Papyrus4Manufacturing: A model-based systems engineering approach to AAS digital twins

Saadia Dhouib, Yining Huang, Asma Smaoui, Tapanta Bhanja, Volkan Gezer

Papyrus4Manufacturing: A Model-Based Systems Engineering approach to AAS Digital Twins

Saadia Dhouib*, Yining Huang*, Asma Smaoui*, Tapanta Bhanja†, and Volkan Gezer†
*Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
†Innovative Factory Systems, German Research Center for Artificial Intelligence (DFKI), 67660, Kaiserslautern, Germany
Email: {yining.huang, saadia.dhouib, asma.smaoui}@cea.fr
{tapanta.bhanja, volkan.gezer}@dfki.de

***Abstract*—As digital twins gain momentum in their usage in diverse domains, the concept of Asset Administration Shells (AAS) has become very relevant for achieving the digital twin approach, where Administration Shells are the digital representation of physical assets. Being a relatively new concept in the Industrial Internet of Things (IIoT) domain, the tools and approaches for creating and deploying AASs are likewise in infancy. This paper introduces an open-source tool, *Papyrus4Manufacturing*, which provides a model-based systems engineering approach to the AAS. This toolset supports the creation of AAS digital twins from modeling to automatic deployment and connection to assets using the OPC UA protocol. This paper also includes an evaluation of its usability, as it is put to test with an academic use case.**

***Index Terms*—Digital Twins, Asset Administration Shell, Model-Based System Engineering, Unified Modelling Language, UML Profiles, Generative Software Engineering, OPC UA, BaSyx, Eclipse Papyrus**

I. Introduction

Digital Twins (DT), as defined by [1], are high-fidelity virtual models of physical objects in virtual space in order to stimulate their behaviors in the real world and provide feedback. DT has and will become part and parcel of the Industrial Internet of Things. With the industrial entities gaining computational abilities, connecting the surrounding physical world and the ongoing processes in it, exchanging information, and processing data [2], a need for a common language that describes the system DT with a clear projection of the system architecture, representing the whole system coherently at various levels of abstraction, while avoiding information silos, has become imminent [3]. This standardization is driven by the concept of Asset Administration Shell (AAS) [4], which defines the meta-model for modeling such digital twins of physical assets or cyber-physical systems.

Model-Based Systems Engineering (MBSE) - is a formalized methodology that puts models at the central focus of system design and is used to support the requirements, design, analysis, verification, and validation associated with the development of complex systems that are increasingly adaptable to the digital-modeling environment [5]. The term *system* is defined here, according to Walden et al. [6], as an integrated set of elements, subsystems, or assemblies that accomplish a defined objective. The system could be diverse ranging from a complex chain of interconnected independent machine and machine tools to a single hardware or a software component being coherent to the definition of *asset* defined in [4]. According to [7], to fight the ever increasing complexity of embedded devices, embedded software development has been through a trend shift from manual to model-driven development (MDD), just like high-level programming languages have almost replaced assembly language. However generic model-driven development languages like UML, is an abstract modeling language, built to fit all domains and needs. A need for refinement of UML arises when considering a specific domain, which in the case of this paper are DTs for the manufacturing industry.

The toolset presented in this paper, Papyrus4Manufacturing (P4M), extends the UML Modeling Tool Papyrus, to meet the standards of the AAS. In addition to the AAS models creation, P4M automates the deployment of these models. The code generation of the digital twins from the AAS models permits the establishment of communication between the digital twin and its physical counterpart.

This article is organized as follows. Section II presents the related work. Section III briefly explains the architecture of the developed tool P4M followed by an introduction to the AAS-based modeling environment in Section IV. The BaSyx code generation and deployment of this tool are covered in Section V. Section VI presents an academic use case where the tool was tested and iteratively developed with. The article is concluded with a discussion and conclusion about the work and its further development scope.

II. Related Works

The drive towards the Industry 4.0 and the novel concept of the Asset Administration Shell have invited professionals across industries and the scientific community to put efforts towards realizing the concepts at the operative level. These efforts have led to standardization activities like - the development of the RAMI (Reference Architecture Model Industrie 4.0) [8] and the Details of the Asset Administration Shell [4], which helps in providing the cornerstones while developing the AASs for representing any considered ecosystem of Cyber-Physical Systems (CPS). The advancements are not just lim-

The work has received funding from the European Union's Horizon 2020 program via Project GA Nr. 952003 AI REGIO and from the European Institute of Innovation and Technology (EIT), a body of the European Union, under the Horizon 2020 program, via the CanvAAS project.

ited to the reference architectures and standards but are also seen in the various implementations of the concept of AAS. These projects include not only the development of Software Development Kits (SDKs) such as BaSyx [9], but also some implementation efforts like *Fraunhofer Advanced AAS Tools (FA^{3}ST)* [10], *AAS Package Explorer* [11], *SAP I4.0 AAS* [12] and *NOVAAS* [13].

Eclipse BaSyx provides an execution infrastructure for AAS models, but it doesn't support a ready-to-use HMI (Human-Machine Interface) tool for non-tech-savy users. Instead, it provides software development kits in JAVA, .Net Core, and C++ [14]. The SDKs act as the basis for creating applications where information is modeled and transferred using the standards of the AAS.

FA^{3}ST [10] is a java-coded service-oriented tool. It includes a predefined implementation for HTTP and OPC UA-based protocols endpoint, JSON serializer and deserializer, file and database-backed persistence manager, as well as MQTT and OPC UA-based asset connections. A FA^{3}ST service can be deployed either as a Java JAR file or as a Docker container. Compared to BaSyx, FA^{3}ST provides more features, such as the integration with Apache StreamPipes (a toolbox for Industrial IoT with a focus on stream processing) as well as with the international data spaces (IDS).

The AASX Package Explorer (AASX PE) is a user-friendly application with a GUI which provides working tools and components for the creation of AAS models based on the specification [4] [11]. As AASX PE is a tool most used today for the specification of AAS models, many middelwares offer the possibility of being configured with .aasx files for execution.

Both the FA^{3}ST service and AASX Package Explorer offer an HTTP and an OPC UA-based service endpoint; however, in the AASX Package Explorer, they are not synchronized, meaning that changes to the DT via one type of endpoint are not reflected in the other. The AASX PE provides some functionality beyond the specification and is not implemented by FA3ST Service, such as HTTPS/SSL MQTT endpoint and a graphical user interface. However, connecting the DT to existing assets is limited to OPC UA.

On the other hand, the SAP I4.0 AAS, implemented in JavaScript, TypeScript, and Go, offers a GUI for describing an asset as per the standard of AAS. NOVAAS [13] is a Node-RED-based implementation of the AAS specification [14]. It has a strong focus on JSON, HTTP, MQTT, and usability, e.g., it provides user management and a dashboard to visualize live data. Still, it does not address essential parts of the specification, such as different data formats or OPC UA. Moreover, NOVAAS is realized using only Node-RED so it is less capable to integrate with other systems.

An aspect that lacks in all of the above-mentioned technology tools for the creation of the AAS is the model-driven approach. In light of the benefits that model-driven development approaches bring, as mentioned in Section I, it is apparent that a model-driven development approach to the AAS is also necessary. The tool Papyrus4Manufacturing [15] provides exactly the needed MBSE approach to AAS. P4M was developed on top of the UML-Modeling tool, which provides features restricted to the standard - Details of the Asset Administration Shell, in order to provide a graphical modeling environment that helps to model an AAS.

TABLE I
TOOLS IMPLEMENTING AAS STANDARD

Tool	HMI	MBSE	Asset Connection	AAS Execution
AASX PE	Yes	No	OPC UA	Yes (with different middlewares)
NO-VAAS	Yes (web based)	No	OPC UA/ HTTP/ MQTT	Yes
SAP	Yes (web based)	No	No	Yes
FA^{3}ST	No	No	OPC UA/ HTTP/ MQTT	Yes
P4M	Yes	Yes	OPC UA/ HTTP/ MQTT/ WebSocket/ ROS	Yes (automatic code gen to BaSyx)

Moreover, while Roth and Rumpe (2015) [16] mention that code generators are an integral part of any model-driven development process, Höllder et. al. [17] say that constructive generation or synthesis of code from the models needs to be among the first steps of a model-based development process implying the necessity of code generation. Thus, during the development of P4M, the feature of code generation from the models developed using the UML-based modeling tool was taken into account. The BaSyx Java SDK [9], an open-source middleware for the implementation of AAS, was used as the code generation target.

Table I illustrates a comparison of the previous related works, i.e. tools implementing the AAS Standards. We can see that all the tools are implementing the AAS standard and propose an execution of the AAS model by manually creating the software code. However, only our tool is (1) providing the automatic generation of the executable code from the AAS models and (2) ensuring the synchronization between the AAS models and the executable code.

III. PAPYRUS4MANUFACTURING ARCHITECTURE

P4M is a novel open-source toolset for digital twins modeling and deployment, released with an open-source license in the Eclipse platform [15]. It is fully compliant with the AAS specification and provides automatic integration with BaSyx, the open-source middleware for Industry 4.0 applications. P4M is based on two open-source tools Papyrus [18] and BaSyx [9]: Papyrus for the AAS graphical modeling interface and BaSyx for the AAS execution infrastructure as shown in Fig. 1. The toolset implements the AAS meta-model published in the AAS specification [4], which offers a graphical modeling

environment for designing AAS-compliant digital twins and enables their code generation to the BaSyx middleware, then their deployment to an AAS HTTP server. AAS digital twins running in the AAS server are made accessible through a REST API, thus any data analytic application or dashboard can connect to the AAS server and request data from the Administration Shell (digital twin) of a physical asset. As shown in Fig. 1, owing to the use of BaSyx as the middleware, P4M is able to offer an OPC UA Adapter which enables the establishment of communication with physical assets over an OPC UA communication protocol. It is important to mention that any number of OPC UA connections can be initialized which implies that there is no limit to the amount of physical assets being connected to the AAS as long as they communicate using an OPC UA communication protocol. The toolset is also interoperable with other environments supporting the AAS specification. Indeed, it offers a data exchange functionality using the AASX package format or JSON.

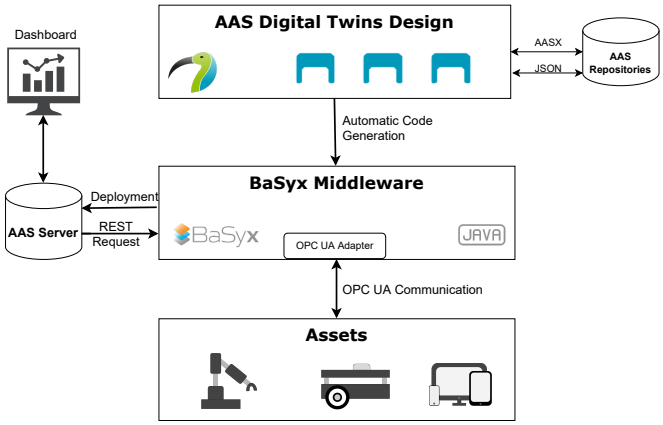

Fig. 1. Papyrus4Manufacturing Architecture

IV. AAS-based Modeling Environment

The AAS-based modeling environment was developed as an ISO/IEC/IEEE 42010 compliant architecture framework [19] (Fig.2). It provides several modeling editors in order to create multiple views for the description of AAS-based digital twins architectures. The modeling environment is embedding a domain-specific modeling language (DSML) that governs all the viewpoints of the architecture framework. The DSML is a UML profile that implements the AAS meta-model (version 3RC1). We have chosen the UML profile mechanism since the AAS-based modeling environment is implemented as an extension of Papyrus [18] which is an open-source model-driven workbench supporting the OMG modeling language standards: UML, SysML, and BPMN. Extending an existing modeling environment is a well-suited approach for developing DSMLs in an iterative way, taking advantage of the already existing modeling diagrams and consequently avoiding developing the environment from scratch.

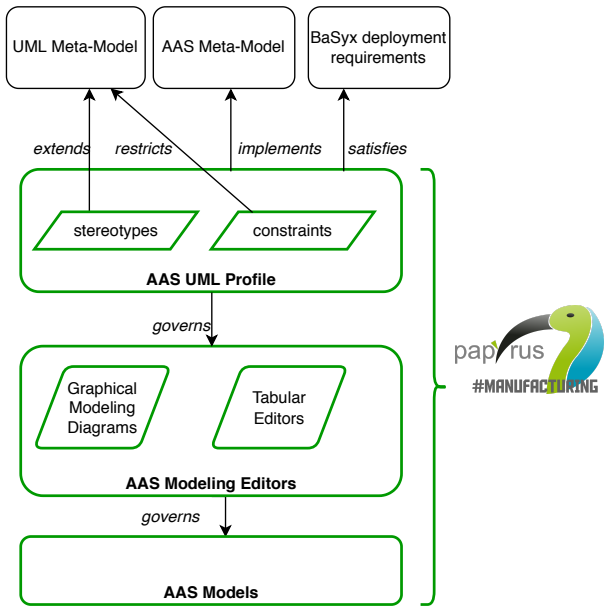

Fig. 2. Papyrus4Manufacturing Modelling Environment

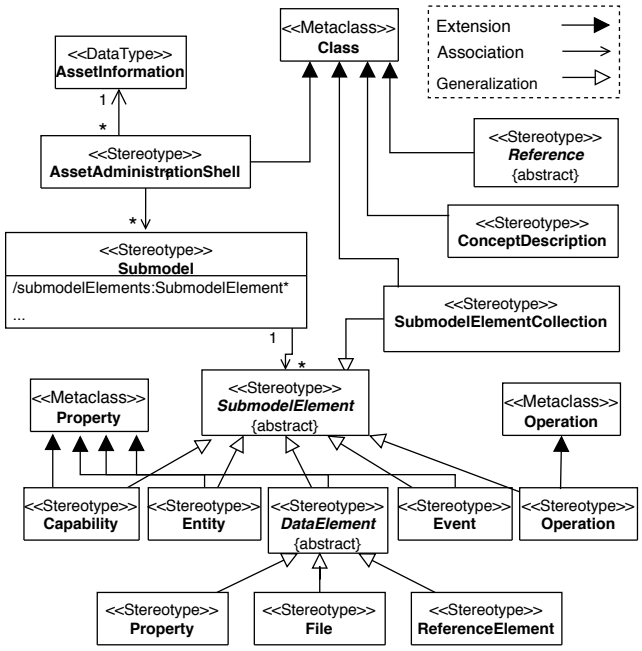

Fig. 3. Excerpt from the AAS UML profile

A. AAS UML Profile

First, we implemented the AAS meta-model as a UML profile in Papyrus. Figure 3 shows an excerpt from the UML profile, the whole model is available in the git repository of P4M[1]. UML profiles are a straightforward mechanism for extending the UML meta-model with concepts that are specific to a particular domain. The primary extension construct in

[1]https://git.eclipse.org/c/papyrus/org.eclipse.papyrus-manufacturing.git/tree/aas/plugins/modeling/org.eclipse.papyrus.aas.profile/resources

a profile is the Stereotype. We defined a mapping between the AAS meta-model and the UML meta-model based on the semantics of each meta-model construct, the semantics must be compliant.

- AssetAdministrationShell, Asset, Submodel, SubmodelElementCollection, Reference, ConceptDescription stereotypes extend the UML meta-class Class since each of the semantics of these AAS concepts are compliant with the semantics of Class: "The purpose of a Class is to specify a classification of objects and to specify the Features that characterize the structure and behavior of those objects" [20]. In order to restrict the semantics of UML meta-class Class, we define constraints attached to each stereotype. For example, an Asset must not contain attributes, operations, and behaviors.
- Each SubmodelElement extends a specific meta-class depending on its semantics. For example, the "Operation" stereotype extends the UML meta-class Operation with the restriction: the return parameter of the UML operation is not considered since AAS operations support only in, out, and inout parameters.
- Capabilities and Skills are specializations of SubmodelElements. Capabilities are represented using the "Capability" stereotype that extends the Property meta-class from UML with the constraint that the Capability does not have a type. Skills are represented using the "Operation" stereotype that extends the Operation meta-class from UML. In capability-based engineering captured from [21] and [22], the Capability concept (a type of AAS SubmodelElement) is an abstract description of the functionality of a production resource while the skill concept is the asset-dependent implementation to achieve a certain effect. The main goal of capability-based engineering is to design, implement and then dynamically operate the system according to the functions required in each step of the production process, rather than explicitly specifying the actual production resources. In article [23], we presented the possibility of Capability and Skills modeling in P4M.

B. AAS UML Profile Constraints

Besides the UML stereotypes presented in the previous section, we developed Java EMF constraints[2] that restrict the usage of UML meta-classes extent by the stereotypes. These Java constraints implement the semantics of the AAS concepts. Examples of constraints are listed below. All the Java constraints are available in the AAS validation plugin[3].

- Constraint "AAS must contain only submodels": all the UML nestedClassifiers of the class stereotyped with "AAS" must be stereotyped with "Submodel"
- Constraint "A submodel contains only submodelElements": all the UML ownedElements (attributes, operations and nestedClassifiers) of a submodel UML class must be stereotyped with a stereotype that inherits from the SubmodelElement stereotype.
- Constraint "an Asset must contain neither attributes, nor operations, nor sub-classes"
- Constraint "an Identifiable must have an identifier": the identification attribute of the stereotype Identifiable must have a value.
- Constraint "an AAS must reference an asset" : the assetInformation attribute of the AAS stereotype must have a value.

Constraints are implemented as model validation rules that can be executed manually by the AAS designer (batch mode) or automatically triggered by the modeling tool (live mode). Model validation is mandatory before automatic code generation, it prevents generating erroneous code.

C. Semantic Annotation of AAS Models

Although the AAS standard provides syntactic interoperability for assets cross-vendor, the semantic interoperability problem remains. Ontologies define semantic models of data combined with relevant domain knowledge and formulate inference strategies. In P4M, we covered the semantic annotation of standardized AAS data models with domain-specific ontologies and reasoning features, which bring together the power of modeling and ontology to create the ultimate combination. In article [24] we addressed an approach, which provides the semantic meanings defined in a specific ontology (MaRCO ontology [25]) to digital twin AAS models with a concrete example. In addition, this method is not only applicable to MaRCO ontology but also to any other ontology. The reason we choose MaRCO is that it provides a good foundation for capability matchmaking in a capability-based engineering approach.

D. AAS Modeling Editors

The AAS graphical modeling editors consist of:

- a diagram for creating AASs and Submodels. This diagram is an extension of the UML Class diagram, it hides all the UML terminology and exposes only the AAS-specific concepts.
- a diagram for creating a bill of material submodel. This diagram is an extension of UML composite structure diagrams.
- a diagram for creating BPMN processes diagrams. This diagram is an extension of UML activity diagrams and exposes a subset of the BPMN [26] language. It is the basis for modeling production and assembly processes inside an AAS Submodel.
- tabular editors for SubmodelElements creation and bill of material creation. In order to integrate data from external sources, the tabular editors support importing/exporting data from/to MS Excel sources.

All the details of the editors are shown in the P4M [15] documentation.

[2]https://www.eclipse.org/emf-validation/

[3]https://git.eclipse.org/c/papyrus/org.eclipse.papyrus-manufacturing.git/tree/aas/plugins/modeling/org.eclipse.papyrus.aas.validation

V. Code generation and deployment

As discussed in Section II, code generators are an integral part of model-driven development, which allows an automatic and iterative transformation of abstract models to concrete code. The code generator is defined as a deterministic and interactive system that terminates creating an implementation (code) in a programming language from a set of input artifacts, where the artifacts are typically models. Papyrus4Manufacturing perfectly fits the definition above. The developed model-based systems engineering tool does in fact provide an iterative development of AASs, from the UML-based models converting them into deployable implementation codes.

Besides, the code generators generally consist of a front-end part, *language processing* and a back-end part, *code generation*. The front-end part receives information in an input language and processes it using the *language processor*, while the back-end part maps this information to the target language using the *code generator*. P4M follows a similar principle. UML models which represent the AAS of assets being modeled are the set of input artifacts that are converted to the target language Java. The developed AAS-based modeling environment in Papyrus is the *language processor* here, while the BaSyx code generator takes the role of *code generation*.

A. BaSyx Code Generation

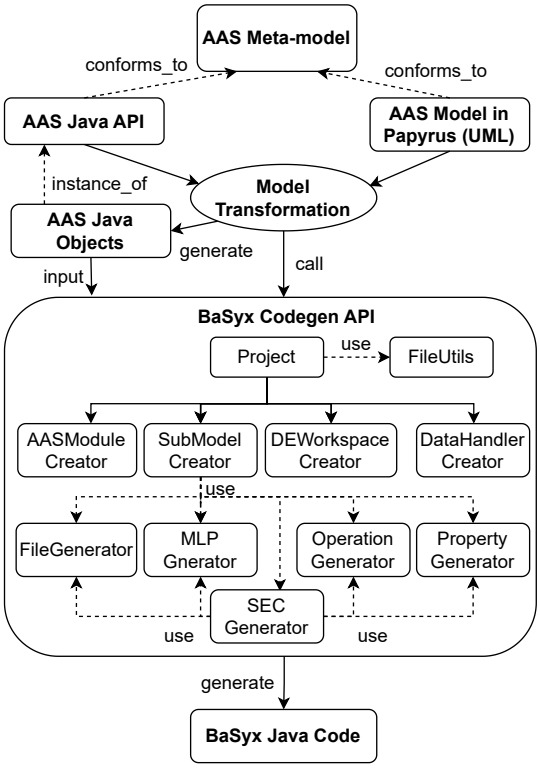

Fig. 4. Architectural Overview of Code Generator

Figure 4, shows an overview of the code generator architecture P4M uses in translating the input model and its artifacts to the code generator using the BaSyx SDK [9] as a middleware. In the background, when a model is created in P4M, it effectively generates *AAS Java Objects* by transforming UML/AAS model elements to *AAS Java API* objects [27]. The *BaSyx Codegen API* uses the same object in its framework for code generation. This enables the tool, *P4M*, to invoke the *BaSyx Codegen API* using the created *AAS JAVA API* objects, as mentioned above, and generate code that corresponds to the model. The generated project is a holistic package ready for deployment of the contained AAS on any HTTP Server (which is specified in the model by the user during modeling phase).

The architecture and the interaction of each class in the code generator is shown inside the *Basyx Codegen API* block (Figure 4. The code generator is added to the Papyrus Codebase as an Eclipse Plugin. The *Project* class is used as the gateway to the several constituent generator classes of the code generator plugin. The code generator has a *FileUtils* class which contains static methods that help in the File operations at the system level - creating directories, copying files and artifacts, writing files with contents are some of the major functionalities that this class offers. There are four main generator classes associated with the *Project* class which take on the role of code generator from the model inputs - *AASModuleCreator*, *DEWorkspaceCreator*, *DataHandlerCreator* and *SubModelCreator*.

The *AASModuleCreator* - generates codes for classes that are responsible for creating the BaSyx Context Information and hosting the AAS thus created, on a Tomcat Server to have it available communicating with the HTTP Protocol. Communication with AAS, thus working with the REST API calls.

The *DEWorkspaceCreator* - generates a class *DynamicElementsWorkspace* which creates the space to define the behavior of SubmodelElements of the AAS that are defined by the user to be dynamic. The behavior of these dynamic elements is to be encoded using Java. This *DynamicElementsWorkspace* ensures that the user is able to deal with the code base at a single point and with knowledge of Java.

The *DataHandlerCreator* generates classes in the generated code that are responsible for generating the connector wrappers for assets and the dependent variable class types for the generated BaSyx code. The connector wrappers now supported are - *OPCUAConnectorWrapper*, *HTTPConnectorWrapper*, *MQTTConnectorWrapper*, *WebSocketConnectorWrapper* and *ROSConnectorWrapper*.

The *SubModelCreator* generates the constituents of the Submodels. By parsing the AAS instance from the *AAS Java API*, the SubModelCreator invokes the different submodel elements generator class - *FileGenerator* - for SubmodelElements *File*, *MLPGenerator* - for SubmodelElements *MultiLanguageProperty*, *SECGenerator* - for SubmodelElements *SubModelElementCollection*, *OperationGenerator* - for SubmodelElements *Operation* and *PropertyGenerator* - for SubmodelElements *Property*. The code is generated based on the user specification at the modeling environment, thereby making it conformant to AAS API in BaSyx.

B. Java Development Tool Synchronization with AAS Models

As mentioned above, the *DEWorkspaceCreator* generates a class *DynamicElementsWorkspace* which creates the Java class to define the behavior of dynamic submodelElements of the AAS. The behavior of these dynamic elements (Property, Operation) should be encoded using Java. In fact, it is easier for the user to specify the behavior of each dynamic SubmodelElement (Property, Operation) directly in the Java file (and not in the Papyrus model) to take advantage of the several features of the Eclipse *JDT Editor* like completion, syntax analysis, etc. However, if the user defines the behavior of a dynamic property by encoding a *getProperty()* in the *DynamicElementsworkspace* and regenerates its Java code, this behavior will be lost because the Papyrus model is not aware of it (the code is generated from the Papyrus model). To overcome this synchronization problem, we developed a functionality that ensures the automatic synchronization between the Java code edited by the user and the P4M UML/AAS model in both directions. If the user modifies the generated Java code, the UML/AAS model will be notified and updated. At the same time, if the user modifies the UML/AAS model, the Java code is notified and updated. This functionality improves the usability of the P4M tool by relieving the user from the error prone manual synchronization task.

VI. Academic Use Case - I4.0 Robotic Cell

We have developed an academic demonstrator representing an I4.0 Robotic Cell which is a proof of concept for I4.0 digital twin research. The I4.0 production cell comprises several components for production and communication as illustrated in Figure 5. It includes a Niryo Ned robotic arm, a TurtleBot3 AGV, a conveyor belt, a human operator, a storage zone, a workspace, and a Raspberry Pi 3 Model B+. In the initial development phase, Raspberry hosts an OPC UA PubSub brokerless server, which provides the operational data and the device operations access via an OPC UA information model.

The use case for this production cell involves transporting objects from the storage zone to the workspace for assembly by the operator. The I4.0 Robotic Cell demonstrates the ability to control a mini production cell using the P4M toolset. In this use case, the toolset provides the roles of modeling and deployment:

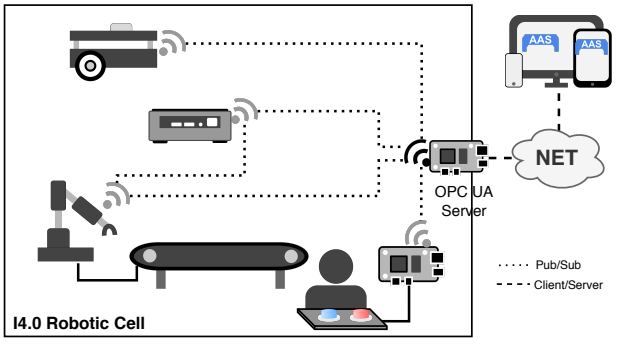

Fig. 5. I4.0 Robotic Cell demonstrator components

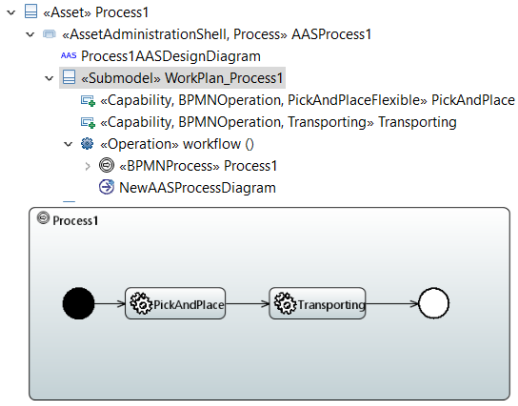

Fig. 6. AAS Process specification example

- Design standardized AAS models of production resources, and their provided capabilities and describe the production processes using BPMN diagrams.
- Generation of runtime AAS code for monitoring and automated control

A. I4.0 Robotic Cell AAS Models

According to the definition given by Plattform I4.0, all relevant participants are I4.0 components, which we call assets. The assets in the use case are the resources, production processes, and products introduced in the previous paragraph. To model the processes, BPMN diagrams are provided in P4M for modeling a process workflow.

The AAS model specification of the assembly process can be found in Figure 6, where a Submodel declares the capabilities required by the process steps and their semantic annotation comes from MaRCO ontology (PickAndPlaceFlexible & Transporting). An operation "Workflow" is be created to host the BPMN Process diagram of this production process.The BPMN Process diagram can also be used to describe the skill behaviors. The *skills* in the context of capability-based engineering refer to the implementation of AAS Capabilities. In our use case we modeled the skill by an AAS Operation, which has a behavior modeled as a BPMN process (Figure 7). The BPMN process defines the sequence of Operation Calls that compose the behavior (i.e. implementation) of the skill.

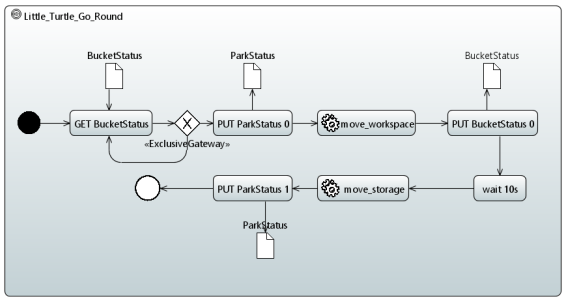

Fig. 7. BPMN Process diagram of TurtleBot3 transport implementation

An OPC UA server permits the accessibility of the assets' information model from the AAS server. For the resource modeling in P4M, in order to connect with real-world devices via OPC UA, the operations and dynamic properties of each device are specified in an "OperationalData Submodel" with their OPC UA nodeId information. Figure 8 shows the AAS design diagram of TurtleBot3. The "Capabilities_Submodel" specifies the transporting capability offered by the device. The "OperationalDatal_LittleTurtle" exposes not only the dynamic properties such as "BucketStatus" which refers to the object reception status of this robot but also the operations exposed by the OPC UA information model.

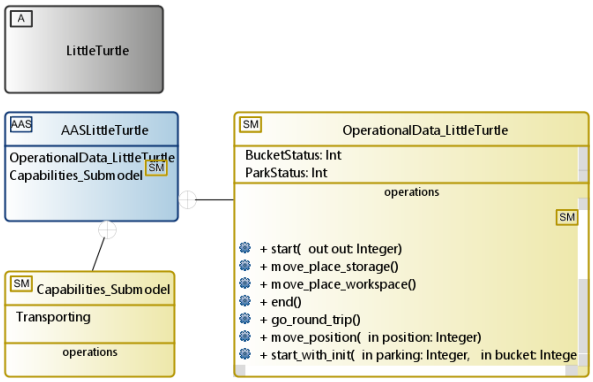

Fig. 8. AAS diagram of a TurtleBot3 instance

B. I4.0 Robotic Cell AAS Modules Generation

After the model specification phase, as presented in Section V-A, a right-click menu on the AAS model allows the generation of BaSyx executable code. The generated code contains the Java methods to get Property values. However, the behaviors of Operations still need to be implemented. To enable the synchronization between Java code and P4M UML/AAS model, the user needs only to right-click on the selected Operation and choose "Designer - Open a JDT Editor". This command will open the *DynamicElementsworkspace* Java class of the generated submodelElement code, where we can specify the behaviors. This synchronization function is reflected in that after saving the newly changed code, the code body will be saved in this AAS Operation as a UML method as shown in Figure 10.

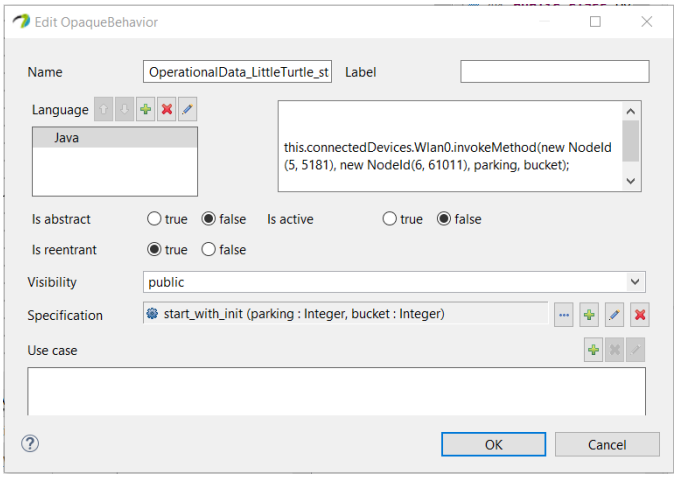

Fig. 10. JDT synchronization example

C. I4.0 Robotic Cell Process Execution

The functionality of transforming AAS models into BaSyx code facilitates the deployment of an AAS model into an HTTP server. The asset's dynamic information model can be accessed and modified through the HTTP requests sent to the dynamic AAS server. Additionally, the API for invoking AAS operations is available. To orchestrate the production process, all the resources' AAS servers involved in the process plan should be started and continuously accessible. To visualize

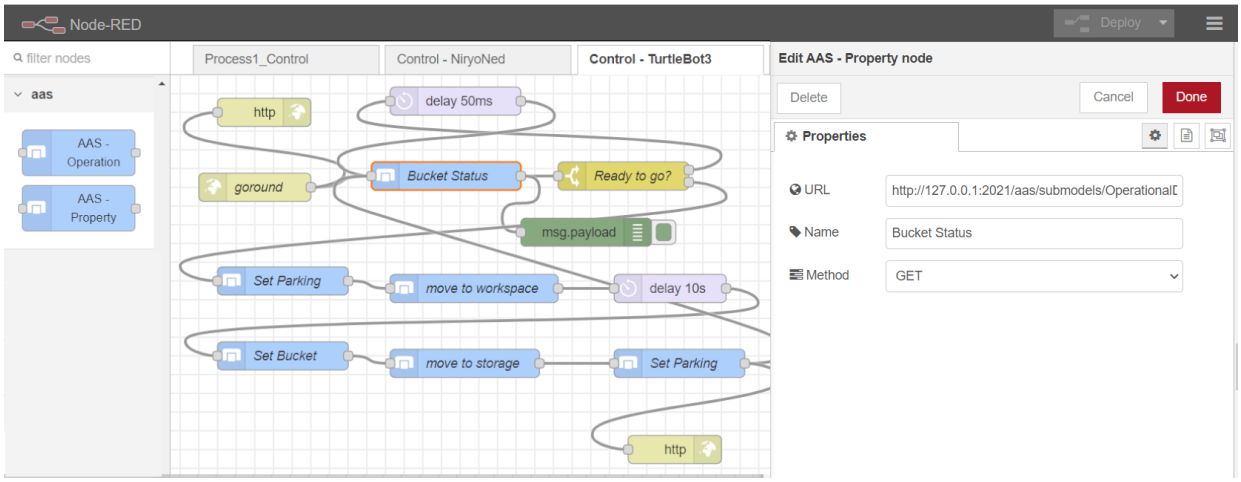

Fig. 9. Node-RED flow

the result of the process execution, we published a video[4] in the documentation of Papyrus4Manufacturing.

To implement the production process, we used Node-RED: a visual programming language development tool enabling hardware devices to connect with online services and APIs. The BPMN process in P4M can be easily transformed to Node-RED workflow (Figure 9) for the execution. Two customized AAS nodes in Node-RED have been developed to minimize the number of nodes involved in the workflow: (1) AAS Operation, this node enables the invocation of AAS Operations by specifying the URL endpoint and the input parameters. (2) AAS Property, this node permits the read/write value functionality of a given URL endpoint.

VII. Conclusion

In this paper, we proposed a novel model-driven toolset, Papyrus4Manufacturing, which enables the creation and deployment of digital twins following the AAS specification. The toolset integrates user-friendly editors for creating AAS models (digital twins) and provides an automatic deployment functionality based on model transformations and code generation. All along the development process of P4M, an educational use case has demonstrated the functionalities of this toolset.

Though P4M with its novel approach to bringing a model-based system engineering approach to the domain of DTs and AASs, it has a lot of potential for improvement. Integrating Grafana Dashboards for data visualization and analysis could enhance the value addition achieved. The current version of P4M does not include an integrated connector to a database, a possible further advancement would be the integration of a database for data storage. Last but not the least, P4M is at its current implementation not able to deploy multiple AASs to the server. Enabling the use of BaSyx registry to host multiple AASs would further enhance the usability and functionality of the tool.

References

[1] D. M. Grieves, "Digital twin: Manufacturing excellence through virtual factory replication," US Florida Institute of Technology, Melbourne (2014), Tech. Rep., 01 2014.

[2] L. Monostori, *Cyber-Physical Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 1–8.

[3] E. R. Carroll and R. J. Malins, "Systematic literature review: How is model-based systems engineering justified?." 3 2016. [Online]. Available: https://www.osti.gov/biblio/1561164

[4] "Details of the asset administration shell - part1, version 3.0rc02," Tech. Rep., 11 2020. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.pdf?__blob=publicationFile&v=5

[5] N. Shevchenko, "An introduction to model-based systems engineering (mbse)," Carnegie Mellon University's Software Engineering Institute Blog, Dec. 21 2020, accessed:30.06.2022. [Online]. Available: http://insights.sei.cmu.edu/blog/introduction-model-based-systems-engineering-mbse/

[6] D. D. Walden, G. J. Roedler, K. J. Forsberg, R. D. Hamelin, and T. M. Shortell, *Systems Engineering Overview*. San Diego, CA, USA: John Wiley & Sons, Inc., 2015, pp. 5–16.

[7] P. Liggesmeyer and M. Trapp, "Trends in embedded software engineering," *IEEE Software*, vol. 26, no. 3, pp. 19–25, 2009.

[8] D. K. Schweichhart, "Reference architectural model industrie 4.0 (rami 4.0)." [Online]. Available: https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference_architectural_model_industrie_4.0_rami_4.0.pdf

[9] Fraunhofer iese, eclipse basyx. [Online]. Available: https://www.eclipse.org/basyx/

[10] L. Stojanovic, T. Usländer, F. Volz, C. Weißenbacher, J. Müller, M. Jacoby, and T. Bischoff, "Methodology and tools for digital twin managementmdash;the fa3st approach," *IoT*, vol. 2, no. 4, pp. 717–740, 2021. [Online]. Available: https://www.mdpi.com/2624-831X/2/4/36

[11] A. Orzelski, M. Hoffmeister, and M. Ristin. Eclipse aasx package explorer. [Online]. Available: https://projects.eclipse.org/proposals/eclipse-aasx-package-explorer

[12] S. Wolf, K. Rehman, H. Dickel, T. Ostermann, M. Sauer, P. Huebner, and Y. Schiebelhut, "Sap/i40-aas." [Online]. Available: https://github.com/SAP/i40-aas

[13] G. D. Orio, P. Maló, and J. Barata, "NOVAAS: A reference implementation of industrie4.0 asset administration shell with best-of-breed practices from IT engineering," in *IECON 2019*. IEEE, 2019, pp. 5505–5512. [Online]. Available: https://doi.org/10.1109/IECON.2019.8927081

[14] P. D. Rudolf Pribiš, Lukáš Beňo, "Asset administration shell design methodology using embedded opc unified architecture server," *Electronics*, vol. 10, no. 20, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/20/2520

[15] S. Dhouib, A. Smaoui, I. Khemir, T. Bhanja, and V. Gezer, "Papyrus for manufacturing," 2023, (accessed: 06.04.2023). [Online]. Available: https://www.eclipse.org/papyrus/components/manufacturing/

[16] A. Roth and B. Rumpe, "Towards product lining model-driven development code generators," in *In Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development*. SciTePress, 2015, pp. 539–545. [Online]. Available: http://www.se-rwth.de/publications/Towards-Product-Lining-Model-Driven-Development-Code-Generators.pdf

[17] K. Hölldobler, J. Michael, J. O. Ringert, B. Rumpe, and A. Wortmann, "Innovations in Model-based Software and Systems Engineering," *The Journal of Object Technology*, vol. 18, no. 1, July 2019.

[18] "Papyrus model driven workbench," accessed: 30.06.2022. [Online]. Available: https://www.eclipse.org/papyrus/

[19] ISO/IEC/IEEE, "Systems and software engineering - architecture description," *ISO/IEC/IEEE 42010:2011(E)*, pp. 1 –46, 1 2011.

[20] "Omg, unified modeling language (uml) specification, version 2.5.1," accessed: 30.06.2022. [Online]. Available: https://www.omg.org/spec/UML/

[21] P. I4.0, "Describing capabilities of industrie 4.0 components." [Online]. Available: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Capabilities_Industrie40_Components.html

[22] ——, "Information model for capabilities, skills services." [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/CapabilitiesSkillsServices.html

[23] Y. Huang, S. Dhouib, and J. Malenfant, "Aas capability-based operation and engineering of flexible production lines," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2021, pp. 01–04.

[24] Y. Huang, S. Dhouib, L. P. Medinacelli, and J. Malenfant, "Semantic interoperability of digital twins: Ontology-based capability checking in aas modeling framework," in *2023 IEEE 6th International Conference on Industrial Cyber-Physical Systems (ICPS)*, 2023, pp. 1–8.

[25] E. Järvenpää, N. Siltala, O. Hylli, and M. Lanz, "The development of an ontology for describing the capabilities of manufacturing resources," *Journal of Intelligent Manufacturing*, vol. 30, no. 2, p. 959 – 978, 2019. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85048584278&doi=10.1007%2fs10845-018-1427-6&partnerID=40&md5=e848963588207b3b7f28956fdece2c3c

[26] OMG, "Business process model and notation v2.0," 2010. [Online]. Available: https://www.omg.org/spec/BPMN/2.0/

[27] "Admin-shell-io: Java model." [Online]. Available: https://github.com/admin-shell-io/java-model

[4] https://youtu.be/G5Hfinm_guE