

57th CIRP Conference on Manufacturing Systems 2024 (CMS 2024)

## Concept of Event-based AAS Transformation Engine

Benjamin Goetz<sup>a\*</sup>, Matthias Schneider<sup>a</sup>, Thomas Bauernhansl<sup>a,b</sup>

<sup>a</sup>Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Nobelstrasse 12, 70569 Stuttgart, Germany

<sup>b</sup>Institute of Industrial Manufacturing and Management, University of Stuttgart, Nobelstraße 12, 70569 Stuttgart, Germany

\* Corresponding author. Tel.: +49 711 970 1354. E-mail address: [benjamin.goetz@ipa.fraunhofer.de](mailto:benjamin.goetz@ipa.fraunhofer.de)

### Abstract

The Asset Administration Shell (AAS) is the backbone of the Industry 4.0 movement. It provides an implementation concept for the Digital Twin and is a solution for the challenges of interoperability on the shop floor. The principle of storing multiple data models - so-called submodels - describing a distinct aspect of an asset enables information exchange between hardware and software. To support the development, standardization, and maintenance of AAS submodels, this work presents a concept for an event-based AAS transformation engine. The vision of the engine is to create submodels based on properties of already existing submodels but with a different structure. The scope of this work is outlined by collecting abstract use cases and requirements regarding different viewpoints of the engine: model transformation in general, interaction with the AAS environment, and functions of the engine. Based on this work, a data model for transformation rules is developed. The data model focuses on how to address submodels in an AAS environment and how a submodel transformation can be described as a list of basic transformation actions. In order to provide horizontal scalability, a software architecture is developed based on the service-oriented design principle. Therefore, the model transformation task is split into basic sub-tasks. Each sub-task has a corresponding software service type that handles the specific task. As the architecture allows parallel read and write threads in an AAS environment, an access lock mechanism is introduced to avoid interference between the services. The data model and software architecture provide the foundation for an event-based transformation engine. For future work, the concept will be implemented and tested in a real-world use case to provide feedback for the next development iteration.

© 2024 The Authors. Published by Elsevier B.V.

This is an open access article under the CC BY-NC-ND license (<https://creativecommons.org/licenses/by-nc-nd/4.0>)

Peer-review under responsibility of the scientific committee of the 57th CIRP Conference on Manufacturing Systems 2024 (CMS 2024)

*Keywords:* Digital Twin; Asset Administration Shell; Engine; Transformation; Data Model;

### 1. Introduction

The digitalization of production is an ongoing process since the fourth industrial revolution has been announced and promoted with the term “Industry 4.0”. Among others, achieving interoperability between hardware and software on the shop floor is the main task for companies in the context of Industry 4.0 [1]. Interoperability is defined as “the ability for two or more systems or applications to exchange information and to mutually use the information that has been exchanged” [2]. As identified by recent publications, the main barrier to interoperability is the large number of legacy systems using proprietary data models [3]. In order to overcome this barrier,

each system shall provide a digital twin, which stores information about an asset. A digital twin may contain multiple data models describing a distinct aspect of the asset. By giving access to those data models via interfaces, it is possible to exchange information between assets and enable interoperability [4].

One concept for implementing the digital twin in Industry 4.0 is the Asset Administration Shell (AAS) developed by the German Platform Industry 4.0 and the Industrial Digital Twin Association (IDTA). While the specification and standardization of the meta model and the interface of the AAS are at a good level of development, the process of

standardizing data models – so-called submodel templates – is still in an early stage [5,6].

As there are already many projects focusing on the translation of non-AAS to AAS-compatible interfaces [3,7–10], this work presents the concept for a transformation engine that helps transform the structure of submodels and store the result in a new submodel instance. This is beneficial, especially during the early stage of developing new submodel templates, when submodel templates get revised and when companies use not-standardized and standardized models for assets at the same time.

The concept of an event-based AAS transformation engine presented in this paper gives an overview of requirements regarding the model transformation in general, the AAS infrastructure – the engine has to interact with – and the engine itself. Two use cases are identified based on the requirements, and a data model for transformation rules is developed. Finally, a service-oriented software architecture is presented that enables scalability depending on the load of the transformation engine.

The paper is structured as follows: Section 2 describes the main aspects of digital twins (DT) and the asset administration shell (AAS) as an implementation of the DT in the context of Industry 4.0. Section 3 describes the basics of model transformation in general and related work in the context of AAS Model Transformation. Section 4 shows the motivation of the work described in this paper and describes the requirements the AAS transformation engine has to fulfil. Section 5 shows the concept of the proposed transformation engine and is split up into Section 5.1, describing the main use cases, Section 5.2, describing the data model of transformation rules, and Section 5.3, describing the service-oriented architecture of the engine. Finally, Section 6 concludes the work and suggests tasks for further research.

## 2. State of the Art – Digital Twin and Asset Administration Shell

The definition of the Digital Twin (DT) has evolved over time. In the latest publications, DTs describe the historical and current states of assets and their properties in the form of information models [11, 12]. An asset is a physical or logical object that has value to an individual, an organization or a government. Examples of this definition of an asset are machines, raw materials, process definitions or software licenses [5]. An implementation based on the ISO 23247 standard of a DT of a manufacturing system is presented by Kibira et. al [13]. The motivation for introducing DTs in Industry 4.0 use cases is the ability to simulate/optimize production systems and the need for interoperability of hardware and software in different companies, industries and countries [14–16]. To fulfil this need, Standards in accessing DTs and for information models which are part of DTs are necessary [17].

Therefore, the Asset Administration Shell (AAS) was developed as an implementation concept of the DT by the Platform Industry 4.0. One AAS represents exactly one asset with a unique asset ID and enables the interoperability of those assets along the value chain regardless of the number of

companies and organizations involved [5,9]. Each AAS may contain multiple so-called submodels, which describe a specific aspect of an asset and correspond to the information models of DTs [10]. Each submodel may contain multiple properties with values of simple data types like integers, strings or dates. Those properties are stored in submodel elements. If properties require a list structure, the AAS standard provides submodel element lists for (un-)ordered lists or multidimensional arrays of (non-)unique values. In the case of complex data types, the AAS standard provides submodel element collections, which should have a fixed set of properties with unique names. The standard recommends that each property has a semantic definition represented by a SemanticID or a reference to a semantic definition, e.g. provided by ECLASS (see [18]) or IEC CDD (see [19]) [5].

In [6], the Industrial Digital Twin Association (IDTA) describes a fine-grained list of application programming interface (API) specifications. The APIs provide standardized access to the information stored in AAS and will solve the challenge of providing interoperability among (legacy) hardware and software from different vendors [8,10].

## 3. Literature Review - Information Model Transformation

In general, model transformation encompasses converting source information models into target models. Basic rules describe how elements from the source will be transformed into the target model. A transformation engine is a system that is able to understand those rules and carries out the transformation depending on the source models and a set of rules defined by a user or another system [10].

According to our research, there is already related work about model transformation in the context of AAS.

With [8,9], Platenius-Mohr et al. show a solution that bidirectionally converts information from proprietary digital twin (ABB Ability™ Digital Twin) to AAS. They evaluate the solution with a real-world use case, including motors and drives from ABB.

In comparison, Zielstorff et al. propose a generic strategy for integrating legacy systems into AAS environments. Their prototype utilizes the DataBridge [20] of the Eclipse BaSys project [21] to transform data from the OPC UA interface of an articulated robot into an AAS. The authors describe that their solution is also capable of integrating legacy systems by being compatible with other communication protocols other than OPC UA [7].

Behrendt et al. developed the mapping tool “aas2openapi” to convert DTs based on AAS into openAPI 3.0 objects and vice versa [22].

Miny et al. developed a model transformation language (MTL) called AASMTL. It is used to automatically generate new AAS Submodels using existing information from other submodels. In contrast to [8,9], [7] and [22], this solution is focused on model transformation within an AAS environment rather than translating models from and to AAS environments [10].

### 4. Motivation & Requirements

As described in Section 2, the AAS provides an interface to access asset information and to store data models about different aspects of an asset as submodels. While the standardization process of the AAS meta model (v3.0.7) [23] and the API (v.3.0.1) [24] is already well advanced, the process of standardizing structures of submodels as so-called submodel templates [5] has just started. The Industrial Digital Twin Association (IDTA), which takes care of the harmonization of submodels, lists 19 published templates and eight in development state [25]. It is expected that these numbers will rise slowly because the standardization of submodel templates is a complex and time-consuming process that requires the commitment of many companies across different domains and has to fit the different processes developed and machines used in the companies over decades. Furthermore, submodel templates will evolve over time, as shown by the “Digital Nameplate”, one of the first published submodel templates. It has been revised once [5,25]. Additionally, Zielstorff et al. distinguish between intra-company and standardized submodel templates as they expect companies to run proprietary submodel templates for internal processes while using standardized templates for data exchange between companies.

This brings us to the conclusion that a tool is required that supports companies:

- to develop new submodel templates,
- to derive submodels from older submodels in case the submodule template gets revised,
- to run intra-company and standardized submodels in parallel that may have an intersection, and
- to enable generic products to provide multiple domain-specific submodules depending on their current operation area

With this paper, we provide a concept for an AAS transformation engine capable of creating new submodels (target) based on the information from already available submodels (source) in a company’s network.

For collecting the requirements regarding the AAS transformation, the categories and characteristics summarized in [10] have been used:

Table 1. List of Transformation Requirements

No.	Transformation Characteristics	Requirement description
R1	direction of transformation	unidirectional
R2	incrementality [26]	update target based on source; user edits are not preserved in target model
R3	type of source/target	Both submodels in same or different AAS
R4	number of sources/targets	1:1 (one source AAS, one target AAS) M:1 (multiple source SM, one target SM)
R5	change of abstraction level [27]	horizontal and vertical depending on rule set by user
R6	relationship source / target [27]	out-place transformation
R7	type of source / target meta models	endogenous transformation; keeping the same language

R8	transformation paradigm [27]	imperative
R9	execution conditions	always when source submodel is modified
R10	parameterization	no support for parameters of transformation rules
R11	scheduling [26]	no support for scheduling; rules are executed when sources are changed (see R10 “execution conditions”)
R12	rule selection	rule to be executed is determined by detected changes in source models
R13	rule iteration	rule is executed once per change of source model
R14	phasing	rules are not executed in phases
R15	modularization	rules consist of reusable transformation actions
R16	reuse of rules	no reusability of rules

As mentioned in R10 and R12 (see Table 1), the transformation occurs when a transformation rule’s source submodel is changed. The notification about every change shall be published via an event bus (compare with “data integration via message bus” [3]). This adds additional requirements to services, which have AAS-compatible interfaces (from now on referred to as *AAS services*):

Table 2. List of AAS infrastructure requirements

No.	Requirement description
R17	AAS services are communicating every change via message bus / broker
R18	AAS services are capable of communicating what kind of change has been made (create, modify, delete)
R19	AAS services are capable of communicating the new value of the submodel / property in case of create and modify action
R20	AAS services are using standardized format based on AAS meta model for communicating changes

Finally, there are requirements regarding the implementation of the transformation engine that are derived from Table 1 and Table 2:

Table 3. List of AAS Transformation Engine requirements:

No.	Requirement description
R21	engine has an API to create, modify and delete transformation rules during runtime
R22	engine is capable of handling multiple transformation rules
R23	engine triggers transformation rule based on changes in properties of source submodels, only (no support for Operations in Submodels)
R24	engine follows service-oriented design principal for horizontal scalability
R25	engine has to be able to write target submodel into source AAS or create / use separate target AAS
R26	engine has to be able to detect and prevent cyclic dependencies between transformation rules
R27	engine has to have the ability to write target AAS / submodels via API of external AAS service
R28	engine has to have write permission for API of external AAS service to write target AAS / submodel

### 5. Concept

#### 5.1. Use Cases

In the next step, the requirements are used to derive the concept of the AAS transformation engine. According to R3 the input and the output type of a transformation is a submodel. R4 describes that there is always one source and one target AAS (1:1) while there may be multiple source submodels as inputs for a transformation rule and one submodel as output (M:1). Based on those requirements, two main use cases have been identified. The first use case shows one AAS as source and target. The second use case has one AAS for the source and one for the target (see Fig. 1). Both use cases show unidirectional transformation as defined in R1.

#### 5.2. Data Model for Transformation Rule

According to the use cases described in the previous section, transformation rules require a destination definition (see Fig. 2) for the target submodel and the target AAS in case the source AAS should not be the target (see Fig. 1). In order to define a target AAS the meta model of the Asset Administration Shell specifies a mandatory, globally unique “id” attribute for identifying AAS. The recommended value format of the attribute is an Internationalized Resource Identifier IRI (or URI/URL) [5]. To resolve the Identifier (ID) of an AAS, an application needs the address of a registry service with the AAS Registry Interface, according to [6], implemented. With the AAS endpoint received by the AAS registry, the information about the AAS may be retrieved. That information contains a list of submodel IDs of the submodels held by the AAS. According to [5], a submodel has a mandatory, globally unique id attribute in the format of IRI (or URI/URL) or a Custom format. An “idShort” (format: string) and a “semanticId” (format: IRDI, IRI) are recommended additional, globally unique attributes for submodels. With one of those submodel IDs, it is possible to retrieve the endpoint of a submodel instance from a submodel registry service that provides a Submodel Registry Interface [6]. The endpoint of the submodel provides the ability to

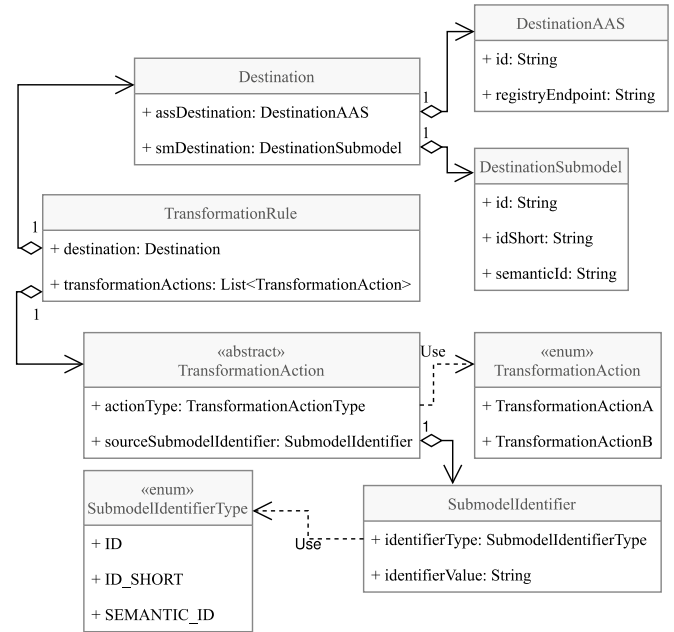


Fig. 2. Data format for Destination of Transformation Rule

create, get, modify or delete a submodel and its submodel elements.

In order to fulfil the imperative transformation paradigm mentioned in R8 (see Table 1), every transformation rule has an ordered list of transformation actions (see Fig. 3). When the rule is triggered, the actions are executed in sequence according to their position on the list. A transformation action is an abstract class containing a unique action type among all implementations and an ID for the source submodel, on which the action is applied. Since the transformation engine is provided as an open-source project [28], users can add their desired implementation of actions on their own. For example, a simple copy action is provided by the project that copies a submodel element from the source to the target submodel.

#### 5.3. Software Architecture

The software design of the transformation engine follows the principle of a service-oriented architecture<sup>1</sup>. Therefore, the

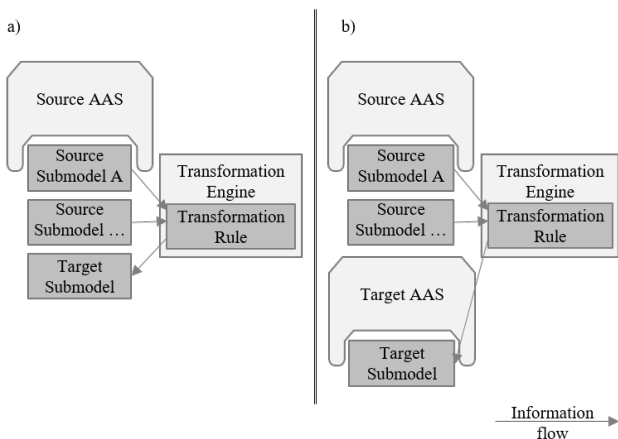


Fig. 1. Use Cases based on R25: (a) source and target AAS are the same; (b) source and target AAS are different

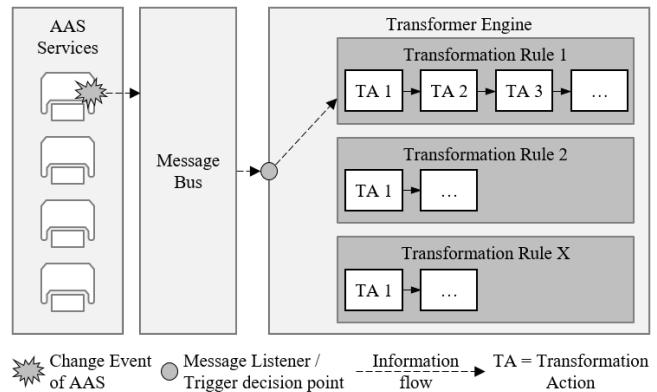


Fig. 3. Communication Principal between AAS Services and Engine (left to right); Action-based Transformation Rules as ordered list (right)

<sup>1</sup> “Service-oriented architecture (SOA) is a type of software design that makes software components reusable using service interfaces that use a common communication language over a network” [29].

task of transforming submodels is split into sub-tasks following the divide-and-conquer principle<sup>2</sup>. A related microservice<sup>3</sup> type is responsible for each sub-task. All services communicate over the network, forming the transformation engine together. By following this principle, it is possible to add instances of one microservice type in case the demand for its related sub-task rises. This enables horizontal scalability as required by R24.

To address R21, a microservice called *rule management* provides an API that enables users and other systems to create, modify and delete transformation rules. The rule management has access to an internal rule database where transformation rules are persisted. The database system allows the storage of multiple transformation rules at the same time (see R22) and the modification of those rules during runtime. To prevent cyclic dependencies / closed loop transformations, as shown in Fig. 4, that might lead to an infinite loop of transformations (R26), the rule management has to do an integrity check. A new rule has to be refused if its source submodels are target submodels of existing rules and the target of the new rule is a source of existing rules.

According to Fig. 3, the transformation engine has to listen to changes in the AAS structure, which the *message bus listener* does. The listener receives standardized messages about AAS changes, containing which AAS / submodel instance has been created or deleted or which part of an AAS / submodel has been modified (see R20). Based on that information, the listener must detect if a rule has to be triggered and if the target submodel has to be created/deleted or just updated incrementally. The information on which rules have to be run and what kind of change (create, modify, delete) has to be done is written into a transformation job and sent to a *transformation job queue*.

The queue is a buffer for transformation jobs coming from the message bus listener and has to be able to forward those jobs in a first-in-first-out manner. This means that the oldest job in the queue is always the next to be processed by the transformation engine.

The transformation jobs are executed by *rule executor*

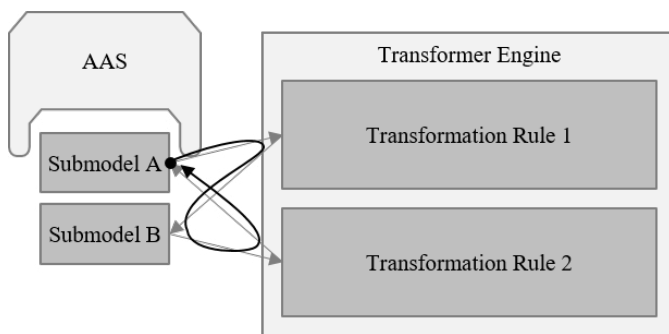


Fig. 4. Forbidden closed loop transformation of two rules

<sup>2</sup> “A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly” [30].

<sup>3</sup> “the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms” [31].

services. They listen to jobs saved in the job queue and process the transformation actions of a rule depending on the kind of change made to the source submodel.

If multiple rule executors run in parallel, it is necessary to lock access to source and target submodels as long as an executor interacts with them. This avoids interference between the executors and guarantees the first-in-first-out work order on source/target submodel combinations in case old jobs take longer than new ones. Therefore, the executors acquire a lock on the relevant submodels at the start of a job and release the lock when the job is finished. In case a second executor tries to acquire the lock of submodels that are already locked, it has to wait until the lock is released. The locks are managed by a microservice called *AAS access lock management*, and all executors have to get the permission from that service to run a transformation rule.

With the separation of listening to changes in the AAS structure (message bus listener service) and the execution of transformation rules (rule executor service), it is possible to do a fine-grained scaling of the transformation engine. Depending on the expected number of changes in the AAS structure, running a low or high number of listener instances is possible. The same applies to the number of transformation jobs in the job queue and the corresponding number of rule execution services. With online monitoring of the load of listener/executor services, it is possible to adjust the number of microservices based on the demand automatically. For an overview of the proposed architecture, see Fig. 5.

## 6. Conclusion and Future Work

While AAS meta model and API standardization are well advanced, the process of standardizing submodel templates is still at the beginning and expected to be a slow and time-consuming process. As described in section 3, previous work has focused mainly on translating non-AAS data sources into AAS-compatible ones. This work aimed to provide a concept for a transformation engine capable of transforming existing AAS submodels into new submodels with a different structure.

The first step was to collect requirements regarding model transformation in general, the AAS infrastructure and the transformation engine’s implementation.

Based on the requirements, two main use cases were identified writing the target submodel of a transformation into the source AAS or into a different target AAS that the engine may create at the beginning of the transformation. In order to

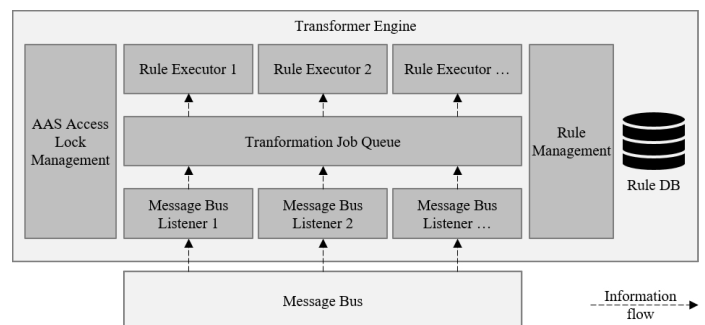


Fig. 5. Software architecture of transformation engine

meet the requirements, a data model for transformation rules was developed that defines how the target of a transformation is set and that a transformation is an ordered list of basic transformation actions processed when the rule gets triggered. An action is defined as an abstract class providing a basis for other developers to contribute implementations of actions as the engine is set up as an open source project.

Finally, a software design inspired by the concept of service-oriented architecture is proposed. Therefore, the overall task of the engine was split up into sub-tasks and corresponding microservice types were derived. This approach enables horizontal scalability to scale the engine depending on its demand.

For future work, AAS implementations will be looked for, evaluated and selected to be used for implementing the transformation engine. The described concept will be implemented in the context of real-world use cases. Based on the requirements and required data models of the use case, first transformation action types will be developed and the overall concept of the engine will be evaluated. The first implementation will be re-evaluated in more use cases and the feedback will be used to refine the concept and the implementation. In order to test the performance of the architecture, stress scenarios and performance indicators will be defined to find the capabilities and limits.

## Acknowledgements

This paper is based on the work in the research projects FabOS and H2Giga-FRHY funded by the Federal Ministry for Economic Affairs and Climate Action (BMWK).

## References

- [1] M. Mamad, Challenges and Benefits of Industry 4.0: An overview, in: *International Journal of Supply and Operations Management*, pp. 256–265.
- [2] ISO/IEC Organization, Internet of things (IoT) – Interoperability for iot systems – Part 1: Framework. <https://www.iso.org/standard/71885.html>.
- [3] F. Schnicke, A. Haque, T. Kuhn, D. Espen, P.O. Antonino, Architecture Blueprints to Enable Scalable Vertical Integration of Assets with Digital Twins, in: *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Stuttgart, Germany, IEEE, 2022, pp. 1–8.
- [4] X. Ye, S.H. Hong, W.S. Song, Y.C. Kim, X. Zhang, An Industry 4.0 Asset Administration Shell-Enabled Digital Solution for Robot-Based Manufacturing Systems, *IEEE Access* 9 (2021) 154448–154459. <https://doi.org/10.1109/ACCESS.2021.3128580>.
- [5] Industrial Digital Twin Association, Part 1: Metamodel, 2023.
- [6] Industrial Digital Twin Association, Part 2: Application Programming Interfaces, 2023.
- [7] A. Zielstorff, D. Schöttke, A. Hohenhövel, T. Kämpfe, S. Schäfer, F. Schnicke, Harmonizing Heterogeneity: A Novel Architecture for Legacy System Integration with Digital Twins in Industry 4.0, in: S. Terzi, K. Madani, O. Gusikhin, H. Panetto (Eds.), *Innovative Intelligent Industrial Production and Logistics*, Springer Nature Switzerland, Cham, 2023, pp. 68–87.
- [8] M. Platenius-Mohr, S. Malakuti, S. Grüner, T. Goldschmidt, Interoperable Digital Twins in IIoT Systems by Transformation of Information Models, in: *Proceedings of the 9th International Conference on the Internet of Things*, Bilbao Spain, ACM, New York, NY, USA, 10222019, pp. 1–8.
- [9] M. Platenius-Mohr, S. Malakuti, S. Grüner, J. Schmitt, T. Goldschmidt, File- and API-based interoperability of digital twins by model transformation: An IIoT case study using asset administration shell, *Future Generation Computer Systems* 113 (2020) 94–105. <https://doi.org/10.1016/j.future.2020.07.004>.
- [10] T. Miny, M. Thies, U. Epple, C. Diedrich, Model Transformation for Asset Administration Shells, in: *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, Singapore, Singapore, IEEE, 10182020, pp. 2207–2212.
- [11] S. Malakuti, J. Schlake, C. Ganz, K.E. Harper, H. Petersen, *Digital Twin: An Enabler for New Business Models*, 2019.
- [12] B. Schleich, N. Anwer, L. Mathieu, S. Wartzack, Shaping the digital twin for design and production engineering, *CIRP Annals* 66 (2017) 141–144. <https://doi.org/10.1016/j.cirp.2017.04.040>.
- [13] D. Kibira, G. Shao, R. Venketesh (Eds.), *Building a digital twin of an automated robot workcell*, Annual Modeling and Simulation Conference (ANNSIM), Hamilton, CA, 2023.
- [14] W. Kritzinger, M. Karner, G. Traar, J. Henjes, W. Sihn, Digital Twin in manufacturing: A categorical literature review and classification, *IFAC-PapersOnLine* 51 (2018) 1016–1022. <https://doi.org/10.1016/j.ifacol.2018.08.474>.
- [15] S.-W. Lin, K. Watson, G. Shao, L. Stojanovic, B. Zarkout, Digital Twin Core Conceptual Models and Services, An IIC Technical Report, 2023. [https://tsapps.nist.gov/publication/get\\_pdf.cfm?pub\\_id=935370](https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=935370).
- [16] Plattform Industrie 4.0, The Asset Administration Shell: Implementing Digital Twins for use in Industrie 4.0: A starter kit for developers, 2019.
- [17] DIN SPEC 91345:2016-04, Referenzarchitekturmodell Industrie 4.0 (RAMI4.0), Beuth Verlag GmbH, Berlin.
- [18] ECLASS e.V., ECLASS data standard for products & services. <https://eclass.eu/>.
- [19] International Electrotechnical Commission, IEC - Common Data Dictionary (CDD). <https://cdd.iec.ch>.
- [20] Eclipse Foundation, Eclipse BaSyx DataBridge. <https://github.com/eclipse-basyx/basyx-databridge>.
- [21] Eclipse Foundation, Eclipse BaSyx. <https://eclipse.dev/basyx/>.
- [22] S. Behrendt, F. Stamer, M.C. May, G. Lanza, Towards a Service-Oriented Architecture for Production Planning and Control: A Comprehensive Review and Novel Approach, *publish-Ing*, 2023.
- [23] Industrial Digital Twin Association, Repository of the Asset Administration Shell Specification IDTA-01001 - Metamodel. <https://github.com/admin-shell-io/aas-specs>.
- [24] Industrial Digital Twin Association, Repository of the Asset Administration Shell - API. <https://github.com/admin-shell-io/aas-specs-api>.
- [25] Industrial Digital Twin Association, Submodel Templates for AAS. <https://github.com/admin-shell-io/submodel-templates> (accessed 1 March 2024).
- [26] K. Czarnecki, S. Helsen, Feature-based survey of model transformation approaches, *IBM Syst. J.* 45 (2006) 621–645. <https://doi.org/10.1147/sj.453.0621>.
- [27] T. Mens, P. van Gorp, A Taxonomy of Model Transformation, *Electronic Notes in Theoretical Computer Science* 152 (2006) 125–142. <https://doi.org/10.1016/j.entcs.2005.10.021>.
- [28] B. Götz, M. Schneider, FabOS-AI/aas-transformer. <https://github.com/FabOS-AI/aas-transformer>.
- [29] Red Hat, Inc., What is service-oriented architecture (SOA)?, 2020. <https://www.redhat.com/en/topics/cloud-native-apps/what-is-service-oriented-architecture>.
- [30] Wikipedia, Divide-and-conquer algorithm. [https://en.wikipedia.org/wiki/Divide-and-conquer\\_algorithm](https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm).
- [31] M. Fowler, J. Lewis, Microservices. <https://martinfowler.com/articles/microservices.html>.