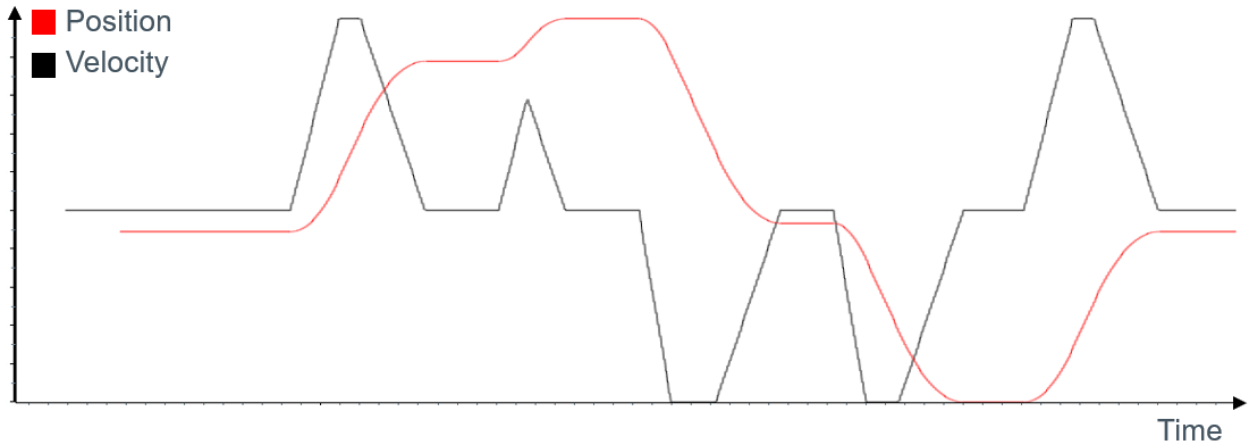

Figure 25: Exemplary scope recording of a positioning drive

Time [s]	Target position [mm]	Enable	Target speed [mm/s]	Target acceleration [mm/s2]
10	100	True	10	10
20	120	True	10	10
30	0	True	10	10
40	-100	True	10	10
50	0	True	10	10

Table 4: Exemplary list of the inputs and parameters of a positioning drive associated with the scope recording

In addition, the documentation must mention which functions, parameters and other scopes of the real component have been implemented in the model and which have not. In this way, the user knows the limits of the simulation model and can adapt the test scope of the VC. This shows the level of detail. This enables a clear distinction from the real component. The user must know what he can and cannot expect from the model. For example, a screw from the hardware store is not installed in a rocket that flies to Mars. If necessary, individual tests can be outsourced to real commissioning (COM).

Since the FMI standard does not offer sufficient possibility to describe the control panel (the GUI for operating the FMU) in a standardized way and to integrate it into the simulation environment, ideally, an example is needed in the documentation. Here it is up to the component manufacturer to sketch them abstractly (cf. Figure 26) or attach a screenshot from any VC tool. For example, error triggers (such as short circuits, cable breaks) can be triggered via buttons from the GUI or various parameters can be set via textual input. The concept listed in the

documentation can be recreated by the user in the respective target tool and serves as an example visualization. In addition to this description in the documentation, images (vector graphics) can also be included in the FMU and used for the GUI in the target tool. For example, images of signal lamps, door switches, etc. must be stored in the resource folder. This eliminates the need for users to search for suitable CAD data or images of the real component. As an alternative to the resource folder, a further standard such as the asset administration shell (cf. Chapter 10) can be used for this purpose. Furthermore, there is the possibility to refer to corresponding Figures via links in the documentation.

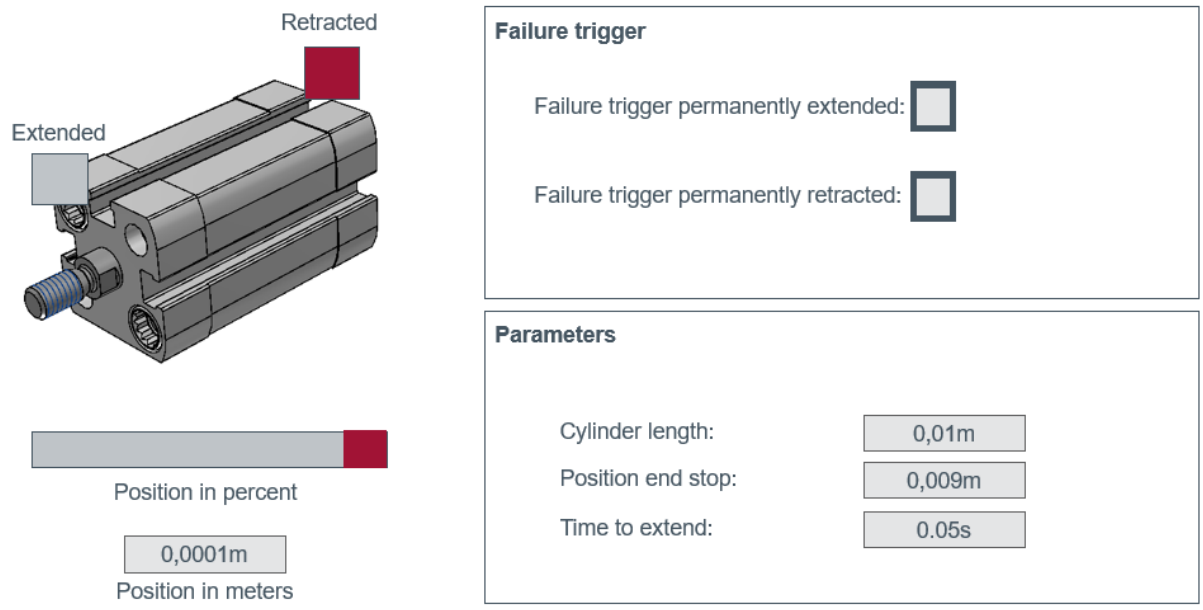

Figure 26: Exemplary representation of a control panel of a pneumatic cylinder including error triggers, end position sensors and parameters

Performance tests must be carried out and recorded in the documentation. It is important to specify the hardware (computer), software (operating system) and the number of integrated FMU instances used, as well as the cycle time achieved. In this context, minimum requirements or target requirements for the hardware and software should also be mentioned. If additional hardware (e.g. network cards for TCP/IP communication) is required, this should also be emphasized in the documentation. This provides information about the required computational resources.

If FMU is licensed (cf. Chapter 7.6), this must be mentioned in the documentation. Details regarding the application and restrictions can be found here.

If images or other documents are included in FMU, they must be named and described as part of the documentation. To ensure that the FMU comes from a trusted source, a signature may be attached (cf. Chapter 9.9). This must be named and described as part of the documentation.
External dependencies (cf. Chapter 5.7) and system requirements must also be listed in the documentation. This includes relevant settings in the co-simulator (e.g. predefined maximum or minimum step size). Adjustments or settings for the connection to the visualization should also be mentioned in the documentation.
In general, it is of enormous importance to provide a contact person or a support address (cf. Chapter 7.9). Since the FMU is a black box, the user must know whom to contact in case of problems. This creates a bridge between the user and the developer. In the VC, problems are sometimes time critical. Additionally, feedback can be given by the user and FMU can be further developed in a targeted manner.
Another component of the FMU manual is the documentation of changes in previous FMU versions. This makes it clear which adjustments have taken place compared to previous versions and whether an update of the model is relevant for the user. For example, changes to signal names must be listed here so that the update process can be coordinated. In the documentation, this can be described via a change history. It is also important to refer to the real component in the versioning and to describe which firmware versions and article numbers are represented by the respective FMU.
It is possible to refer to the documentation of the real component. Individual sections (e.g. parameters, functions, the meaning of individual bit patterns) can possibly be adopted directly. If abbreviations are used, they must also be listed and explained in a list of abbreviations. The documentation must match to the provided FMU.

Documentation is of enormous importance, as FMU itself is an encapsulated black box. A qualitative FMU contains information about inputs/outputs/parameters, metadata, trace records, exemplary control panels and the level of detail depicted, as well as information about logical behavior patterns.

7.4 Validation of the model

After the FMU has been created, intensive tests and targeted validation follow. Due to the encapsulation in a black box, debugging becomes more difficult in case of problems. It is crucial that the maturity level is sufficiently high at the time of delivery in. In general, the more automated the component creation process, the less effort it takes to validate it (*excluded from generation by AI*). As everywhere, manual work leads to errors. If simulation and modelling tools are used to create the FMU (cf. Chapter 7.2), initial tests are possible within this development environment. In the case of e.g. MATLAB/Simulink or OpenModelica, FMUs can be simulated and cross-tested via a "scope" recording. A final test should be carried out in the direct VC environment. This also ensures correct behavior in a "real-time" environment. At the end of the validation, the FMU, especially the signals to and from the PLC, must behave in the same way as the real component.

To validate the logical behavior, the OEM can also provide a standard module (PLC module) against which the FMU is cross-checked (cf. Figure 27). In some cases, the component manufacturer already has its own PLC module for complex components. A simple test project can be used to validate the behavioral model in a "unit test" (at least for a specific use case with a user). If the FMU behaves correct and without error messages in this test project, it can be assumed that there are no problems in the VC (for this use case). Likewise, test cases can be derived and individual aspects can be validated based on the documentation of the real component. The validation must be adapted to the implemented scope and to the specific use case of FMU.

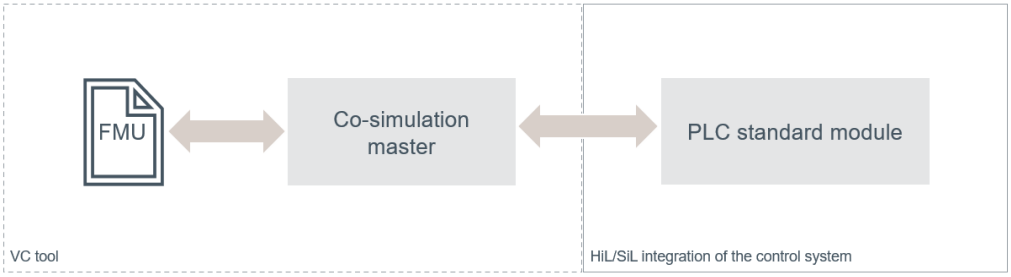

Figure 27: Basic structure of the setup for validating an FMU using a PLC standard module

Detailed physical behavior is currently rarely implemented in the context of VC in the automotive industry. Temporal sequences, such as ramps, can be validated based on the ideal behavior (from the manual). Due to an incomplete representation of the physics (e.g. friction, temperature, aging, ...) in the visualization, validation against the real component is only partially meaningful here. In this context, there will always be deviations currently. In the case of complex physical models, the functionality of FMU should be checked for the jump in the inputs described in the sections 5.4.2 and 6.4. Even if there are large differences in the inputs between two simulation cycles, the FMU must run stably and must not crash.

In general, it is essential to comply with and implement the standard without any deviations. For example, the FMI header files provided by the FMI organization must be integrated into the export tool and the co-simulator. No adjustments should be made here. In any case, the modelDescription.xml must match the provided binary file. If changes are made, both files must be synchronized.

The FMI Association also offers a so-called "FMU Checker" (https://fmu-check.herokuapp.com/), which can be used to check the modelDescription.xml of FMU for conformity. Further validation and verification options are available via the following links:

- https://fmi-standard.org/validation/
- https://github.com/modelica/reference-fmus
- https://github.com/INTO-CPS-Association/FMI-VDM-Model
- https://github.com/modelica-tools/FMUComplianceChecker/releases/tag/2.0.4

The "FMU Checker" reveals errors in the implementation of XML. This syntax test is an essential part of validation. If the FMU throws an error in the checker, the modelDescription.xml was not created correctly. In the best case, the XML should be checked for conformity both by the component manufacturer and in the co-simulator (before the simulation). In some creation tools, the check by the "FMU Checker" takes place automatically when the FMU is built. In general, the component manufacturer should aim not to release faulty FMUs for productive operation. In the co-simulator, it is possible to output a short info text when importing the FMU. Here, for example, the information read from the modelDescription.xml can be shown. Incomplete information, assumptions made by the co-simulator (e.g. DefaultExperiment) or errors in the XML can be logged and displayed. If the FMU is not compatible with the co-simulator, a clear warning message is required. The aim is to comply with the standard in all use cases and to enable smooth simulation in the co-simulator.

A systematic and comprehensive validation of the FMUs is required. Misbehavior and problems in the model can affect the VC and cause delays.

7.5 Versioning, handling and updates of FMUs

The exported FMU is only a snapshot of the source code at a specific point in time. In order to create an overview between different model versions, the FMI standard also offers the option of versioning. The corresponding version number is stored in the modelDescription.xml for this purpose. Redundant information should be avoided so that the version should not be placed in the FMU naming. Appropriate and well-maintained versioning makes it possible to update behavioral models in VC projects in a targeted manner. It is advisable to use data management tools (such as Git, SVN) when creating FMU, analogous to software development. A link between the FMU version number and the firmware status of the real component must be established. This can be done, for example, via the documentation (cf. Chapter 7.3). If the function of the real component changes, the version number of the FMU must also be incremented. If the FMU is not derived directly from the firmware, it makes little sense to take over the versioning of firmware development for FMU creation directly.
Analogous to software development, the principle of "major", "minor" and "patch" can be applied (cf. Figure 28) (cf. Semantic Versioning 2.0.0) [13]. The first validated version starts with the version number "1.0.0". The major version is incremented as soon as major changes are made that do not allow compatibility with previous models. For example, major adjustments have been made to the structure of the FMU. The FMU is therefore no longer interchangeable one-to-one, as there have been changes to the interfaces (inputs, outputs, parameters). An update of the behavior model should not be done during the ongoing VC project, unless the changes are urgently needed. The previous major version is no longer maintained. If new or extended functions are integrated, the minor version must be incremented. The new FMU version is compatible one-to-one with the previous version.

Switching is not trivial and should only be made if the project requires it. A bug fix or an adjustment as well as optimization in the logic leads to a change in the patch version. The patch version may also match FMU's revision number. An update of the FMU in the event of a patch change in the ongoing project is recommended.

Figure 28: Composition of a version number according to the standard Semantic Versioning

If necessary, the Figure 28 can be extended during the development process. For example, it is possible to use a "dev" to inform the user that this is an early stage of development with exemplary subfunctions. "beta" signals a completed development. However, validation based on a PLC project and use in a pilot project is still pending.
An important question is how the user knows that a new version of the behavioral model exists. This can be pushed directly by the component manufacturer as soon as he has made an improvement in the model. Alternatively, the user must actively search for a new version of the behavioral model, e.g. in the event of problems.
A new model version requires an update and possibly integration into an existing VC project. Logical changes in the FMU (e.g. bug fixes) can be easily maintained, as only the binary file has to be reloaded. Signal changes (e.g. naming, new signals) usually require manual adjustments by the user. Such changes must be avoided as far as possible. In general, extreme caution is advised when updating FMU. Replacing FMUs or parts thereof (e.g. DLL, modelDescription.xml) can lead to serious errors. To make the update process as pleasant as possible for the user, the co-simulator or the assistant can compare the modelDescription.xml of both FMU versions. This makes it clear whether signals, data types, or signal names have changed. It should be noted that all instances of an FMU are updated. Errors can occur in mixed operation of two different versions. An automated update process by the co-simulator should be sought. Before the models are updated (e.g. automatically via a global importer), a note must be given to the user and wait for its confirmation. In the case of a manual new import by the user, it can happen that all linked signals and parameters must be redefined.
The co-simulator can read the version of the FMU from the modelDescription.xml and store it in its metadata. In addition, it is possible for the co-simulator to change the GUID (cf. Chapter 5.4.3) must also be examined. This usually changes with every build of the FMU and thus signals changes in the model, even if the version has not

been extrapolated. Adjustments in the interface signals must be recognized urgently, otherwise the FMU may crash.

A well-maintained, structured versioning system supports both the creator and the user of the FMU. This greatly simplifies troubleshooting, the model update process and continuous model optimization.

7.6 Licensing of FMUs

When creating an FMU, internal licenses, licenses of templates and libraries as well as image and usage rights must be stored. In general, it is recommended to store a license for any type of software. This offers legal protection and guarantees control over the use of the software. Even if the FMU is created for free use, this should be documented in the form of a corresponding license. When creating FMUs, third-party libraries or open-source software are sometimes used. The corresponding licenses must be made available accordingly in the FMU. For this purpose, a subfolder "licenses" must be created in the "documentation" folder in the FMU structure, which serves as a storage location for such licenses. The FMI standard itself (e.g. C headers, XML schema files) has been openly released under the CC-BY-SA 4.0 license and is available via a 2-clause BSD [11].

The FMI standard provides the ability to license behavioral models. However, it should be noted that many component manufacturers are used in plant engineering, so that a confusing overall situation quickly arises with different licensing mechanisms. The licensing methods are different. They range from dongles and license keys to servers, clouds and contracts (cf. Figure 29). These different licensing mechanisms must be set up, integrated and, under certain circumstances, moved. In addition, many VC setups are encapsulated by the company network and form an isolated system. The effort for maintaining and renewing the licenses is also enormous. The plant operator (OEM) must operate the models (digital twin) of different plant manufacturers, so that the OEM has to cover even greater diversity. In addition, even old models must continue to run after several years. In general, the individual licensing of FMUs in the context of the VC is difficult to implement and is often not tolerated by the user. Licensing issues during the VC would bring the whole process to a halt. Thus, in the case of licensing, a stable, reliable and simple solution must be found. Frequent license checks can lead to performance issues. It is also important to note that FMU must also be operated offline (without connection to the company network or Internet). A type of licensing could also include an expiring certificate (cf. Chapter 9.8).

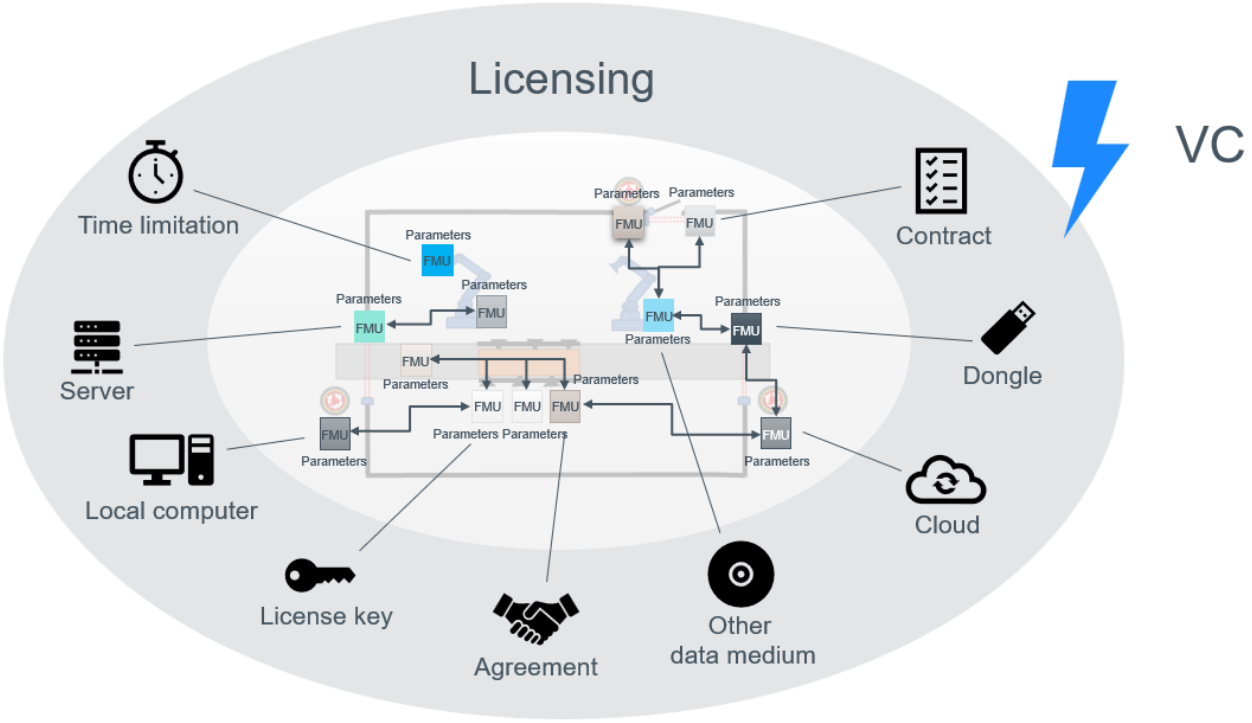

Figure 29: Complexity due to the many possibilities of licensing FMUs in the context of the VC

The delivery of time-limited models is considered very critical. In many cases, such "licensing" leads to the fact that the provided model is not used for productive operation (VC). The risk is high that the FMU license phases out during the VC and that the PLC and robot programmers will not be able to use the simulation model. In some cases, the VC lasts several months to even years. A time limit for FMUs is only recommended for beta versions.
It must be clearly regulated between the component manufacturer and the user for which purposes the FMU may be used and to what extent the FMU may be passed on to customers or suppliers. In the moment that the digital twins will passed on to the plant operator (OEM), licenses and rights must also be transferred across companies. If there are any restrictions in this context, this must be clearly defined at the time of handover or in the corresponding licensing method. It must also be clarified to what extent maintenance and optimization measures are to be expected.
Currently, behavioral models are licensed via the libraries in the respective VC tool. Exchange platforms for component models (such as CADENAS, TwinStore, ...) are also managed via specific accesses. In the case of FMUs, the co-simulators can also be licensed by VC-Tools.

This section is intended to raise awareness of the challenges associated with licensing. Especially for the introduction of the FMI standard, it is important not to put any additional hurdles in the way due to complex licensing mechanisms.

The FMI standard offers the possibility of licensing. In general, complex licensing mechanisms should be avoided, as these can have a negative impact on the VC in terms of performance and administrative effort.

7.7 Model Usage Traceability

Regarding export restrictions or plagiarism protection, it would be desirable for the model creator to know where his FMU applies. One possibility is the individualization of FMU itself. For example, a separate model could be created for each real component based on the serial number, which only works by entering the corresponding serial number. This procedure is equivalent to licensing via license keys. However, since the user usually does not have a real component in an early phase, this procedure is difficult. Often not even the exact article number is determined.

Alternatively, the FMU creator can be informed via the Internet where and how many of his FMUs are in use. However, this contradicts the requirement that the FMU must not have a connection to the network if this is not provided for in the real component (cf. Chapter 6.3). The VC setup is often set up separately from the Internet (cf. Chapter 9.9).

In general, appropriate licensing helps to ensure traceability and model usage. Often, however, it is only possible to see whether the user is using the model. The information about the number of operated instances cannot be provided with many licensing mechanisms.

In general, such traceability procedures require increased effort on the part of the component manufacturer, but also on the part of the operator. The effort must be kept as low as possible here.

CAD files (e.g. step) currently do not allow traceability. Since FMU forms an encapsulated black box, the dissemination of data is not critical. Rather, a kind of license agreement should be concluded between the component manufacturer and the user in this context. In this context, it must be clearly defined which restrictions apply to the transfer to third parties or the publication of the model (keyword: export restrictions). It should also be mentioned that FMU may not be used for reverse engineering. Contracts, general terms and conditions, NDAs and FMU regulations can be used to determine in which areas of a company (development, production, ...) FMU may be used.

The traceability of model use is technically difficult to implement. For this reason, agreements and contracts must be concluded at the time of model handover.

7.8 Delivery of the behavioral models

FMU can be deployed in different ways. This is about how the component manufacturer makes the models and associated data available to the user (plant manufacturer or plant operator). In general, it is not expected that the source code will be delivered unless the FMU development has been especially commissioned by the user. It can be assumed that the FMU has already been validated at the time of delivery. Since the topic of FMU provision is still in its early stages, the market situation is still unclear. A decision on a format or a generalized platform has not yet been made. In addition to the requirements for the automotive industry, the component manufacturer also must serve other industries and areas. One possibility is the asset administration shell (cf. Chapter 10.2). In this case, the FMU can be provided as AAS type 2 via a server.

For the user (Plant Manufacturer or Plant Operator) a component-related exchange analogous to CAD data (possibly via comparable platforms, such as CADENAS) would be desirable, as this would allow the data to be provided and managed in a bundled manner. If the databases are external, it is important that they can be connected via an interface (API). Many OEMs have their own platforms through which component manufacturers already provide CAD data and manuals. This can also be used to manage and document the delivery of FMUs. The exchange via a platform ensures availability and offers security against cyber threats (cf. Chapter 9.9) as the data is delivered through a specific and secure channel. Alternatively, the FMUs can also be offered for download via the homepage of the component manufacturer.

The selection for a generalized exchange platform has not yet been made. Delivery of FMUs via the asset administration shell (AAS) is technically feasible and will be pursued in the future.

7.9 Liability for provided models

A distinction must be made here between damage caused by malware (viruses, etc.) and misconduct in the model. Damage caused by malware is intercepted by mechanisms like certification (cf. Chapter 9.9). Misbehavior in the model can lead to delays in the VC (costs due to late delivery dates) or, for example, to collisions on the real machine. Personal injury during COM can also result in a shortage. In this context, the question arises as to who assumes liability here.

Ideally, the validated model behaves analogously to the real component, so that no misbehavior occurs. In general, a distinction must be made between whether the component manufacturer offers FMU for free use or

creates it on behalf of the user. In addition, it also plays a role whether the misbehavior is caused by the implementation or by incorrect parameterization.

In general, good documentation of FMU is important (cf. Chapter 7.3). This can significantly speed up troubleshooting and uncover user errors. It is recommended that the user (plant manufacturer/OEM) tests the FMU after receipt based on a test project or via an EA test before use in the VC and validates it accordingly for his application.

In addition, it is essential that a contact person is provided by the component manufacturer when the FMU is delivered. This can be done via the documentation (cf. Chapter 7.3) or the modelDescription.xml. The contact person must be available in case of problems and provide a quick remedy. Since FMU is a black box, the user cannot make any adjustments to the behavior model. First and foremost, the model creator (component manufacturer, platform operator, plant manufacturer or service provider) is responsible for FMU's conduct. Currently, the provision of FMUs is an offer from the component manufacturer. The models are modeled and validated to the best of knowledge and belief and do not claim to be complete and error-free functionality. In addition, the creation of models in the FMI standard is new for many companies. Initial experience must be gained. When providing the behavioral model, the FMU creator can also protect himself with regard to liability by means of standardized formulations (in the FMU documentation or in the modelDescription.xml) and contracts. Many companies already have standardized terms and conditions for software liability. These can also be used for FMUs and can be adapted if necessary. Such disclaimers also exist when exchanging CAD data. The topic of liability cannot be generalized and must therefore be considered on a case-by-case basis.

Conclusion:

In general, the VC offers an opportunity to detect errors and dangers at an early stage. The risk of going to the COM without a VC is currently significantly higher, so that no liability claims are made. Currently, in most cases, only the documentation of the real component and the PLC code (building block) are available as the data basis for the model creation. Therefore, no behavioral models are expected that correspond 100% to the real component. The downstream COM represents an additional validation of the system, which reveals possible shortcomings in the model and represents a further safeguard. The use of FMUs and the involvement of the component manufacturer increases the model quality, so that a general improvement of the current situation is achieved. The issue of liability for provided models is currently seen as uncritical. This applies equally to provided CAD models. Currently, behavior libraries are released by the OEM and used for the VC. In this process, there is no difference whether FMUs or native library elements are released. For the release of the behavioral models, they are tested and validated by the OEM. Through this process, responsibility is clearly to be defined.

The topic is to be considered analogous to the current situation. Before application, the models must be approved by the OEM. A clause to exclude liability provides additional protection for the component manufacturer.

7.10 Benefits of FMUs for component manufacturers

The main users of FMUs are plant manufacturers or OEMs. However, the greatest added value of standardized behavioral models is achieved when the component manufacturer itself benefits from the FMUs it creates. It is conceivable that in the future it will integrate the virtual representation (Software in the Loop (SiL)) into the development process instead of the real component (Hardware in the Loop (HiL)). The FMU is already available before the real component. In this way, the engineering process at the component manufacturer can be shortened, as initial safeguards are already possible at a very early stage. In this way, FMUs also support internal development and system testing. Concrete examples of this are integration tests into certain control systems (e.g. technology objects in the TIA Portal), the development of standard modules (PLC modules) or the connection to parameterization tools. The component manufacturer can also create more detailed models for their own applications, which are precisely tailored to the respective use case. In addition, the FMI standard offers the possibility of coupling different tools with each other and thus improving data consistency in the development process of the component.
Component manufacturers often also have their own special machine construction or develop test benches for their components. In this context, the behavioral models (FMUs) can also be used for internal VC activities.

In order to maximize the added value of behavioral models, FMUs should also be used by the component manufacturer (e.g. in the development process).

8 Integration into the VC tool landscape

The VC setup at the plant manufacturer/OEM is made up of several software tools that take over different areas of responsibility. There is talk of a so-called tool landscape. This consists of emulation tools of the controllers (e.g. PLCSIM Advanced, TwinCAT, ...), an emulation of the robot controller (e.g. KUKA OfficeLite, ABB RobotStudio, Fanuc ROBOGUIDE), a tool for behavior modeling (e.g. SIMIT, WinMOD, ViPer) and a visualization (e.g. MCD, Process Simulate, SIMLINE, YAMS). In some cases, visualization and behavioral modeling are combined in one tool.

8.1 Integration with the Behavior Modeling Tool

In principle, the FMU as a neutral format can be adapted by a co-simulator to each of the tools, listed in chapter 8. Since FMUs are mainly behavioral models, integration into the behavior modeling tool (e.g. SIMIT, ViPer) has proven to be useful (cf. Figure 30). In this way, it is also possible to easily adapt existing assistants and automatisms for model generation to use with FMUs. This is done with a so-called "wrapper", which adapts the FMU to a specific scheme and supports the integration of the FMU into the native tool library. The FMU forms a function block for a block diagram with inputs, outputs and parameters. With clever integration of FMUs, the user does not notice any difference to native library elements. This simplifies the integration of FMUs. The FMU runtime is usually directly linked to the VC tool, so that the FMUs are started and stopped with the behavior simulation. These tools often offer the possibility of recording. Inputs, outputs and internal states are stored, which allows certain fault cases to be simulated and analyzed. In this context, FMUs must also support this functionality (cf. Chapter 5.4.2 canGetAndSetFMUState und canDe/SerializeFMUState).

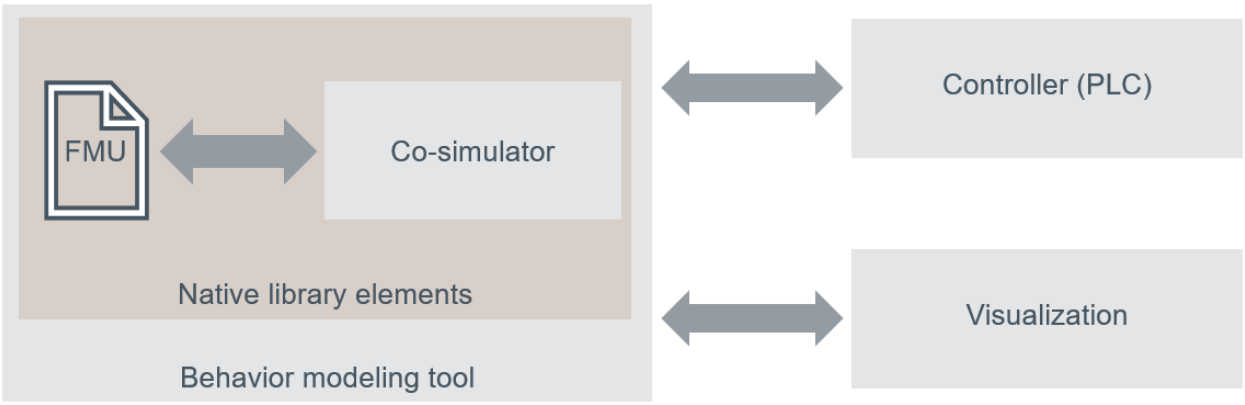

Figure 30: Principal connection of FMUs via the tool for behavior modeling

This type of setup allows hybrid operation with native behavior models and FMUs. Existing libraries can still be used. If an FMU is offered by the component manufacturer, it must be integrated in a targeted manner. In this way, the libraries are gradually enriched with FMUs (cf. Figure 31). In case of FMUs which do not meet the user's requirements, they can be optimized to a certain extent in the respective VC tool. Examples include splitting a word into individual bits or the additional normalization of signal values. Such hybrid libraries will be available in many use cases in the automotive industry in the short to medium term.

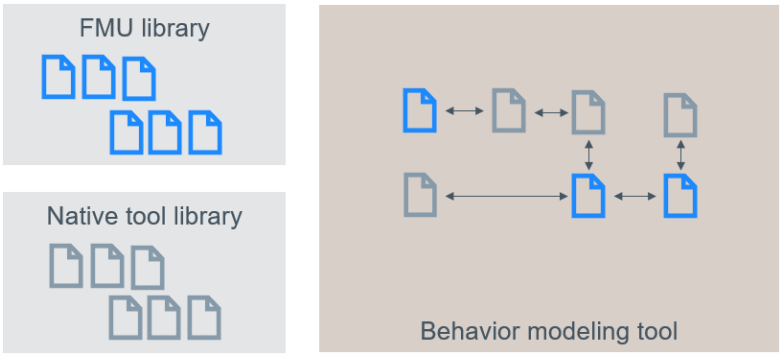

Figure 31: Simplified representation of a hybrid operation of FMUs and native library elements in the behavior modeling tool

8.2 Integration through an external co-simulation master

In addition to integration of FMUs into the behavior modeling tool, it is also possible to outsource the FMUs and operate them in an external co-simulation master (e.g. WinMOD, iSILOG PLCConnect) (cf. Figure 32). The connection to the respective VC tool is realized by an interface (e.g. Shared Memory (SHM), API, TCP/IP). Especially in the case of performance problems in the VC tool, this approach can help to outsource complex models to an external process. In this way, the computing effort can be scaled as required, as it also enables distribution to several systems. In addition, this approach improves system stability.

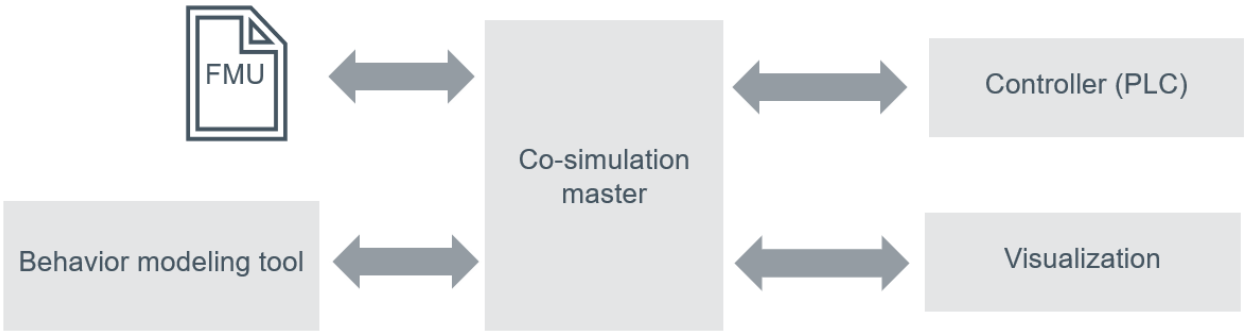

Figure 32: Principal connection of FMUs via an external co-simulation master

In this case, too, hybrid operation of FMUs and native library elements is possible. FMUs can be integrated into the behavior modeling tool via dummy modules and macros and passed on to the respective controllers. Existing modules can still be simulated natively in the respective VC tool.

8.3 Integration with visualization

In principle, it is also possible to integrate FMUs into the visualization tool (usually 3D CAD environment) (cf. Figure 33). In terms of performance, however, these tools are often heavily utilized (especially in physics-based simulation), so that the integration of FMUs sometimes causes problems here. The visualization tools usually offer only rudimentary possibilities for behavioral modeling and are therefore clearly different from the tools described in the section 8.1.

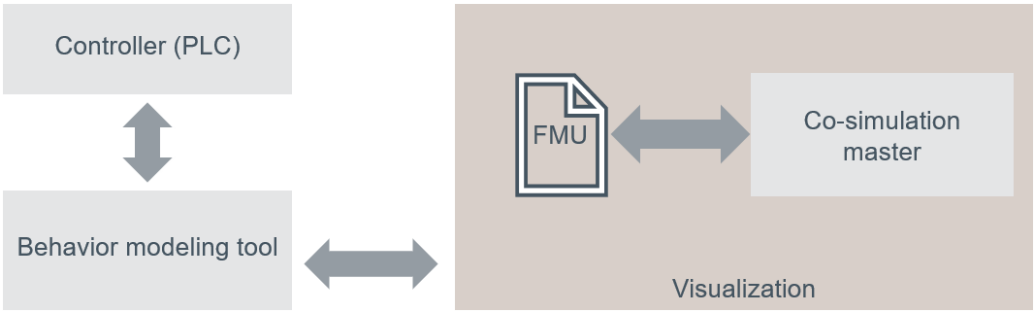

Figure 33: Principal connection of FMUs via visualization

Furthermore, there are VC tools that combine visualization and behavioral modeling. These tools can integrate FMUs directly and connect them analogously to their native behavior models.

8.4 Summary

Since the behavior models can be integrated into the simulation setup as FMUs at several points, it is important to find a suitable, optimal solution for the respective application. A blanket determination of how the FMUs are to be distributed does not make sense. In all cases, hybrid operation of FMUs and native library elements must be enabled. A 100% use of FMUs is difficult to implement in the short and medium term.
Currently, a large number of VC tools already offer an FMU interface. However, the integration of FMUs is still done manually in some cases without automated instantiation. However, the first concepts have already been validated, so that further optimizations will be made as FMUs are used. In the automotive industry, the share of FMUs is currently <5%. In special machine construction, projects with >95% FMUs have already been successfully carried out.

There are various ways in which FMUs can be integrated into the VC tool landscape. A suitable solution must be found in the respective application.

9 Connection to the simulation model

In order to enable a seamless integration of FMUs into the VC tool, various aspects need to be considered. These must be taken into account on both the creator and user side. Chapter 9 lists various topics and supports the connection of FMUs to the digital twin.

9.1 Implementation of the standard (depth of implementation)

To enable a smooth use of the FMI standard in the VC environment, it is essential that the VC tools use an implementation of the FMU co-simulator in a sufficient depth of implementation. The FMI standard is extensive. Not all functionalities and options are relevant in the context of the VC. Chapter 5 lists the required scope of the standard. Various flags are described here and relevant functions are mentioned. In principle, the co-simulator cannot be implemented based on specific examples. Rather, it requires the most comprehensive integration of the FMI standard. In the event that certain functionalities and methods are not implemented, it is important to evaluate the corresponding flags and give the user clear feedback. At this point, it is worth mentioning that the FMU co-simulator may behave differently in a different system environment. In the Windows context, tests on Windows Desktop (Windows 10/11) and Windows Server (if supported) are necessary.

In addition to FMUs from various creation tools, other use cases need to be analyzed. For example, models with a large number of inputs and outputs (>500) are to be tested. Here you can see whether the VC tool (co-simulator) can also handle it visually. Many internal status variables can also cause problems in the co-simulator. In addition, there is the possibility that large FMUs exceed the memory capacity of the solver and cause it to crash. In the co-simulator, it is necessary to investigate how many instances of the same model can be loaded (multiple instantiation). It is also necessary to analyze whether the co-simulator can handle different versions of the standard (hybrid operation) and FMUs from different creation tools. In this context, scaling tests with several hundred FMUs must also be carried out.

The FMUs generated by various tools must be simulated without any problems. For example, it is not sufficient to validate your co-simulator solely on the basis of the standard test FMUs (e.g. BouncingBall). It is recommended that the manufacturers of templates and export tools provide sample FMUs against which the co-simulator can be tested. In this way, misinterpretations of the standard and shortcomings are detected at an early stage and the co-simulator is optimized in a targeted manner.

In the current situation, FMUs must always be tested for functionality in the corresponding co-simulator before they are used. In the future, this preliminary test will be omitted by the user as far as possible. As soon as the FMU has been approved by the component manufacturer, it must later also be compatible with the common VC tools. The challenge here is mainly on the side of the co-simulator, which has to fully implement various functions, data types or flags.

In general, it should also be considered on the creator side that not all co-simulators have implemented the full functionality. The FMU is to be developed in such a way that extended functionalities are not absolutely necessary for the simulation of the FMU and are to be regarded as optional. The advanced properties can be used if the co-simulator supports them. The core functionality must be ensured. An example is the flag "canGetAndSetFMUState" (cf. Chapter 5.4.2). Here, the FMU must also be able to run if the co-simulator does not enable this function. Another example is unsupported data types, which are mapped to those available in the VC tool. In the co-simulator, the user must be provided with a clear information as to which functionalities cannot be used and where compromises in the simulation are to be expected.

The co-simulator must implement all areas of the FMI standard relevant to the VC. A pure validation based on the standard example FMUs is not sufficient.

9.2 Naming concept of inputs/outputs and parameters

The signal names must be named in English. Special characters (e.g. @, ", &, |, %) lead to problems in many tools and should therefore be avoided. Even though umlauts are now supported by many VC tools, it makes sense to do without them in signal naming. In general, the signals should be named in such a way that they are understandable to the user. A short, meaningful name is to be strived for. The signal naming of the FMU is up to the component manufacturer. It is recommended to stick to the real component when naming and to adopt the designations. Even if the FMI standard allows it, care must be taken to ensure that the same signal name is not used for an input, output or parameter. A uniform naming scheme such as Camel Case, Pascal Case or Snake Case must be observed. This makes the signal names easier to read.

Currently, OEMs in the automotive industry use their own standard for signal naming, which simplifies the mapping of the various signals. In general, it does not make sense to use the naming standard of individual companies (OEMs) for standardized behavioral models. A global standard does not currently exist. Components can often be connected to different bus systems. It does not make sense to link the designation to a specific communication (e.g. PROFINET). Even if this makes integration more difficult on the user side, the naming must be carried out in a generalized manner as far as possible. The effort of mapping falls on the plant manufacturer (OEM). The assignment to the standardized signal naming (OEM) may take place in the respective VC tool or automatically in the form of assistance via a wrapper.

With a suitable signal naming, the effort for this wrapper is reduced. A uniform concept simplifies the interpretability of different signals. For example, the signal name can indicate whether it is a PLC signal or whether the communication is directed towards the visualization. It is not relevant to store information about the data types in the signal naming, as these are already mentioned in the modelDescription.xml. This only results in

redundancies, sources of error and unnecessarily complex signal names. A reference to the signal direction is also irrelevant (e.g. "from", "to"), since the modelDescription.xml clearly describes whether it is an input or output. In this context, purely redundant information is stored.

About the enrichment of information in the signal name, there is no uniform standard so far. Within the framework of the DIAMOND research project, the following convention was adopted as a guide (cf. Figure 34):

- **PLC_**: This prefix is used to name signals that go from or to the controller. These signals also exist in the real component and form the interface to the PLC. The data types here are to be selected analogously to the real component.
- **VISU_**: This prefix signals that the signal is going from or to the visualization. The signals do not exist in the real component. (e.g. trigger signal of a sensor from the visualization, position value from a drive to the visualization) The visualization can also correspond to a 2D simulation (e.g. label simulation).
- **TRIG_**: This signals that the signal is an additional signal from the component model, which does not exist in reality. The signals are usually addressed via a control panel in the VC tool. These include, for example, error triggers or other signals for worker interaction (e.g. control buttons, door locks, authorizations).
- **PARA_**: This summarizes all parameters that are used to adapt or parameterize the behavior model to a specific component. For example, cylinder length, operating mode, times, positions, speeds or accelerations should be mentioned.
- **PROC_**: This abbreviation can be used to describe process information. Possible aspects are the position, the internal condition, the force, speed or the compressed air applied. This term also includes signals that are used to exchange information with other behavioral models. (e.g. air, electricity, fluid).
- **ROBOT_**: This prefix can be used to identify a signal that is connected to the robot controller. Examples of this are welding guns or robot grippers.

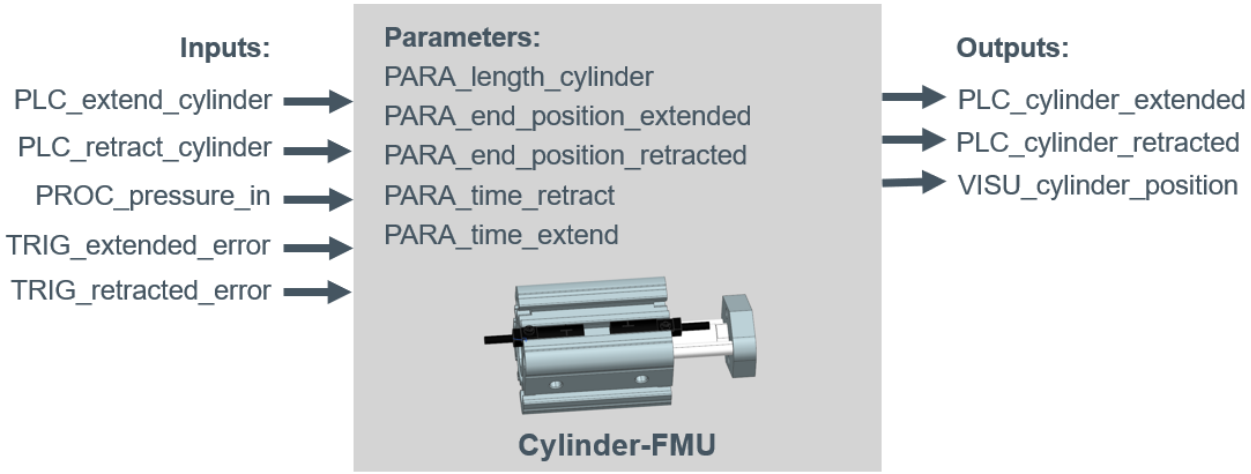

Figure 34: Designation with prefix to simplify the connection of FMUs using the example of a pneumatic cylinder

In case of difficulties with the signal names and the need for adjustments for the wrapper, the SSP standard (cf. Chapter 10.1) can be used by placing a bowl around the FMU. SSP variables can be used to freely rename the signal names on the user side.

The component designations from the plant manufacturer (e.g. BMKs) are not part of the signal or FMU names and are assigned when the FMUs are instantiated. The assignment of the FMU name is up to the creator. However, it makes sense to integrate the company name of the component manufacturer into the naming. Furthermore, it is recommended to list the article number of the specific component. The FMU name is listed in this section because, depending on the type of instantiation, it can be part of the signal name. If several instances of an FMU are simulated in the co-simulator, it is common practice to enrich the FMU signal names with the respective instance names. This makes the signals clear. When instantiating, the naming of the FMUs should be based on the BMK designations from EPLAN. This allows signals to be automatically connected to each other later.

The naming of the signals is up to the FMU creator. Prefixes can help the user connect to the simulation model.

9.3 Default values

The default values can be found in the modelDescription.xml. This is particularly important for parameters and, if necessary, inputs. It is also important that the user can overwrite these values in the simulation. This enables parameterization via the co-simulator/VC tool. The default values should be selected in such a way that the FMU is as executable as possible immediately after the simulation starts. (cf. Chapter 7.3).

The default values are to be transferred from the modelDescription.xml to the co-simulator.

9.4 Requirements for the interfaces (data types)

Depending on the version of the standard (FMI 2.0/3.0), different data types are defined (cf. Table 5). The FMI Standard 2.0 supports only commonly used types such as Boolean, Integer, Real, and String. If various integer formats (8/16/32-bit, signed or unsigned) are required, the newer version 3.0 should be used. Additionally, array support is only available starting with FMI version 3.0. In many cases, arrays help to make FMU more structured and simplify the connection to the simulation model and the controller. Especially for components such as cameras, printers, or RFID readers, long arrays often need to be passed to the controller [27]. In the control domain (PLC and robot controller), it is important to adapt to the data types stored in the PLC program. These data types should to match the real component. Otherwise, problems and possibly sources of error may arise.

Data type	modelDescription.xml	FMI 2.0	FMI 3.0
Float	single		✓
Double	double	✓	✓
Int8	byte		✓
UInt8	unsignedByte		✓
Int16	short		✓
UInt16	unsignedShort		✓
Int32	int	✓	✓
UInt32	unsignedInt		✓
Int64	long		✓
UInt64	unsignedLong		✓
Bool	boolean	✓	✓
Char	string	✓	✓

Table 5: Overview of the supported data types in the FMI standard 2.0 and 3.0 [11]

In many cases, the signals of an FMU are made up of different data types. Depending on the connected controller, a byte flip may be necessary for integer values to ensure correct interpretation. The component manufacturer has to choose an appropriate data type for each signal. In view of the increasing complexity in communication, arrays

will become increasingly important in the future. Custom-defined structures and data types, as well as dynamic strings commonly used in IT applications, are not required in the context of the VC. Currently, the challenge is that VC tools often support a limited set of data types and must map FMI 3.0 data types accordingly. Mapping to a different data type often leads to errors and loss of information. Interfaces to PLC software and other VIBN tools are also increasingly demanding additional data types, meaning that extensions independent of the FMI standard must be implemented in various tools and controllers. In some cases, this has already been done. When connecting to visualization tools, it is recommended to use commonly supported data types. There is no standardization in this area. The data types defined in FMI version 2.0 are fully supported in the common VC tools (with the exception of strings).

The data type String (limited to a certain length) is currently causing some problems in the VC tool landscape, as not all tools support it. In addition, strings are rarely used in communication with the PLC. For this reason, ASCII encoding is often used, which allows characters to be exchanged as integers/bytes. Strings should only be used in the FMU if required by the PLC program and if they are present in the real component accordingly. In the future, more and more VC tools will support the String data type. For users, strings are easier and faster to interpret. Complex data can be easily exchanged. Especially for parameters (e.g. IP addresses, serial numbers, paths, configurations) the data type String is important.

With regard to the overall system performance (digital twin), it is desirable to keep the number of signals (inputs, outputs) as low as possible. The parameters can be set before the simulation starts and therefore do not necessarily affect the communication during the simulation. In general, the interfaces are designed to handle large volumes of data (>1000 signals). Thus, a suitable balance must be found, aiming for a reasonable number of signals: essential signals must be exposed, while optional information (e.g., internal states) can be made available via FMI logging. Words and integer values should be passed to the interface as such and not split into individual bits. This also keeps the model clearer. It is beneficial to represent large data packages and structures using arrays and terminals (cf. Chapter 5.9). The guiding principle is: "as little as possible, as much as necessary". The SSP standard (cf. Chapter 10.1) can be used to route signals internally and relieve the interfaces. Regarding this topic, please refer to section 9.10 which addresses performance aspects in more depth.

The co-simulator must support all data types defined in the FMI 3.0 standard. In the future, the integration of arrays will become increasingly relevant.

9.5 Units

Units play a major role in the standardized exchange of data across system boundaries. The transmitted, represented as a number, must be interpreted correctly [30]. Therefore, it is recommended to adhere to standards

for linking the individual components and for connecting to other systems, such as controls or visualization. By using a unit standard, the connection is simplified and the risk of errors is reduced. In this context, the widely used SI standard has proven to be useful. If, in exceptional cases, models deviate from this standard (e.g. because the real component or the PLC block uses different units), this should be clearly highlighted in the documentation and in the signal comments. In case of doubt, the real component serves as the reference. Real parameter units should also be taken over to the FMU accordingly.

In this context, it should be noted that different VC tools use different base units. SI units are not always set as default. However, in most tools, units can be defined in the settings. Another possibility is to switch between different units via a parameter in the FMU. This functionality simplifies the connection of the FMU. A common example is pressure, which is often expressed in the unit “bar.” According to the SI standard, however, the unit "Pascal" is set. Other non-SI units that are commonly used are "mm/s" or "l/min".

The AML standard mentioned in Section 10.3 also addresses the standardized definition of units. Since the SI standard does not fully cover the areas of transport, packaging, sales and information technology, AML refers to UNECE Recommendation N°20 "Codes for Units of Measure Used in International Trade" [30]. This serves as an extension to the SI standard and should be used whenever additional requirements for unit definitions arise [28]. Furthermore, the ECLASS standard can be used if Recommendation N°20 does not ECLASS defines various properties via attributes and describes them using predefined units. In doing so, ECLASS uses standard units according to DIN and ECE [9].

The FMI standard provides the option to define units in the modelDescription.xml. The units are composed of base SI units. In addition, a conversion can be specified. This allows the co-simulator to perform a unit check during signal mapping and, if necessary, carry out a conversion. In the context of the VC, such information and functionalities are currently rarely implemented. However, in the medium to long term, this option offers significant potential for error reduction and model optimization [11].

Common SI units are to be used when modeling FMUs and in the interfaces.

9.6 Control panel in the VC tool

As mentioned in Section 7.3, the FMI standard—even in version 3.0—does not support visualization for a graphical user interface (GUI). The control panel, as it is currently modeled in the VC tool, cannot be described in a standardized way. As a result, it will continue to be represented within the behavior modeling tool. In the simplest case, the control panel describes the block diagram of the black box with the inputs, outputs and parameters. This form is currently used in many VC tools when importing an FMU for automatically generated visualization. FMU parameters are either represented as inputs or linked to global variables. This automatically

derived operating concept is not practical for handling in the VC. It requires a connection of buttons and input fields for error triggers, for example. In addition, displays must be created that enable the user to recognize important values immediately. It is the task of assistants and automation mechanisms to link the respective FMUs with the control panels modeled in the VC tool.

The control panel is not part of the FMU and can be implemented in the VC tool. An exemplary structure is to be provided in the FMU documentation.

9.7 Parameterization of the behavior model

With a higher level of detail in the behavioral models, the number of parameters and the associated effort in parameterization increases. For example, while the travel paths of a pneumatic cylinder were previously defined only in the visualization, more detailed FMUs require these to be specified within the behavior model itself. The parameters of an FMU can be roughly divided into two categories. First, there are parameters that adapt a generalized behavior model to a specific use case or component. The broader scope of an FMU (one model for an entire product family), the greater the number of parameters. The second category of parameters is based on the real component and is used to configure settings for the specific application. The parameterization process differs for each of these categories. Ideally, an FMU should allow parameters that correspond to those of the real component to be configured in the same way. The implementation can be carried out, for example, via a web UI or a TCP/IP connection to a parameterization tool, that interacts with the FMU in the same manner as with the real component. The parameters for setting the generalized behavior model must be defined as automatically as possible. Information from databases, internal tools, VC tools, the circuit diagram or CAD data can be used here. For example, the operating mode of a drive or the stroke lengths of a pneumatic cylinder should be mentioned here. These parameters are set via the VC tool (co-simulator) and can be defined via macros like previous behavior models. If the co-simulator is suitably integrated into the VC tool, parameterization can be carried out automatically using existing assistants.

In general, parameters of the type "tunable" should be defined in the context of the VC if possible. These can also be adjusted during runtime. This makes it possible to simulate error scenarios, as FMU provides implausible values when a change is made. "Fixed" parameters can lead to issues, as their values may not be available before the FMU simulation starts. Under certain circumstances, starting sequences must be defined for this purpose.

In relation to parameterization, the SSP standard, which explained in more detail in section 10.1 should be named. Parameters (also "fixed" parameters) can be set here, which override the default values of the FMU during instantiation.

In general, parameters are to be defined as "tunable" and set via the co-simulator. The SSP standard can provide additional support for instantiation here.

9.8 Instantiation

Instantiation turns the behavior model of a component into a specifically defined instance (serial number/BMK). The designation of each instance must be unique (see naming in the circuit diagram). Using appropriate assistants, FMU instantiation can be automated based on stored behavior models. In this context, a corresponding API, interface or import of lists (.txt, .csv, ...) must be implemented in the respective VC tool or co-simulator. With proper integration of the co-simulator into the VIBN tool, instantiation can follow established procedures. In some cases, a mapping table may be relevant to refer to the respective FMUs.

Automated instantiation using the co-simulator/VC tool should be sought.

9.9 Security (Cyber Security)

In general, security risks are currently associated with the VC due to network activities, relaxed Windows and firewall rules and the large number of tools and models. Since FMU is an encapsulated, compiled file, there are certain risks associated with co-simulating and executing a binary file. These must be minimized. It is not transparent what the behavior model (FMU) is doing in the background on the user's system. In the worst case, malware can be installed or executed. Currently, most FMUs are provided directly by the creator, making them a trusted source. However, with extensive use of FMUs, it becomes increasingly burdensome for users to obtain all FMUs directly from the component manufacturer via a secure, analog delivery method. Once FMUs are exchanged over the internet and across various platforms, proof of origin is lost. FMUs from unknown sources may impersonate legitimate ones.

For this reason, signatures/certificates are of great importance. This ensures that the FMU comes directly from the component manufacturer (or service provider) and has not been modified by third parties. The responsibilities and the origin of the models can be clearly defined by a signature. It ensures that the FMU comes from a trusted source and has not been altered afterwards.

There are risks associated with executing FMUs. The FMU must come from a trusted source. (Signing or secure exchange channel)

9.9.1 Basic principle of a signature

The cryptographic signature is designed via private and public keys (cf. Figure 35). A HASH code (e.g. SHA512) is generated via the files to be signed, which is converted into the signature of the FMU using the private key (e.g. RSA signature, padding mode, PKCS). This becomes invalid as soon as adjustments are made in the FMU (e.g. adaptation/replacement of the DLL or modelDescription.xml). The public key can be used to check the authenticity of FMU's signature. This key must be made available to the user. The challenge is to provide these public keys [26]. In addition to a direct exchange, Public Key Infrastructure (PKI) certificates, for example, offer another option. In addition to the key, meta information can also be transmitted via this way [16]. In the final step, a hash is generated from the signature and the public key, which is then compared with the hash of the FMU. This allows verification of whether the FMU is original or has been modified afterwards. It is also possible to create multiple signatures for each file (FMU).

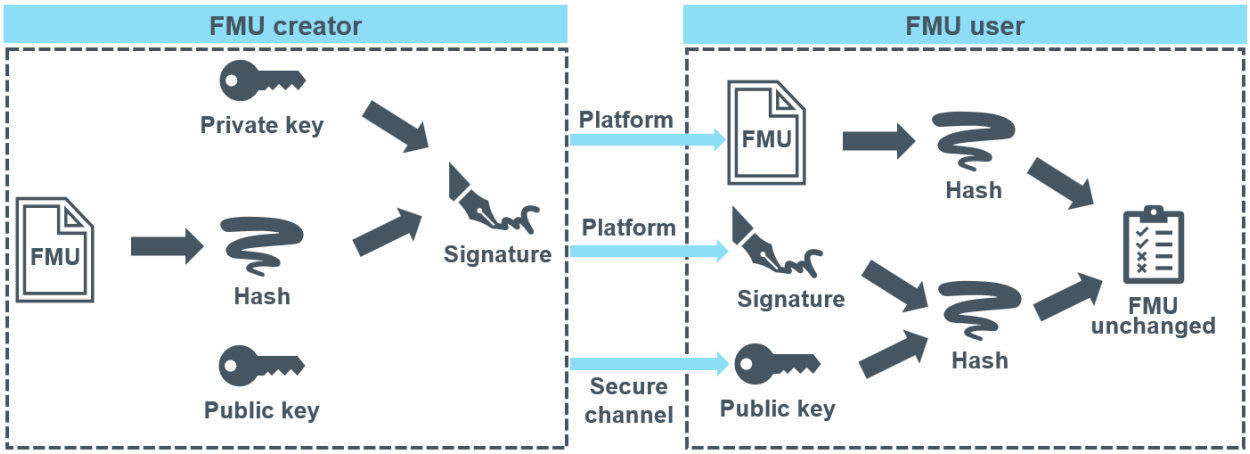

Figure 35: Principal procedure for signing an FMU using private and public keys

In the context of FMUs, common procedures for signing can be used.

9.9.2 Use of signatures in the context of VC

Signing is used to authenticate the FMU provider. In addition, a signature can be assigned by the OEM to release the FMUs for further use (e.g. by a system integrator). This enables secure exchange of entire FMU libraries. By signing, the FMU is set to valid and a clear responsibility for the content is defined.

In this context, the co-simulator (VC tool) can provide a warning when executing or loading the FMU and inform the user of the risk as soon as no suitable certificate (signature) is stored or the signature has been broken. Through this warning, the tool manufacturer informs the user of the potential danger. The latter must decide

whether he wants to take the risk. The responsibility does not lie with the tool, but with the user. Additionally, the tool's terms of use may include a notice about the risks of executing unsigned FMUs from third-party sources. Permitted signatures (from the respective companies) can be stored via so-called "white/trust lists". If the FMU contains several signatures, it is sufficient for the check if one of them has been deposited in the "Trust List". It is recommended that these root certificates are defined by IT during installation, so that the tool user does not have to deal with this issue. These root certificates ("public keys") must be exchanged between the signature creator (component manufacturer) and the user via a secure platform. As soon as a root certificate is offered together with the FMU, caution is advised. It could be a fake certificate. Global platform providers may be able to securely distribute the certificates of the component manufacturers.

The signature must be stored when FMU is created and checked when FMU is used.

9.9.3 Signing in the FMU itself

Until version FMI 3.0, the topics of certification and signature is not addressed in the standard. Therefore, reference should be made to the best practice described in section 9.9.6 from the DIAMOND project [28].
In general, it should be noted that the entire area of an FMU (including the modelDescription.xml, the resource folder, ...) is included in the signature. This means that all content—except the signing key itself—is integrated. This process must be done before zipping the container. The signing and verification of the certificate can either be implemented by yourself. Otherwise, there are tools (possibly also open source) that support this process and take responsibility in a certain way. The signature must either be integrated directly when building the FMU (especially in the case of FMU templates). Otherwise, it is possible to sign the FMU afterwards. In this context, simple signing tools can be developed or used.
In principle, FMU can be extended with an additional folder e.g. "certificate" (cf. Figure 36), which contains signatures and documents with metadata (information about the certificates/signature) [16]. According to the FMI standard, additional folders must be placed within the "extra" subfolder [11]. Alternatively, it is possible to create a subfolder "signatures" in the folder "documentation", for example. Common ZIP-based signature formats using the "META-INF" subfolder (cf. Chapter 9.9.4) can also be used. However, this creates a new folder at the top level (e.g. "META-INF"), which is not included in the FMI standard at the time of publication of this guideline.

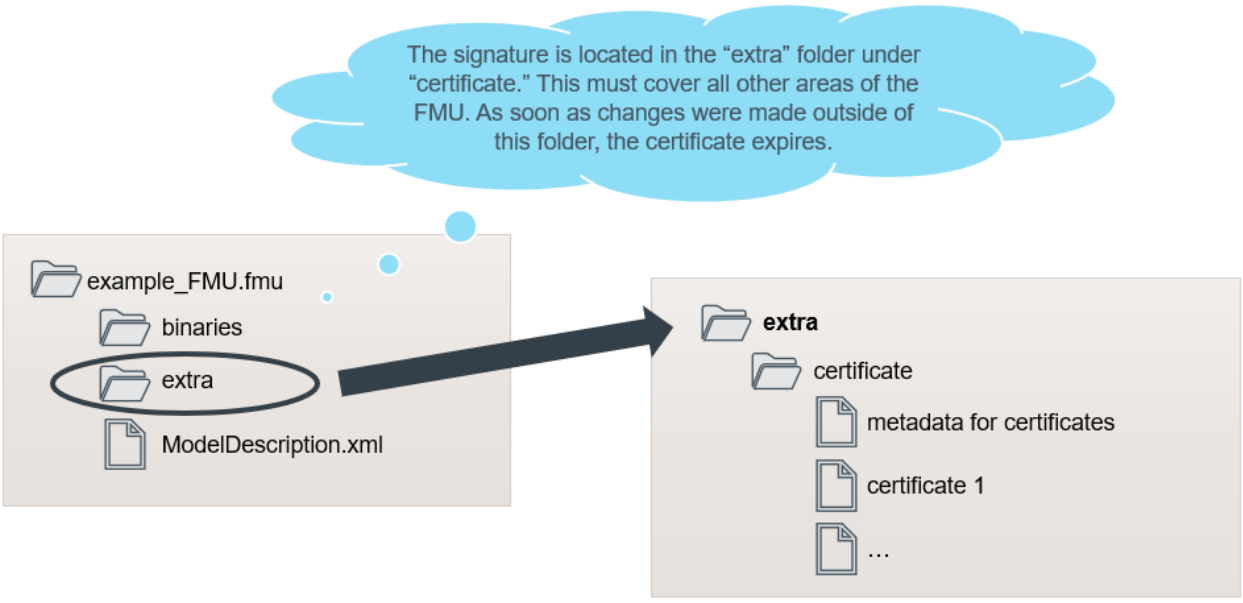

Figure 36: Possible implementation of the integration of certificates into the FMU

Currently, the FMI standard does not specify a specific procedure for signing. Nevertheless, the signature can be stored in the FMU container.

9.9.4 Signing via FMU's zip container

Another option is to set the signature directly via the Zip containers. This does not change the contents of the FMU, which means that the signature covers the entire container. This is not possible in the case of an integration via the "certificate" [16] or "signatures" folder (cf. Chapter 9.9.3).

Since an FMU is a zip container, common procedures for signing a zip file can also be used.

9.9.5 Signing in a parent shell

Under *Further standards* (cf. Chapter 10), various approaches are presented for embedding FMUs into a higher-level system . It is also possible to place the signing outside the FMU. The procedure for all three solutions (AML, AAS, SSP) is analogous [16]. In general, the overarching standards do not provide detailed specifications regarding what should be signed and how the signature should be structured. The process is illustrated by the following Figure 37. In this case, the FMU creator must provide FMU via a further standard. An external signature is only valid if the FMU has been integrated into the structure and is stored in one of the standardized shells. When

using this approach, the FMU can only be exchanged via the respective higher-level standard. The stand-alone FMU itself does not contain a signature.

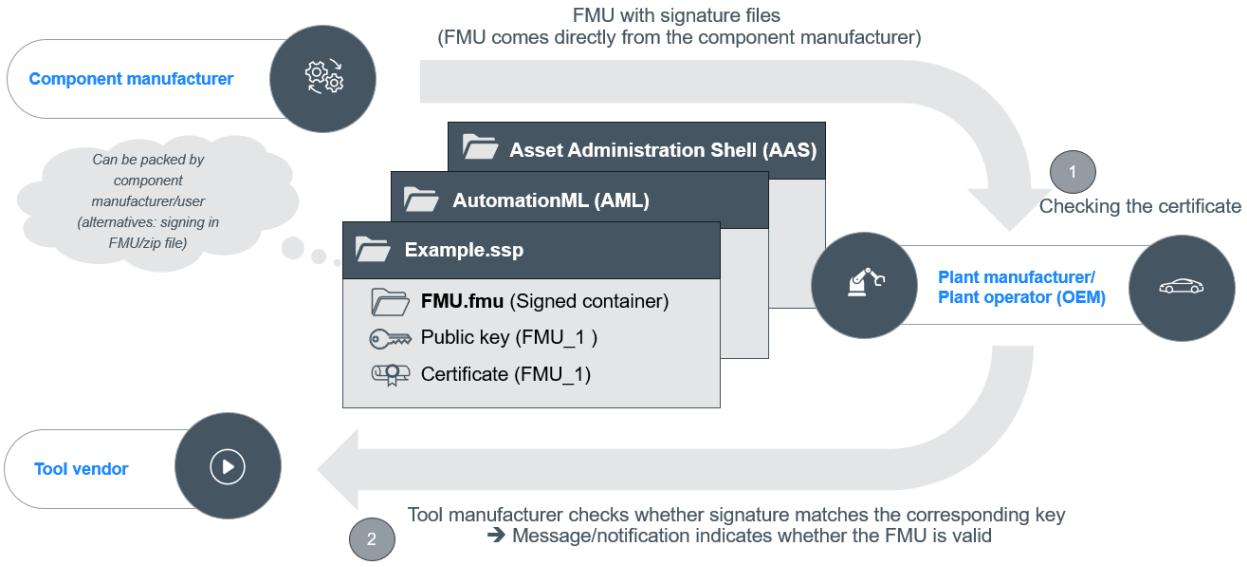

Figure 37: Process of deploying FMUs between roles when signing using higher-level standards (e.g., SSP, AML, and AAS)

A concrete example of the integration of signatures in relation to FMUs via the AML standard has already been examined and validated in the article "Secure Exchange of Black-Box Simulation Models using Functional Mockup Interface in the Industrial Context" [26].

Alternatively, FMUs can be signed using a higher-level standard (e.g. SSP, AML, AAS). A secure exchange of single FMUs is not possible via this route.

9.9.6 Possibilities for uniform use in the VC

As part of the DIAMOND research project, various approaches were explored to enable standardized use of signatures. For this purpose, a best practice has been formulated under [27]. This can now be introduced into the Modelica Association or associations (such as ProSTEP iViP) by means of a "Layered Standard". Furthermore, there is the option of using a decoupled tool for signing and verifying the signature. This outsources responsibility and simplifies the application. Strict use of signatures is essential for the widespread use of FMUs in the VC [16].

In order to enable a uniform approach, best practices must be developed (cf. [27]) and placed in the FMI standard or in associations.

9.9.7 Protection against malware and unwanted activities

The signature generally does not prevent malware (e.g. viruses) from being present in the FMU. Especially when using third-party DLLs, the FMU creator must make sure that they are also free of malware. Signatures only provide a clear proof of origin and define responsibility. Other ways in which security can be increased when dealing with FMUs are listed below:

A preliminary analysis of DLLs is generally difficult. The safest way would be to integrate a malware checker directly into the co-simulator, which performs a check before loading the DLL. An FMU checker, similar to the one used to check modelDescription.xml (in the form of a virus and malware checker) does not exist and can only be implemented with great effort. Under certain circumstances, common virus scanners can detect malware when it is executed. However, but no guarantee can be given in this context.

In general, it is possible to verify certain FMU activities before it is put into operation. For example, it must be ensured that FMU does not establish a connection to a server outside the VC network. Currently, certain real components (IT devices, server units, ...) are analyzed and approved by IT before their use (sandbox tests). Something similar could also be used for FMUs in the future. It is also important that the FMU applications are run without admin rights. This way, important software mechanisms of the operating system (Windows) can remain active.

The VC setup is currently often disconnected from the company network and the internet and is considered an encapsulated system (island network). The reason for this is also the use of product data (e.g. CAD data), which must be particularly protected. In the future, this separation should be avoided as it introduces limitations (e.g., in data exchange or license server access). Malware risks can generally be managed through clear responsibilities and traceability via signatures.

Ideally, the FMU approval process should follow the supplier data release process. Approval via IT (similar to software release) would take too long in the context of VC.

Signing creates traceability but does not exclude malware.

9.10 Stability of FMUs

During simulating via FMUs, many different, encapsulated binary files are executed cyclically. Nevertheless, the stability of the software (co-simulator) must be maintained during the VC. Crashes of programs and, in the worst case, of computers must be avoided, as restarts often result in long delays and waiting times. FMUs must also be

operated stably and potential errors must be handled appropriately. This includes, for example, memory access violations or attempts to access unavailable resources. In some cases, it may be advisable to run FMU simulations in a separate process to prevent issues from causing the main process (tool) to crash. In the best case, the FMUs should be checked and validated during loading. The first errors can already be caught here. To simplify troubleshooting, errors and warnings should be captured and communicated to the user via output mechanisms (e.g. FMU logs, 5.8).

The start of the simulation in the VC tool is often coupled with the start of the FMU co-simulation. The same procedure applies to stopping the simulation. Regarding stability and runtime properties, it is recommended to decouple these two processes from each other.

It is difficult for tool developers to reliably catch faulty behavior. For this reason, the FMU creator must focus on a qualitative and stable implementation. FMU export tools must be thoroughly validated to prevent faulty behavior. It is essential that FMUs are comprehensively tested by the creator and checked for potential errors.

In the VC, a stable plant model is of enormous importance. Robustness must be considered in FMU development. The co-simulator must intercept and log error cases as far as possible.

9.11 Model performance and computational effort

In the context of a VC, it is important that the behavioral model can be calculated in a certain cycle time. This depends on the respective hardware, the operating system, but also on the load and the number of simulated FMUs. In general, the implementation of the FMI standard in the co-simulator, the computer setup used, the hardware used and software running in parallel also influence the performance of the FMUs.

Compared to the controllers (PLC) connected in the VC, the cycle time of FMUs must generally be the same or shorter. If the behavioral model runs slower than the controller, this can lead to problems. The FMUs are then not sufficiently "real-time-capable" for this application. Exceptions here are behavioral models that do not replicate the components, but processes. Cycle times of up to 500 ms are also conceivable here. In the automotive industry, control systems in complex systems sometimes run with a cycle time of 20 – 200 ms. Normally, these are significantly lower (approx. 1 – 2 ms). Component FMUs must be within the range of these cycle times, otherwise a smooth VC cannot be guaranteed. In the Windows environment, a cycle time of <10ms in a common VC co-simulator is to be aimed for. Delayed feedback from FMU means that monitoring times in the PLC can no longer be met, which means that error messages from the real PLC code limit the VC.

The model performance must not suffer from a high level of detail. With a high cycle time (>50ms), it is important to reduce the level of detail (LoD) of the model and optimize the FMU accordingly. The FMU must achieve a specified cycle time (<10ms). Often, it is not the FMU, but the co-simulator that is the limiting factor, as it usually

only allows a minimum cycle time of 1ms. Many FMUs can be simulated much faster. In general, FMUs and co-simulators are very performant. This makes it possible to simulate several hundred FMUs with a cycle time <10ms. When creating FMU, performance tests of this magnitude (depending on the component) should be carried out.
In principle, work should be carried out according to the principle of "as quickly as possible". Especially when different FMUs are synchronized with each other via the co-simulator or with other tools, cycle times of 1 – 2ms are important. Often, all instantiated FMUs are synchronized with each other and run in one clock. Thus, slow FMUs may slow down the entire simulation. In systems with different time slices, faster FMUs can also be operated more slowly. The cycle time under consideration is the time between two communication steps at which the inputs, outputs and parameters are exchanged with the rest of the simulation. Within FMU, however, it is possible to work with finer or even coarser steps. If the internal cycle times do not match the co-simulation times, interpolation can be used. In the standard version 3.0, so-called timers and clocks are also offered, which can be used for synchronization. The handling in this context is complex and is currently not being used productively.
In this context, the tool manufacturer (of the co-simulator) is expected to display or log the respective cycle times of the individual FMUs. This makes it easier to identify performance issues and high cycle frequencies. If the VC tool integrates the co-simulator, an asynchronous connection is recommended. This allows FMUs with dynamic and fixed step times (fixed/dynamic step size) to be connected. In a synchronous and coupled simulation, "slow" FMUs can slow down the entire VC tool. This is often referred to as performance issues associated with FMUs. It is recommended to outsource the co-simulator to its own process so that the integration of FMUs does not have a negative impact on the tool itself.
By co-simulating behavioral models (FMUs), the corresponding computational effort can be outsourced as desired. In this way, critical FMUs can be distributed to a new process, a different CPU core or even to other computers and servers. It should be noted that the number of simulated FMUs has an impact on performance. Every FMU requires resources, even if the complexity of the FMU itself is kept low. The FMUs must be distributed on the target system as a thread or task. This generally supports a combination of simple components into a system/module (cf. Chapter 6.2). Due to the outsourcing via co-simulation, several thousand FMUs can be stably simulated with a common level of detail. As the number of FMUs increases, the time to start the simulation usually increases. Loading, unpacking, and instantiating the FMUs may take several seconds (depending on the implementation).
The outsourcing of computing effort is always strongly dependent on the specific application. Latency times and irregularities in communication with other computers, servers, etc. often require a simulation directly on the target system. This is the biggest obstacle to simulating FMUs in the cloud. The interfaces are usually too slow here. This means that computationally intensive FMUs, which exchange less time-critical signals (possibly artificial intelligence (AI) applications with neural networks), can be easily outsourced to other systems.

The distribution of the behavioral models is accompanied by a cyclical exchange of interface signals (inputs, outputs, parameters). There is communication between different processes and systems. In this context, it is recommended to limit the number of interface signals to the essentials. When scaling FMUs, the signals quickly add up and lead to large amounts of data, which must be exchanged cyclically every millisecond. Speed suffers as a result. Especially in the case of cross-system distribution, the high-performance shared memory cannot be used. In this context, reference is made to section 9.4.

The FMUs must be developed as high-performance as possible, so that cycle times of <10ms can be realized. The co-simulator (VC tool) must distribute the computing effort in a targeted manner.

10 Further standards

None of the standards listed below compete with the FMI standard or exclude the use of FMUs in the context of behavioral modeling. All these standards offer the possibility to integrate FMUs and enrich them with additional metadata. The co-simulator can implement a connection to the aforementioned standards so that FMUs are automatically extracted and simulated. The standards serve a uniform exchange of models. They can therefore also be used by the component manufacturer to deliver the FMUs to the user. The standards can be integrated or referenced with each other.

The FMI standard is compatible with the SSP, AAS and AML standards and is not in competition with them.

10.1 System Structure and Parameterization (SSP)

The Standard *System Structure and Parameterization* (SSP) comes from the Modelica Association [1], analogous to the FMI standard. At its core, it is used to link several FMUs with each other and to define different parameter sets. In the context of behavior modeling for the VC, it is possible to combine several FMUs into an overall network and to parameterize them (cf. Figure 38). This unit can be integrated into the structure of the overall system by means of a higher-level standard, such as AML.

The component manufacturer can use the SSP standard to connect individual components to build a module or unit. Thus, the standard allows to flexibly design both individual components and systems (cf. Chapter 6.4). For example, a complete system consisting of several components (e.g. gantry, pneumatic unit) can be provided as an SSP together with all participating individual FMUs as a complete solution. Furthermore, SSP is used to parameterize FMUs. In this way, generalized FMUs (cf. Chapter 6.5) can be configured for the specific use case. In general, the standard is very extensive and is currently only used to a limited extent in the VC. For example, SSP

also offers the possibility to define different parameter sets and perform parameter optimization in complex application scenarios.

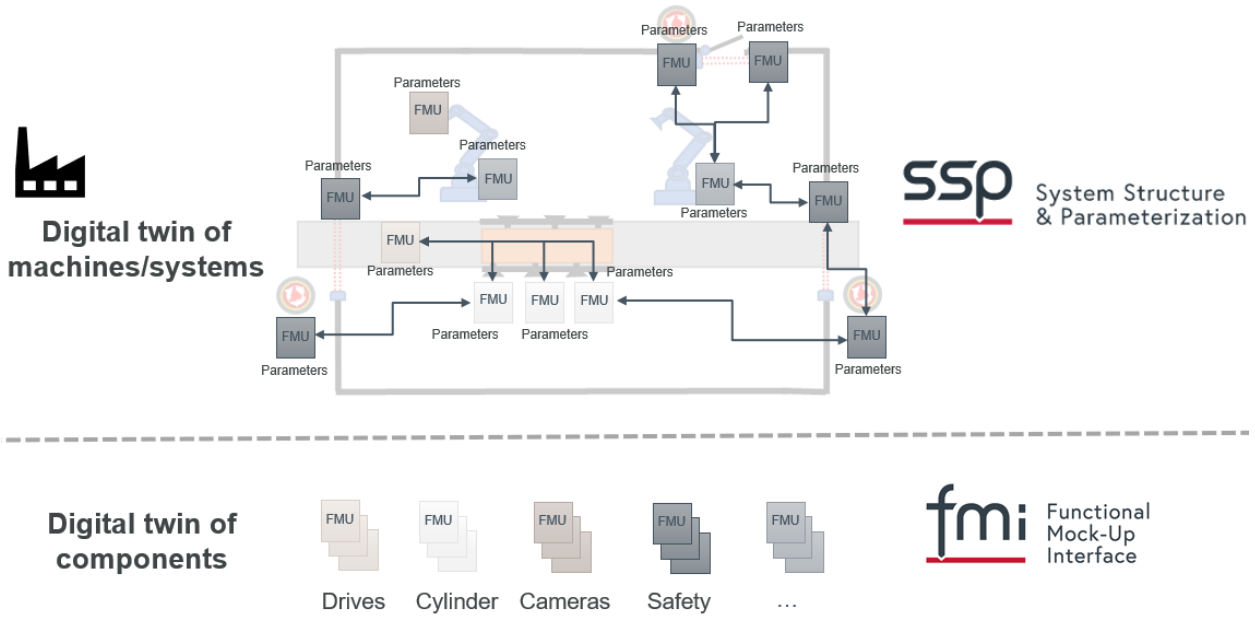

Figure 38: Schematic structure when merging FMUs into a SSP

In the context of the VC, the SSP standard can be used for a so-called "wrapper" to continue enabling the connection of previous automatisms and assistance. The signal names of the FMUs are renamed to the respective company standard. In addition, it is possible to assign the instance names and to set specific parameterization. The SSP standard allows signals to be interconnected internally and thus reduce the scope of interface signals. This increases performance (cf. Chapter 9.4). Such signal links can be created automatically by assistants.
With reference to section 9.9 the SSP standard can also be used to store the signatures of the individual FMUs (cf. Figure 39). A procedure has already been defined in the SSP standard and published in version 2.0 [12].

Figure 39: Integration of signatures via the SSP standard in version 2.0

Compared to the FMI standard, SSP has so far been poorly integrated into the VC tool landscape. Only a few applications support the creation and simulation of SPLC. In general, the SSP standard has only been implemented in 15 tools so far [12]. Nevertheless, productive projects have already been successfully carried out in the context of the VC with the help of SPLC. Several hundred FMUs were integrated and interconnected with each other. The SSP standard will become more important in the future, as it places a shell over the FMUs and is suitable for integration into other structures (such as AAS and AML, cf. Chapter below).

The SSP standard can be used to interconnect and parameterize individual FMUs. SSP includes a procedure for signing.

10.2 Asset Administration Shell (AAS)

The asset administration shell provides a technology to transport and transmit data and engineering content. The standard is formulated and further developed by the Industrial Digital Twin Association e.V. (IDTA) [36]. For many component manufacturers, AAS offers a leading format for digital information. In the context of DIAMOND, the asset administration shell is used to exchange data and information between component manufacturers and plant manufacturers via a standardized structure (container). The asset administration shell has the advantage that the data can be stored as bundles and loaded independently of the target tool. The standard includes, for example, the integration of CAD data, documentation and behavioral models (FMUs). The FMUs are integrated into the simulation submodel (SM_Simulation, cf. Figure 40) [3]. In the future, some component manufacturers want to provide FMUs only within an asset administration shell and not as a single file.

In the context of the asset administration shell, two different types can be distinguished:

- **Type 1:** Exchange via the aasx file, which is realized as a zipped container.
- **Type 2:** Exchange via a server that provides data and information on demand.

The integration and publication of an FMU via the Type 2 asset administration shell has already been investigated as a prototype and represents a possibility for the provision of FMUs. The behavioral models are offered as part of an AAS via a server. Conceivable implementations are the integration of AAS into the shop or the website of the component manufacturer. Type 2 offers different variants. On the one hand, it is only a deployment. However, it is also possible to operate FMU directly on a server and integrate it into the simulation via an interface. In this case, the co-simulation takes place directly on the server. The FMU methods (e.g. doStep) can be accessed via an API. This allows integration with the VC tool. A serious disadvantage of this procedure is the latency period when communicating with the server. This approach can be seen as a long-term goal. With regard to safety (cf. Chapter 9.9), this concept also offers advantages, as the FMU is executed on an external server and thus runs decoupled from the setup/company network.

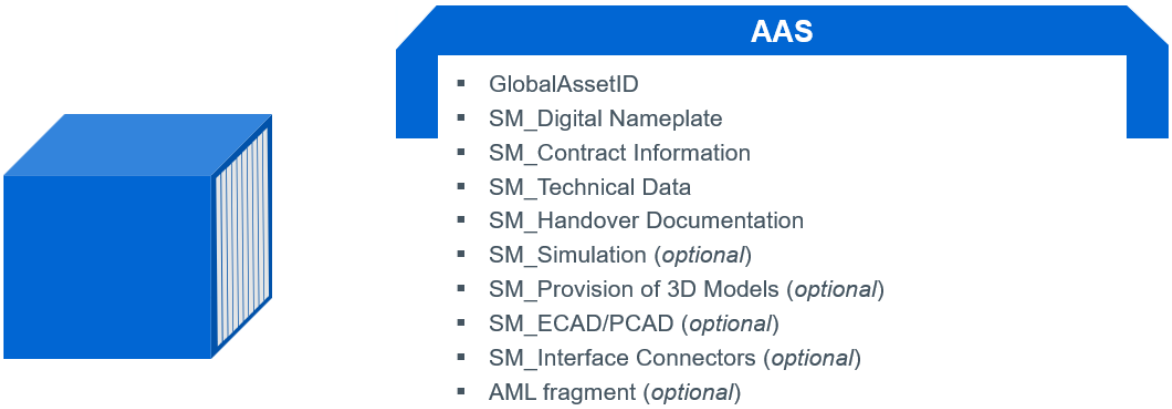

Figure 40: DIAMOND Recommendation for Component AAS (FMU in SM(Submodel)_Simulation)

Some of the information (e. g. EAs, FMI version, FMU versioning, generation tool, date, author) may be stored redundantly and is located in both the modelDescription.xml and the AAS. The asset administration shell offers the possibility to store additional metadata. Examples include the tool used to validate the FMU, the purpose and scope of the model, or the limitations of the simulation (e.g., operating system). In addition, the versioning of FMUs can also be managed. Via the so-called interface connectors (SM_Interface Connectors, cf. Figure 40) [4] AAS allows interfaces to be offered as part of the asset administration shell. The standard can also be used to inform the user as soon as the component has been discontinued by the manufacturer or in which area of validity

the component is to be used (SM_Technical Data, cf. Figure 40) [37]. Furthermore, AAS provides the ability to monetize digital purposes and provide different levels of detail of data [3].

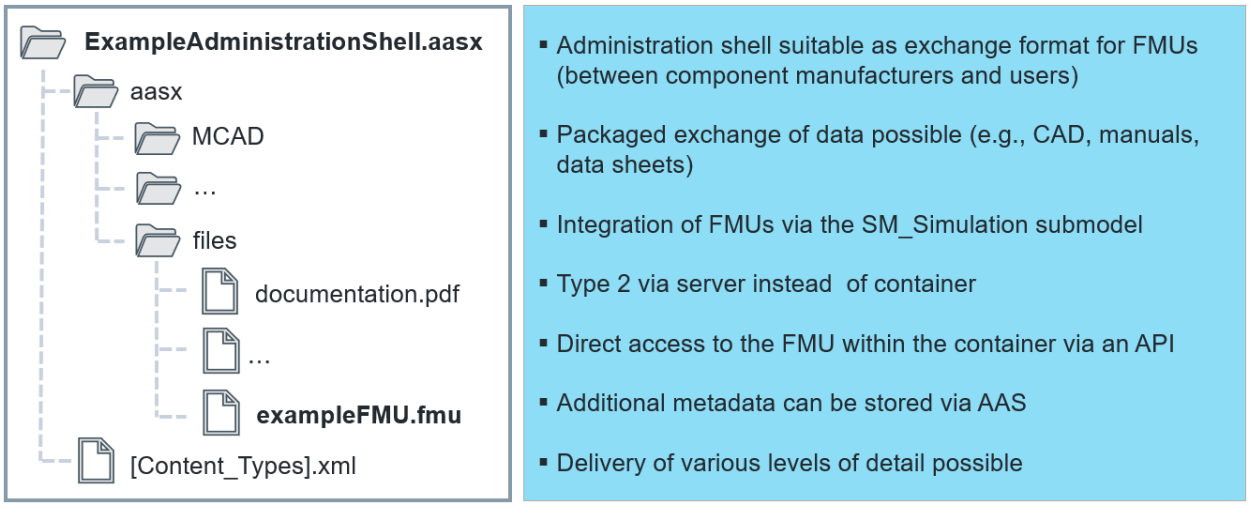

Figure 41: Structure and overview of the integration of FMUs into the asset administration shell

In general, the implementation and use of the asset administration shell (AAS) simplifies the integration of the FMI standard into the company. Through the resulting standardization and bundling of master data, parts of the FMU can be automatically delivered. In addition to FMUs, SPLC can also be linked in an AAS. In relation with the asset administration shell, a reference should be made to the AAS Reading Guide [10], which links to relevant documents for the respective target group.

AAS offers the possibility to exchange FMUs together with other documents and information in a standardized way. Metadata and signatures can also be stored in this higher-level shell.

10.3 AutomationML (AML)

As part of DIAMOND, AutomationML makes it possible to design and describe complex systems during engineering. AML defines the connection of the components in the entire system, including the respective wiring, and is created by the plant manufacturer. The component manufacturer does not have to deal with AML in the first place and to provide data in this form. According to the current approach, AML is only to be supplied by the component manufacturer on request. In the case of modules and entire systems, such as a handling portal, it is conceivable that the component manufacturer also provides related information via AML. Logics and behavioral patterns can only be described with AML to a limited extent. The possibilities that AML provides here are described in [2]. In order to describe complex relationships, component FMUs must be linked to each other. Compared to

SSP, individual behavioral models can also be linked and parameterized via AML. While SSP only allows the linking of FMUs (behavioral modeling), AML offers the possibility to connect additional metadata as well as other engineering data (e.g. CAD, PLC). SPLC can also be integrated into the AML structure.
In general, AML is already being used productively in plant engineering. For example, information is extracted from EPLAN via an AML export and imported into the TIA Portal. The AutomationML standard is also already being used in the CAD environment for data exchange.

AutomationML is used for the standardized description of complex plants during engineering.

10.4 Integration of AAS and AML in the DIAMOND project

In the DIAMOND project the AAS and AML standards are used as transport or referencing options for FMI/SSP (cf. Figure 42) *[8,5]*:

1. Component manufacturers offer the behavioral model as FMU via the asset administration shell. The FMU is included in this via a sub-model. (cf. Figure 40) [3]
2. At the plant manufacturer, the information of the individual component AAS is integrated into an AML plant engineering structure (as external) via an engineering tool. The FMU from "1." is referenced via the AML structure. Integration via AML is used throughout the plant engineering process. (AML structure with referenced AAS of each sub-component)
3. When it is handed over to the plant operator (OEM) at the end of the plant development process, the system is handed over in the form of an asset administration shell (AAS). This references the AML structure and thus documents the engineering (asset administration shell for the entire plant). This process is not the focus of DIAMOND.

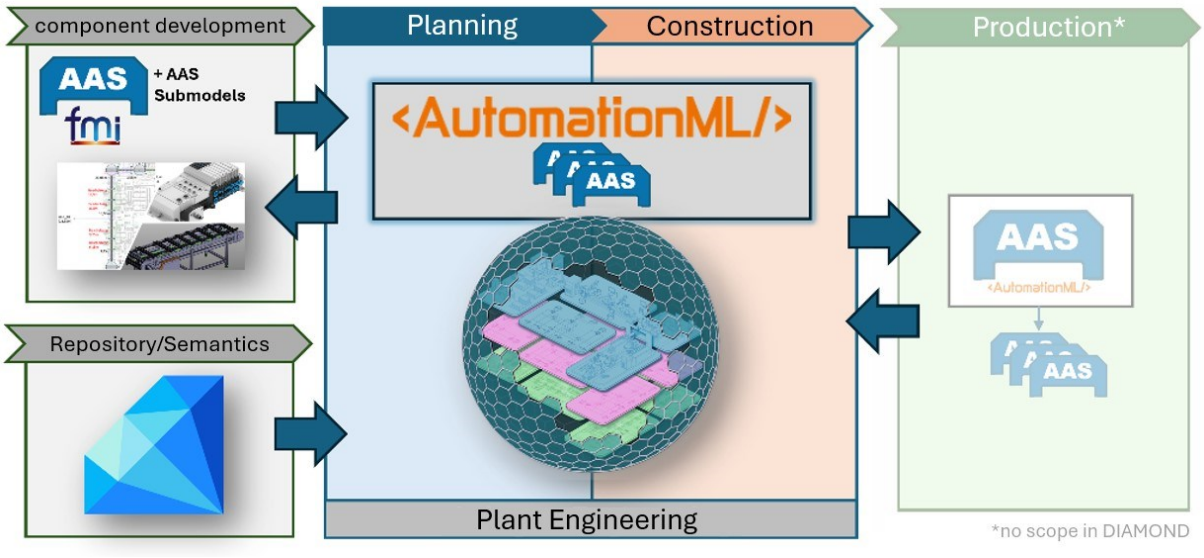

Figure 42: Interaction of AML and AAS during the engineering process in the context of the DIAMOND project [8]

During the engineering process, FMU, SSP, AAS and AML can be used.

11 Outlook: Highly detailed behavioral models as FMUs

Generalized models as FMUs can be used in various engineering phases for corresponding applications. The standard also allows highly complex models to be developed. However, with the provision of highly detailed and realistic models, the risk of know-how protection and the risk of reverse engineering increases. From a certain level of complexity, the focus is on the topic of simulation and not on the VC with its real-time requirements. In all cases, it should be noted that these additional aspects must not affect the VC in any way. If compromises in model performance or stability problems are to be expected, a separate FMU must be defined specifically for the extended use cases. The focus is on a behavioral model with all basic functionalities, which does not represent a 100% representation of the real component. Nevertheless, the potentials of highly detailed behavioral models are listed below.

Highly detailed FMUs must not have a negative impact on the VC under any circumstances.

11.1 Use of AI in the field of FMU creation

With a higher level of detail, the effort involved in model generation also increases. The dependencies on various influencing factors make modelling increasingly complex (e.g. influence of friction, acceleration, mass). Modeling based on logical and physical relationships is time-consuming and performance consuming. In this area, AI plays an important role in delivering cost-effective, high-performance, and realistic models. For example, the behavioral model can be taught or optimized based on a real component. It is possible to compare an existing model with the real component (gap analysis). The differences can be reduced using AI methodologies.

The relevant data can be collected about the real plant/component. It is also possible to generate synthetic data. Under certain circumstances, documentation and manuals can also be read automatically with the help of AI and used as a data basis. In addition, it is conceivable to use AI to bundle the source code of the real component and derive an FMU from it. However, it is questionable to what extent a component manufacturer wants to make its source code (and thus its know-how) available to an AI. It also needs to be clarified to what extent AI may be used in the context of a VC. This is to be classified as critical, especially with regard to safety and liability for damage to the real system.

From a technical point of view, it is possible to use AI methodologies and functions in the context of the FMI standard. For example, parameters can be found and optimized, which are passed on to FMU. It is also possible to operate AI operations directly in the FMU. The TensorFlow framework, for example, offers the possibility of integrating AI directly into FMU. However, it should be noted that such operations can strongly influence the cycle time and loading behavior of FMUs. For this reason, such models are currently used primarily in research projects. In the productive environment, they do not meet the requirements for achievable cycle times.

Another way in which FMU creation can be supported by AI is through chatbots. Because FMI is an open, easily accessible standard, AI can learn from publications and the standard itself. For this reason, rough statements about FMU creation are already possible in common chatbots. However, the utmost caution is required in this context. The statements as well as the models (FMUs) created via chatbots absolutely need to be validated. A chatbot can cause poor and invalid models to interfere with productive VC operations.

11.2 Use of FMUs for component design

Detailed behavioral models provide feedback at an early stage as to whether the selected component has been correctly dimensioned. For example, before procurement, it can be checked whether the component is suitable for its intended use. In this early project phase, changes to individual components are still possible.

11.3 Energy consumption

In terms of sustainability, and energy and cost savings, the energy consumption factor is increasingly coming into focus. Direct conclusions can be drawn about the associated emissions from consumption. It is conceivable that laws or guidelines will be defined in the future that require a clear statement on the energy consumption of a system. It is therefore important to enrich FMUs with the aspect of energy consumption in the future. This allows the energy of a production plant to be estimated, simulated, and optimized in advance. It becomes clear how much energy a production plant needs and where energy flows lie. Initial studies with robots have shown that the savings potential through suitable models is enormous. This idea is currently being addressed by several research projects. However, the topic is very complex, as many parameters and influencing factors must be considered (e.g. pressure, times, friction). Detailed physics must be mapped in the behavioral model and visualization. Complex and validated behavioral models are required for a realistic energy analysis.

The topic of energy consumption and sustainability can be expanded even further: Data from the production of the component and the life cycle can also be integrated. Energy flows make it possible, for example, to analyze the waste heat from components (e.g. compressors) and to use it in a targeted manner. The FMI standard itself provides sufficient functionalities here. In this context, the topic of recording energy consumption and CO2 footprint (PCF) in connection with the asset administration shell (cf. Chapter 10.2). For example, a corresponding value is stored in the production of the components. In the AAS, this can be extended with the consumption during operation.

Overall, initial estimates can already be made using simple models with standard consumption calculation. The energy rating does not necessarily have to correspond 100% to reality. Initial analyses already allow a rough optimization.

It is also possible to compare the actual energy consumption with the simulated energy consumption in parallel to real plant operation. In this way, an "energy observer" is created (cf. Figure 43), which in case of deviations (analogous to predictive maintenance, cf. Chapter 11.6) provides feedback to the operator. The real system can be monitored in this way. This makes it possible to detect leakage in compressed air applications, for example.

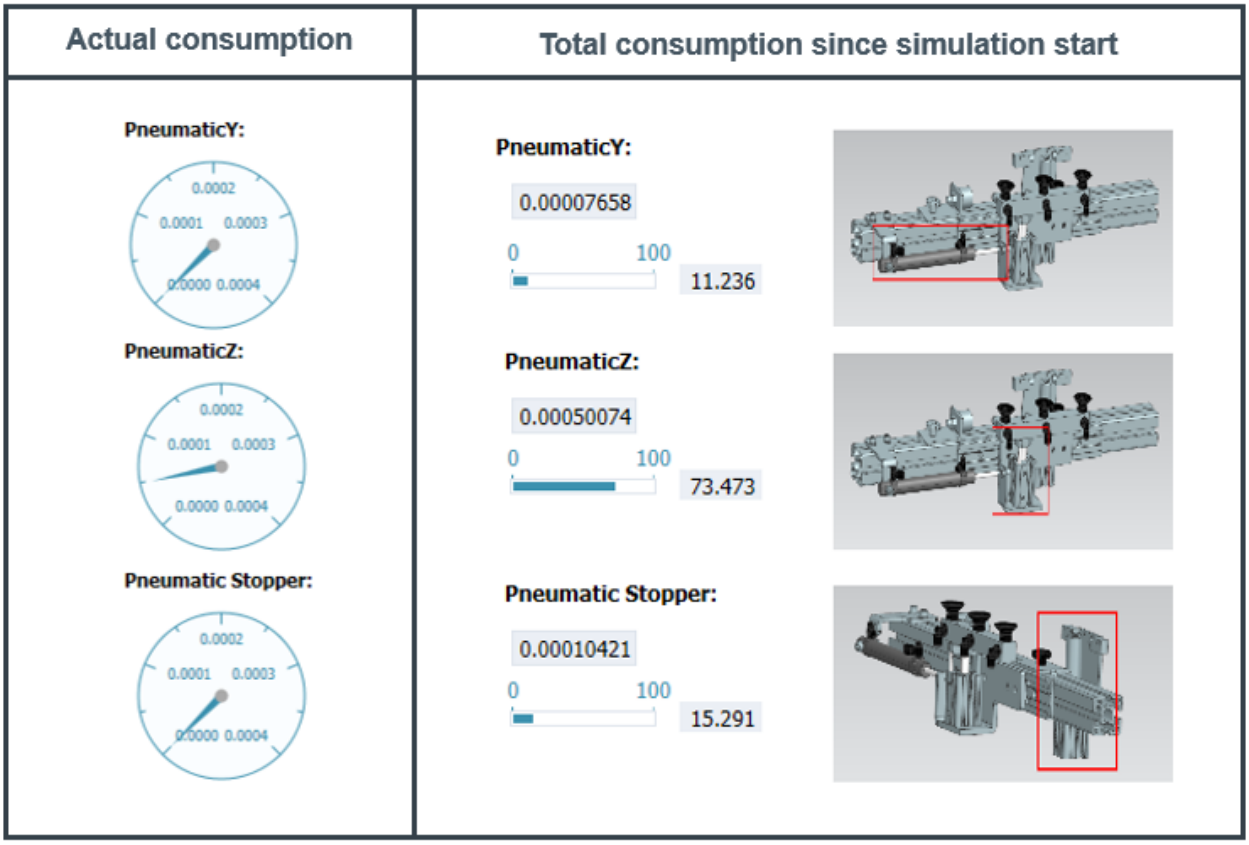

Figure 43: Energy consideration using the example of air consumption of a pneumatic cylinder. The current consumption (left) and the total consumption since the start of the simulation (right) are illustrated

11.4 Detailed physics simulation (following error of drives)

Detailed simulations are of interest for specific use cases. These include, for example, pumping, gluing or dosing processes. Detailed simulations are also important in the field of motor and gear models. The FMI standard enables detailed modelling of physical relationships. For example, temperature examinations can be carried out. It is possible to analyze overload cycles and to support the design. If the component (e.g. a drive) behaves differently than expected, one can react to.

In positioning drives, the simulation of following errors is important. This allows control and setting parameters to be estimated in advance. In addition to the required detailed behavior model, a meaningful physics simulation with inertia is also necessary here. In the context of the VC, there is currently a need for action here, as physics is often strongly abstracted to achieve the necessary stability and cycle time. Currently, parameterization is usually done via a so-called "auto-tuning", which takes place via a reference run on the real component.

11.5 Use of the FMI standard for control simulation

If simulation or emulation software is not available for certain controllers (e.g. PLC, RC), the FMI standard can also be used to simulate control code. FMUs can be derived from the programming tools, which are then connected to the simulation (digital twin) via the interfaces (may be also via Robot Operating System (ROS)). A FMU as a virtual controller can also be simulated in parallel with the real controller and provide certain information about the input/output relationships (e.g. via OPC- UA).

11.6 FMUs at the plant operator (predictive maintenance)

By using FMUs during operation of the real plant, the plant operator receives additional data and information about the components. These can be returned to the component manufacturer. The challenge is to maintain and further develop the digital twin in parallel to the real plant. For example, parameters (e.g. control parameters) and configurations from the real plant must be fed back into the digital twin. This effort would be significantly reduced as soon as FMU independently obtains data from the real system and adjusts the parameters itself. Ideally, the behavioral model (FMU) learns directly from the real system and optimizes itself. FMU can be connected via OPC-UA, for example.

If the twin is simulated in parallel with the real system, the maintenance technician can quickly understand and fix a fault by repeating the event on a digital twin. With more detailed behavioral models, it is easier to reproduce the error. By use of FMUs, the real data can be compared with the simulated, ideal data. This enables activities towards predictive maintenance. The evaluation itself can also be integrated directly into an FMU, for example using AI approaches (cf. Chapter 11.1). The observation and feedback of live data from a real automation component via a server has already been implemented by component manufacturers and could serve as a counterpart to the idealized FMU. In addition, the asset administration shell can also be used to show where and how the respective components are used by the plant manufacturer, for example. Such life cycle information of the real components is used for improvements in new developments.

12 Conclusion

Virtual commissioning offers great potential for improving and accelerating the development of production facilities. To make this process possible, extensive library of behavioral models are required. In addition to the effort currently being put into behavioral modeling, incomplete and incorrect models are a major challenge in creating a digital twin for VC. Since every user or tool manufacturer has had to develop their own models so far, the effort and the resulting costs for creation and maintenance are correspondingly high. In many cases, behavioral modeling makes the creation of a digital twin (especially for new introductions) and uneconomical in medium-sized companies.

In the future, the complexity of production plants will also continue to increase. The connection and interconnection of highly intelligent automation components requires a higher level of programming effort, which can only be controlled by the VC. Accordingly, the complexity and the requirements for the behavioral models of the respective components are also increasing. The creation of models based on manuals and PLC modules is becoming more and more time-consuming and error prone. For this reason, it is essential that component manufacturers deal with the topic of behavior modeling and provide standardized models. In this context, the FMI standard meets all the requirements from technical point of view. In order to facilitate the secure exchange of FMUs with regard to cyber security and to ensure traceability of changes to the model, a best practice [27] has been published in the context of the DIAMOND project (cf. chapter 9.9.6). This describes how signatures can be applied in the FMI standard.

The guide provided here is intended to minimize the familiarization with the FMI standard and increase the quality of the models provided. The introduction of FMI is considered across roles, which is intended to remove previous obstacles and questions. The goal is to provide a behavioral model as an FMU for each real-world component. In the future, this will also be integrated and required into the approval process at the OEM. The development should be based on the established provision of CAD data. Standardized behavioral models (FMUs) make it possible to realize end-to-end digitalization. Without a digital twin, the increasing complexity of the components and plants can no longer be handled in the future, and delivery dates can no longer be met.

The DIAMOND research project is financed by the European Union (EU) and funded by the German Federal Ministry for Economic Affairs and Energy (BMWE).

Figure 44 Logo of funding institution BMWE and DIAMOND

Table of figures

Table index

List of abbreviations

3D	**Three dimensional**
AAS	**Asset Administration Shell**
AML	**AutomationML**
API	**Application Programming Interface**
BDE	**Operational Data Collection (Betriebsdatenerfassung)**
BMK	**Equipment Labeling (Betriebsmittelkennzeichung)**
BMWE	**German Federal Ministry for Economic Affairs and Energy**
CAD	**Computer-Aided Design**
CC-BY-SA 4.0	**Creative Common Attribution-Share Alike 4.0**
COM	**Commissioning**
CPU	**Central Processing Unit**
DIAMOND	**Digital plant modeling with neutral data formats**
DIN	**German Institute for Standardization**
DLL	**Dynamic Link Library**
EA	**Input Output**
ECE	**Economic Commission for Europe**
ECU	**Electronic Control Unit**
FMI	**Functional Mock-up Interface**
FMU	**Functional Mock-up Unit**
GSDML	**Generic Station Description Markup Language**
GUI	**Graphical User Interface**
HiL	**Hardware in the Loop**
HMI	**Human-Machine Interface**
IDTA	**Industrial Digital Twin Association e.V.**
IO	**Input Output**
ODD	**IO Device Description**
IoT	**Internet of Things**
AI	**Artificial intelligence**
LoD	**Level of Detail**
MDE	**Machine data acquisition**
MiL	**Model in the Loop**
MKS	**Multi-body simulation**

NDA	**Non Disclosure Agreement**
OEM	**Original Equipment Manufacturer**
OPC	**Open Platform Communications**
OPC UA	**OPC Unified Architecture**
PCF	**Product Carbon Footprint**
PDM	**Product Data Management**
PKCS	**Public Key Cryptography Standards**
PKI	**Public Key Infrastructure**
PLC	**Programmable Logic Controller**
RC	**Robot Controller**
RFID	**Radio Frequency Identification**
ROS	**Robot Operating System**
RSA	**Rivest-Shamir-Adleman, cryptographic method**
SHA	**Secure Hash Algorithm**
SHM	**Shared Memory**
SiL	**Software in the Loop**
SO	**Shared Object Library**
SSP	**System Structure and Parameterization**
SVN	**Subversion**
TCP/IP	**Transmission Control Protocol/Internet Protocol**
TIA Portal	**Totally Integrated Automation Portal**
UI	**User Interface**
VC	**Virtual commissioning**
XML	**Extensible Markup Language**

Bibliography

[1] Modelica Association (2025). “SSP System Structure & Parameterization”. URL: https://ssp-standard.org/ (visited on July 30, 2025)

[2] AutomationML consortium (2020). „<AutomationML/> The Glue for Seamless Automation Engineering, Whitepaper AutomationML Part 4: AutomationML Logic”. In: *WP Logic, V 1.6.0*. URL: https://www.automationml.org/wp-content/uploads/2024/01/AML_Whitepaper_Part4_Logic_V1.6.0.zip (visited on July 04, 2025)

[3] IDTA (2022). „Provision of Simulation Models, Submodel Template of the Asset Adminsitration Shell”. In: *Specification IDTA 02005-1-0*. URL: https://industrialdigitaltwin.org/wp-content/uploads/2023/01/IDTA-02005-1-0_Submodel_ProvisionOfSimulationModels.pdf (visited on July 04, 2025)

[4] IDTA (2025). „Interface Connectors, Submodel Template of the Asset Adminsitration Shell”. In: *Specification IDTA 02062-1-0*

[5] Dörner L., Hundt L., Mersch T., Schleipen M., Freund M., Paul M., Lüder A., Kose K. (2025). "Synergies through Standards in the Plant Development Process – DIAMOND – AML and AAS in Engineering". In: *trade journal atp magazin* (July 15, 2025: submitted and accepted)

[6] Modelica Association (2025). "fmi Functional Mock-up Interface". URL: https://fmi-standard.org/ (visited on 01.07.2025)

[7] PKWARE Inc. (2022). „.ZIP File Format Specification - Version: 6.3.10”. URL: https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT (visited on June 30, 2025)

[8] Mersch T., Schleipen M., Kuhlenkötter B., Hypki A., Burlein J., Schlögl W., Ach M., Göbeler C., Freund M., Paul M., Dörner L., Lüder A., Hünecke P. (2025). "Data consistency in the plant development process - DIAMOND - Model and Realization". In *the journal atp magazin* 67.04. DOI:10.17560/atp.v67i4.2775

[9] ECLASS (2025). „Technical specification: Structure and structural elements: Unit”. URL: https://eclass.eu/support/technical-specification/structure-and-elements/unit (visited on May 28, 2025)

[10] Neidig J., Orzelski A. & Pollmeier S. (2022). "Asset Administration Shell Reading Guide". URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/AAS-ReadingGuide202201.pdf?__blob=publicationFile&v=1 (visited on May 27, 2025)

[11] Modelica Association (2024). “Functional Mock-up Interface Specification v3.0.2”. URL: https://fmi-standard.org/docs/3.0.2/ (visited on May 20, 2025)

[12] Modelica Association (2025). „Tools that support SSP”. URL: https://ssp-standard.org/tools/ (visited on May 20, 2025)

[13] Preston-Werner, T. (2025) "Semantic Versioning 2.0.0". URL: https://semver.org/#semantic-versioning-200 (visited on May 20, 2025)

[14] ITEA (2011). “MODELISAR: From System Modeling to S/W running on the Vehicle”. URL: https://itea4.org/project/modelisar.html (visited on May 20, 2025).

[15] ITEA 2 Office (2012). „Project Results: Speeding automotive ECU development”. URL: https://itea4.org/project/result/download/5533/MODELISAR%20Project%20results%20leaflet.pdf (visited on May 20, 2025)

[16] Wolf, C., Schleipen, M., Paul, M., Krsticevic, T., Neveu, F., Wincheringer, C., Wünsche, H., Ziegenhagel, V., Zellmaier, T., Moghaddam Nejad, A., Hammerl, F. & Frey, G. (2025). "FMI in the VC: A Guide to Implementation" In: *VDI Congress AUTOMATION 2025* (as of July 15, 2025: submitted and accepted).

[17] Puntel Schmidt, P. & Fay, A. (2015). "Consistent simulation models for the virtual commissioning of production equipment with the help of rule-based model connectors". In: *16th ASIM Symposium Simulation in Production and Logistics*

[18] Sauer, O., Schleipen, M. & Ammermann, C. (2010). "Digital Factory Operation - Virtual Manufacturing". In: *Symposium Simulation in Production and Logistics 2010*

[19] Bergert, M. & Diedrich, C. (2008). "End-to-End Behavior Modeling of Equipment for the Generation of Digital Simulation Models of Manufacturing Systems". In *the journal atp magazin* 50.07. DOI: 10.17560/atp.v50i07.2058

[20] Kiefer, J., Wack, K.-J. & Manns, M. (2010). „Cross-functional digital production validation framework for automotive industry”

[21] Kiefer, J., Ollinger, L. & Bergert, M. (2009). "Virtual Commissioning - Standardized Behavior Modelling of Mechatronic Equipment in Automotive Body Shelling". In *the journal atp magazin* 51.07. DOI: 10.17560/atp.v51i07.92

[22] Hämmerle, H., Strahilov, A. & Drath, R. (2016). "AutomationML in practical use: Experiences with virtual commissioning". In *the journal atp magazin* 58.05. DOI: 10.17560/atp.v58i05.567

[23] Kiefer, J., Prieurs, M., Schmidgall, G. & Bär, T. (2009). „Digital planning and validation of highly flexible manufacturing systems in the automotive body shop”. In: *Proceedings of CIRP MS*

[24] Mewes, J. & Wegener, F. (2009). "Virtual commissioning of conveyor systems with fieldbus emulation and material flow simulation". In: *GMA-kongress "automation"*

[25] Kegel, G., Rauscher, B., Otto, M.-A., Gehlen, S., Nagel, J & Weber, H (2024). "The Importance of Sensor Technology for the Fourth Industrial Revolution". In: *Handbuch Industrie 4.0: Band 2: Automatisierung*, DOI: 10.1007/978-3-662-58528-3_142

[26] Wolf C., Schleipen M.; Frey G. (2023). „Secure Exchange of Black-Box Simulation Models using Functional Mockup Interface in the Industrial Context”. In: *Proceedings of the Modelica Conference 2023*, DOI:

10.3384/ecp204487. URL: https://ecp.ep.liu.se/index.php/modelica/article/view/959/900 (visited on May 21, 2025)
[27] Wolf, C. (2025). "Christian-wolf-eks/ls-sign". URL: https://github.com/christian-wolf-eks/ls-sign (visited on May 21, 2025)
[28] Unified Nations Economic commission for Europe (2010). “Recommendation No. 20: Codes for units of measure used in international trade” URL: http://www.unece.org/fileadmin/DAM/cefact/recommendations/rec20/rec20_Rev7e_2010.zip (visited on May 27, 2025)
[29] Creative Commons (2025). „CC BY-SA 4.0 Namensnennung-Share Alike 4.0 International”. URL: https://creativecommons.org/licenses/by-sa/4.0/deed.de (visited on May 22, 2025)
[30] AutomationML (2018). „Best Practice Recommendation: Units in AutomationML”. URL: https://www.automationml.org/news/bpr-units-in-automationml-available/ (visited on May 27, 2025)
[31] German Engineering Federation (VDMA) (2020). "Guide to Virtual Commissioning - Recommendations for Economic Entry". In: VDMA Verlag GmbH. URL: https://vdma.eu/c/document_library/get_file?uuid=66118d8d-68fc-55a7-a25b-a90ffe89f1f2&groupId=34570 (visited on July 30, 2025)
[32] VDI Guideline 3693 (2025). Sheet 1: Virtual Commissioning - Model Types and Terms. Berlin: Beuth Verlag. URL: https://www.vdi.de/richtlinien/details/vdivde-3693-blatt-1-virtuelle-inbetriebnahme-modellarten-und-begriffe (visited on July 30, 2025)
[33] VDI guideline 3693 (2018). Sheet 2: Virtual Commissioning - Introduction of virtual commissioning in companies. Berlin: Beuth Verlag. URL: https://www.vdi.de/richtlinien/details/vdivde-3693-blatt-2-virtuelle-inbetriebnahme-einfuehrung-der-virtuellen-inbetriebnahme-in-unternehmen-1 (visited on July 30, 2025)
[34] Modelon (2025). „FMI Standard: Understand the two types of Functional Mock-up Units – CS v. ME”. URL: https://modelon.com/blog/fmi-functional-mock-up-unit-types/ (visited on July 31, 2025)
[35] Modelica Association (2025). „Tools that support FMI”. URL: https://fmi-standard.org/tools/ (visited on August 04, 2025)
[36] Industrial Digital Twin Association e.V. (2025). "IDTA – The Standard for the Digital Twin" URL: https://industrialdigitaltwin.org/ (visited on August 04, 2025)
[37] Federal Ministry for Economic Affairs and Energy (BMWE) (2020). „Specification: Submodel Templates of the Asset Administration Shell: Generic Frame for Technical Data for Industrial Equipment in Manufacturing (Version 1.1)) URL: https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2020/Dezember/Submodel_Templates_of_the_Asset_Administration_Shell/201117_I40_ZVEI_SG2_Submodel_Spec_ZVEI_Technical_Data_Version_1_1.pdf (visited on August 04, 2025)

[38] DIAMOND Project 2025. *„Leitfaden zur Anwendung des FMI-Standards im Kontext der virtuellen Inbetriebnahme. DIAMOND-Forschungsprojekt 2025“* URL: https://diamond-project.de/downloads/files/DIAMOND_Leitfaden_zur_Anwendung_des_FMI_Standards_im_Kontext_der_virtuellen_Inbetriebnahme.pdf (visited on November 19, 2025)

Authors

Franz Hammerl,	EDAG Production Solutions GmbH & Co. KG
Ali Moghaddam Nejad,	EDAG Production Solutions GmbH & Co. KG

Contributors

Thomas Zellmeier,	BMW Group
Franz Hammerl,	EDAG Production Solutions GmbH & Co. KG
Ali Moghaddam Nejad,	EDAG Production Solutions GmbH & Co. KG
Miriam Schleipen,	EKS InTec GmbH
Christian Wolf,	EKS InTec GmbH
Toni Krsticevic,	Festo SE & Co. KG
Walter Kuhlbusch,	Festo SE & Co. KG
Manuel Paul,	Festo SE & Co. KG
Christian Barth,	Festo SE & Co. KG
Matthias Freund,	Festo SE & Co. KG
Johannes Hoos,	Festo SE & Co. KG
Anuj Sambhare,	Festo SE & Co. KG
Viktor Ziegenhagel,	Mercedes Benz AG
Lorenz Hundt,	Murrelektronik GmbH
Francois Neveu,	Siemens AG
Christoph Wincheringer,	Siemens AG
Holger Wünsche,	WINMOD GmbH

The origin of this guide lies in the context of the DIAMOND research project. EDAG Production Solutions GmbH & Co. KG conducted neutral surveys with DIAMOND consortium partners on behalf of BMW AG, professionally consolidated the results and contributed its own content to the FMI guide.