Bridging the Gap between IT and OT with AAS digital twins and MDE techniques: An industrial waste management case study

Quang-Duy Nguyen, Saadia Dhouib, Eric Lucet, Antoine Le Mortellec, Fabien Baligand

Bridging the Gap between IT and OT with AAS Digital Twins and MDE Techniques: An Industrial Waste Management Case Study

Quang-Duy NGUYEN
Université Paris-Saclay, CEA, List
F-91120, Palaiseau, France
quang-duy.nguyen@cea.fr

Saadia DHOUIB
Université Paris-Saclay, CEA, List
F-91120, Palaiseau, France
saadia.dhouib@cea.fr

Eric LUCET
Université Paris-Saclay, CEA, List
F-91120, Palaiseau, France
eric.lucet@cea.fr

Antoine Le MORTELLEC
PROSYST
59300, Valenciennes, France
alemortellec@prosyst.fr

Fabien BALIGAND
Université Paris-Saclay, CEA, List
F-91120, Palaiseau, France
fabien.baligand@cea.fr

***Abstract*—Industry 4.0, involving technological advances such as the Internet of Things, artificial intelligence, and autonomous robotics, promises to bring great benefits and outstanding innovations to traditional manufacturing. To make this vision a reality, one key challenge consists in harmonizing and adapting these new technologies to the industrial environment. However, there are two major obstacles. First, the technologies, divided into information technologies (IT) and operational technologies (OT), are not interoperable by default, making the IT and OT convergence a complex and uneasy problem. Second, deploying an Industry 4.0 system requires the participation of experts from different domains who have different technical vocabularies and knowledge. Therefore, communication and collaboration between these experts can be complicated, and every misunderstanding may cause the failure of a part or the whole system. Regarding the mentioned obstacles, this paper shares our successful experiences and approaches in an industrial waste management case study of the project OTPaaS. In detail, we have applied standardized modeling approaches, more specifically, the asset administration shell (AAS) digital twins models, for the interoperability between IT and OT involved in the system and used model-driven engineering (MDE) techniques to automate the translation of data models between the different stakeholders in the project.**

***Index Terms*—Industry 4.0, Model-Driven Engineering, Asset Administration Shell, Papyrus for Manufacturing, OTPaaS**

I. Introduction

The emergence of technologies in Industry 4.0 provides advanced production tools that consequently involve new production models to maximize the tools' potential. Indeed, the industry has witnessed a shift from the hierarchical pyramid defined by ISA-95 to the network-structured model promoted in RAMI 4.0. On the one hand, ISA-95 systematically organizes the components of an industrial system into five levels based on their technologies and business objectives [1]. From low to high, they are `Field Device`, `Control Device`, `Station`, `Work Center` and `Enterprise`. The first three levels use operational technologies (OT)—hardware and software specialized and dedicated to some specific production purposes—to manipulate industrial devices, resources, and processes. The last two levels manage enterprise information and business applications with information technologies (IT). On the other hand, RAMI 4.0 tends to flexibly view all components involved in a production as connected elements [2]. This flexibility removes the separation between the OT and IT layers and reaches toward the IT/OT convergence. It promises to improve the speed of data exchanges between components and the system's resistance to partial failure, thus enhancing production performance and productivity.

In addition to the opportunity, the IT/OT convergence implies interoperability challenges. First is the incompatibility between IT and OT. In detail, hardware and software built for OT must follow strict and hard standards to be robust and reliable against tough industrial environments; however, software built for IT is more flexible, modular, and less constrained. Literally, OT and IT are not designed to satisfy each other's conditions; thus, they can not collaborate by default. The second obstacle in IT/OT convergence relates to human factors. Indeed, an Industry 4.0 system is composed of many technologies, involving the contributions of different experts from different domains with different technical knowledge. Engineering a part or the whole of such a system requires effective communication and exchanges between these experts since every misunderstanding during the development phase may cause system failures at operation time.

Regarding the IT and OT interoperability challenge, a solution is to use Asset Administration Shell (AAS) digital twins (DT) to bridge IT and OT. DT refers to a *"set of digital models"* representing a *"target entity with data connections that enable convergence between the physical and digital states at an appropriate rate of synchronization"* [3], [4]. AAS is an industrial-grade standard promoted by the Industrial Digital Twin Association (IDTA)[1]. Thus, the DTs based on the AAS standard, or AAS DTs, can satisfy OT requirements and

[1]https://industrialdigitaltwin.org/en/

are widely accepted by the industrial community. They are also extensible and adaptive to IT applications. In practice, AAS DTs can represent all manufacturing assets of the OT layer and expose their aspects to the IT layer. Consequently, IT applications can interact with the manufacturing assets through their AAS DTs. Moreover, to facilitate the communication and collaboration between the AAS DTs development team and the other technical providers, this research proposes adopting model-driven engineering (MDE) techniques to engineer AAS DTs. While MDE abstraction is about using models to represent the manufacturing assets with easy-understandable graphic designs, MDE automation allows the development team to generate code and implement AAS DTs rapidly.

This paper aims to share the successful experiences and lessons learned in using MDE techniques to automatically engineer and generate AAS DTs in a real industrial waste management use case. To the best of our knowledge, since there are no shared experiences on this approach [5], this research can be helpful to the public and private sectors.

The rest of this paper is organized as follows. Section II introduces the industrial waste management use case. Section III briefly presents the AAS standard and Papyrus for Manufacturing (P4M), our proposed model-driven tool to develop AAS DTs. Section IV details the methodology and steps for engineering AAS DTs to bridge IT and OT in the case study. Section V discusses our results and shares some lessons learned. Finally, a brief conclusion sums up this paper.

II. Application Context

OTPaaS[2] is a French national project involving 14 partners from different disciplines: research institutes, technology providers and industrial companies. The project aims to promote mature innovations for IT/OT convergence and to deploy such technical advances in six real industrial use cases.

Industrial waste management is one use case of the OTPaaS project. The goal is to automate the process of waste management in a printing factory using autonomous mobile robots (AMR). Traditionally, workers in this factory must manually process the following sequence of actions: (1) go to a full waste bin's position, (2) take it to the waste deposal area, (3) empty the waste bin, and (4) return it to the origin position. Since the full waste bin may be heavy and the distance between its position and the deposal area can be far, using AMRs can significantly reduce human labor.

Regarding the above, several project stakeholders engaged in this use case to produce a robust and intelligent solution:

- A stakeholder produces physical AMRs.
- A stakeholder deploys functionalities and features for AMRs.
- A stakeholder generates the factory's shop floor map and populates it with navigable roads to support AMRs calculating paths.
- A stakeholder produces the edge computing device.
- A stakeholder provides the Time-Sensitive Networking solution for real-time communication at the OT layer.
- A stakeholder deploys a Unified Data Collector Service (uDC)[3] along with computing modules as the control center to monitor AMR's behaviors.
- A stakeholder provides AAS DTs that expose AMRs' data to the control center and help AMRs manage the missions received from mission managers (MG).
- A stakeholder provides MGs for factory workers to generate missions to AMRs.

Due to the project's confidentiality reasons, we cannot share all the case study details, such as the factory's identification.

III. Background

This section presents AAS and P4M, which are the key technologies to our AAS DTs engineering approach.

A. Asset Administration Shell

AAS is a multi-parted standard describing elements and methods to implement an AAS DT. The first two parts contain the fundamental information for every AAS DT. Part 1 defines the structured meta-model to model manufacturing assets [6]. In simple words, it provides a set of concepts to describe the different aspects of an asset as submodels. Each submodel has submodel elements, which can be data elements, operation, capability, entity, submodel element collection, submodel element list, relationship element, and event. Part 2 proposes the programming application interface (API) and methods for IT applications to communicate with an AAS DT [7]. The other parts are out of this paper's scope.

Technically, an AAS DT comprises three basic elements: an AAS information model containing all the submodels, a network interface for data synchronization with its physical twin, and another network interface for interaction with external applications. An AAS server is an application that can host one or several AAS DTs.

B. Papyrus for Manufacturing

P4M is a toolset for implementing AAS DTs with the MDE approach [8]. In detail, it proposes a graphical modeling environment that includes diagram editors and supports various diagram models. Moreover, it defines a new UML profile translated from the AAS meta-model and proposes a new diagram for AAS information models, called the AAS design diagram (ADD), extended from the UML class diagram. P4M users can design models, choose the network interfaces, and generate an AAS server with the MDE automation feature.

Regarding data synchronization between an AAS DT and its physical asset, P4M reuses the communication framework from Eclipse BaSyx[4] and extends it with more network protocols. The current version supports HTTP REST, OPC UA, MQTT, WebSocket (WS), ROS, and ROS 2. For the network interface with external applications, it supports HTTP REST, OPC UA, and WS.

[2]https://linkedin.com/company/otpaas/

[3]https://prosyst.fr/udc-unified-data-collector-l-acces-aux-donnees-pour-l-usine-4-0/

[4]https://www.eclipse.dev/basyx/

IV. AAS Digital Twins Engineering

The methodology used to develop AAS DTs in the industrial waste management use case is mini-waterfall, a version of the well-known waterfall methodology [9]. Mini-waterfall is a loop of five phases: (1) specification, (2) design, (3) implementation, (4) testing, and (5) maintenance. Each loop should be fast and time-limited. Figure 1 illustrates the mini-waterfall methodology. In a sense, mini-waterfall inherits the basic phases of the waterfall methodology; in another sense, the concept of a loop of mini-waterfall is similar to the one of a sprint of the agile/scrum methodology [10]. The reason for choosing mini-waterfall relies on three points. First, the simplicity and popularity of the waterfall's five phases are useful for this multidisciplinary project since the experts from different domains can understand the AAS DTs development easily. Second, AAS DTs are only one module constituting the complete system; thus, their development should be light and flexible to adapt to the changed requirements of the other modules. Third, the AAS DTs' development team members do not have full-packed capabilities to practice agile/scrum.

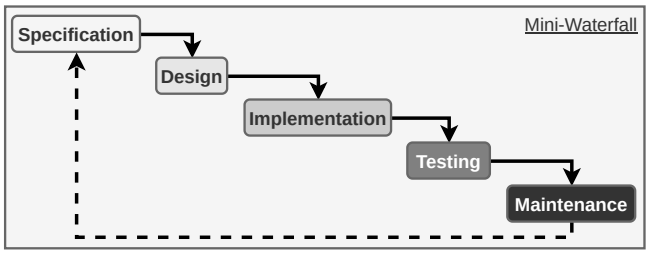

Figure 1: The mini-waterfall methodology

The development team applies the MDE techniques and uses the P4M tool in the design and implementation phases.

A. Specification

In the specification phase, the AAS DTs development team studies and analyses all the input resources proposed by the use case to clarify the needs and determine AAS DTs' required compositions. The input resources include:

- Description of the industrial waste management use case
- Part of the AMR datasheet proposed by its manufacturer
- Specification from the robotic stakeholder, including the operations of AMRs on the shop floor and their detailed hardware and software components
- Reports from project/use case/technical group meetings that list the requirements of stakeholders and the expected results from the project and use case

The following list details the four groups of requirements (**R**) corresponding to four features to implement.

R1. **An AMR must be able to expose some of its real-time data to IT applications**: the data are input for complex computations, such as AMR monitoring.

R1.0. **Technical data representation** is a requirement to present basic information identifying an AMR.

R1.1. **Velocity representation** is a requirement to present real-time data related to the velocity of an AMR. Note that the velocity of an AMR can have two parts: linear and angular. Moreover, other practical information, such as the average speed and velocity units, should be available.

R1.2. **Battery representation** is a requirement to present real-time data related to the battery of an AMR, including the voltage, current, percentage, and charging status. Also, other practical information, such as the percentage limit (maximal and minimal values) and the electricity measurement units, should be available.

R1.2.0. Charging status representation is a requirement to present the battery's four pre-defined charging states: charging, discharging, non-charging, and full.

R1.3. **Control representation** is a requirement to present real-time data related to the mode to control an AMR and the controlling status.

R1.3.0. Drive mode representation is a requirement to present the four pre-defined drive modes: autonomous, human following, joystick, and web service.

R1.4. **Localization representation** is a requirement to present real-time data related to the planar pose of an AMR, specified by three coordinated information of x, y, and theta. Moreover, other practical information, such as coordinate units, should be available.

R1.4.0. Localization status representation is a requirement to present the four pre-defined states: unloaded, stable, unstable, and lost.

R1.5. **Load representation** is a requirement to present real-time data related to the waste bin that the AMR is carrying, including the identification and the current weight of the waste bin.

R1.6. **Hardware components representation** is a requirement to present the information related to hardware components integrated into an AMR. They are two computers, seven cameras, one headlight, and one inertial measurement unit (IMU). The state of some hardware components (on or off) should be available.

R2. **An AMR must be able to receive missions from MGs**: a mission includes an ordered list of tasks that an AMR must operate.

R2.0. **Current mission representation** is a requirement to present real-time data related to the mission that an AMR is processing, including the state of the mission.

R2.1. **Missions management** is a requirement to manage a list of missions and distribute them to an AMR. It also includes the need to represent the basic information of each mission, such as the mission's name and identification.

R2.2. **Tasks management** is a requirement to manage a list of tasks associated with each mission. It also includes the need to represent the basic information of each task, such as the task's name and identification.

R3. **The shop floor must be able to expose its geometric data to AMRs**: the data includes a global map and the highlight positions on the map.

R3.0. Global map management is a requirement to manage the global map of the shop floor. The map exists in the format of a file; thus, it needs to present the information about the file type, graph name, and last updated time.

R3.1. Zones management is required to manage the shop floor's highlighted locations. Each zone includes information about the zone's name, type, and relative positions on the map (x-axis, y-axis, and z-axis values).

R3.1.0. Zone-type representation is a requirement to present the three pre-defined specific zone types: station, waste bin, and deposit.

R4. A registry must provide information for establishing the communication between an AMR and its AAS DT: the information could relate to the endpoint of the AMR if its AAS DT initiates the connection, or relate to the endpoint of the AAS DT if the AMR initiates the connection. The endpoint information includes the address and port.

In addition to the above functional requirements, some technical requirements (TR) are as follows.

TR1. The network protocol between AMRs and AAS DTs is WebSocket (WS)

TR2. The network protocol between MGs and AMRs' AAS DTs is MQTT.

TR3. The network protocols between IT applications and AMRs' AAS DTs could be HTTP.

TR4. The network protocols between IT applications and AMRs' AAS DTs could be OPC UA.

TR5. A global map should be updated into the shop floor's AAS DT using WS.

TR6. An AMR always starts to establish the connection to its AAS DT for the security issue at the OT layer.

B. Design

In the design phase, the AAS DTs development team defines all AAS DTs and puts them as the center of an architecture with other modules. The definition of AAS DTs relies on the requirements resulting from the specification phase. Figure 2 illustrates the role of AAS DTs in the use case.

The AAS DTs development team determines two types of AAS DTs: AAS AMR and AAS ShopFloor. In detail, each AMR has its corresponding AAS AMR, and the shop floor has its AAS ShopFloor. While AAS DTs run on a server on the local computing level, also known as fog, IT applications run on the global computing level or cloud that requires Internet access. The communication between an AAS DT and the cloud can be HTTP or OPC UA, depending on the IT application. AMRs at the Internet of Things (IoT) level communicate with AAS DTs using WS. An MG can be at IoT, fog, or cloud to send missions to AAS AMRs. Moreover, the development team names the module that generates a global map as a cartography generator (CG). A CG can also be at the IoT, fog, or cloud level to update cartography to the AAS ShopFloor with WS. In this architecture, the AAS DTs are rather in the middle of the OT and IT layers. Each AAS AMR implies a mission management service (MMS) addressing the

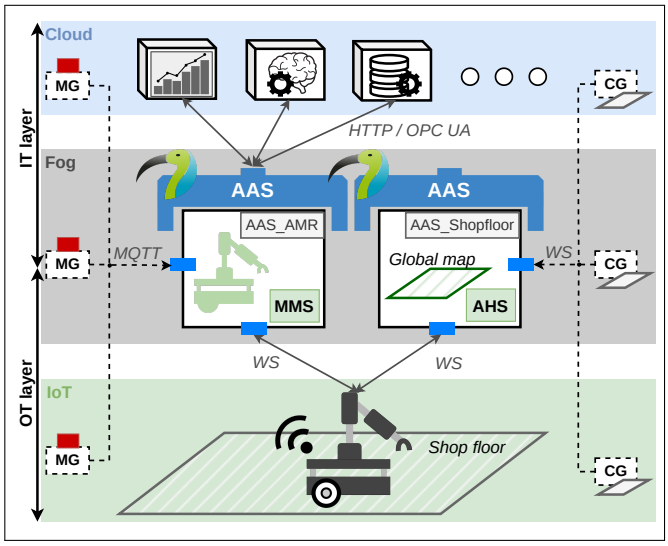

Figure 2: AAS DTs in the middle between OT and IT layers

requirement R2 defined in the specification phase. Also, the AAS ShopFloor implies the AMR helper service (AHS) that replies to the requirement R4.

To design AAS information models for the two AAS DT types, the AAS DTs development team determines the submodels and submodel elements that cover all requirements defined in the specification phase, as in Table I. The first column of the table lists the requirements. The second column shows the model element and its stereotype for each requirement. Note that «*AAS::AAS*» presents the stereotype *AAS*, «*AAS::SM*» presents the stereotype *submodel*, «*AAS::SMC*» presents the stereotype *submodel collection*, and «*UML::Enumeration*» presents the stereotype *enumeration*. While the first three stereotypes are part of the AAS profile of P4M, the last one is from the UML vocabulary. A requirement may need more than one model element; for example, R1.0 is covered by two *submodels* Nameplate and TechnicalProperties. In this table, a blue cell means that the inside model element is part of the AAS ShopFloor's information model, and the blank cell means that the inside model element is part of the AAS AMR's information model.

With the UML vocabulary and AAS profile, the AAS DT developers can straightforwardly model most functional requirements without considering technical ones. The only exception is the case of R4. To recall, it has two options: (1) the AAS ShopFloor provides the WS endpoint information of AMRs, or (2) the AAS ShopFloor exposes the WS endpoint information of AAS AMRs. While considering TR6, only the second option is satisfied, then in the model, the *submodel* Registry is designed to hold multiple AAS AMR Socket *submodel collections*.

Figure 3 illustrates the major part of an AAS AMR's information model. The *AAS* comprises eight *submodels* corresponding to eight aspects of an AAS AMR. Note that for scope limitation and confidential reasons, the two sub-

Table I: Requirements coverage by AAS information models

Requirement	«*stereotype*» Model Element
R1	«*AAS::AAS*» AAS AMR
R1.0	«*AAS::SM*» Nameplate «*AAS::SM*» TechnicalProperties
R1.1	«*AAS::SM*» Velocity
R1.2	«*AAS::SM*» Battery
R1.2.0	«*UML::Enumeration*» ChargingStatus
R1.3	«*AAS::SM*» Control
R1.3.0	«*UML::Enumeration*» DriveMode
R1.4	«*AAS::SM*» Localization
R1.4.0	«*UML::Enumeration*» LocalizationStatus
R1.5	«*AAS::SM*» Load
R1.6	«*AAS::SM*» Hardware «*AAS::SMC*» PCNavigation «*AAS::SMC*» PCPerception «*AAS::SMC*» Headlight «*AAS::SMC*» Camera «*AAS::SMC*» ICM «*UML::Enumeration*» DeviceStatus
R2	«*AAS::AAS*» AAS AMR
R2.0	«*AAS::SM*» Plan
R2.1	«*AAS::SMC*» Mission
R2.2	«*AAS::SMC*» Task
R3	«*AAS::AAS*» ShopFloor
R3.0	«*AAS::SM*» Map «*AAS::SM*» Cartography
R3.1	«*AAS::SM*» Zone
R3.1.0	«*UML::Enumeration*» ZoneType
R4 + TR6	«*AAS::AAS*» ShopFloor «*AAS::SM*» Registry «*AAS::SMC*» AASAMRSocket «*UML::Enumeration*» AASStatus

models, Hardware and Nameplate, are not presented in this paper. One typical *submodel* instance is Battery. It has nine *properties*, four of which are dynamic and five are static. Note that in the figures, static *properties* are underlined. The values of the four dynamic *properties* (`voltage`, `current`, `percentage`, and `charging_status`) are changeable and updated with real-time data retrieved from the AMR. While `voltage`, `current`, and `percentage` have float data values, `charging_status` has one value of the *enumeration* ChargingStatus. The value of the five static ones (`unit_voltage`, `unit_current`, `frequency_update`, `max_percentage`, and `min_percentage`) are constant. They aim to clarify the four dynamic *properties*; for example, `unit_voltage` "V" shows that the unit of the `voltage`'s value is volt. A *submodel* can also have *submodel collections* as elements. For example, the *submodel* Plan can have multiple Mission *submodel collections*; a Mission can have multiple Task *submodel collections*.

Figure 4 illustrates the AAS information model of the AAS ShopFloor. The *AAS* comprises three *submodels* corresponding to three aspects of the AAS ShopFloor. In detail, *submodel* Nameplate provides the basic identification of the shop floor, *submodel* Map presents the geometric aspect, and *submodel* Registry represents an abstract registry storing information of AMRs and AAS AMRs of the shop floor. A Map can have multiple Zone *submodel collections* representing some highlighted locations of the shop floor. A Registry can have multiple AAS AMR Socket *submodel collections* representing the WS endpoints of AAS AMRs.

The AAS DTs development team realises the design using the design environment of P4M. Figure 5 is a screenshot of the tool. In the model explorer window, developers can add, remove and modify all model elements, such as *submodel*, *property*, or *enumeration*. In the model editor window, developers input details to configure each model element.

C. Implementation

In the implementation phase, the AAS DTs development team implements runnable AAS DTs from the model designs resulting from the design phase. The two steps are as follows.

First, the developers profit from the code generation feature of P4M: they generate a Java project from the designed models with only one click. The project includes the basic skeleton of well-organized Java code. Then, the developers can complete the project with further configurations, such as the data source detail. This fast, effective approach satisfies most R1, R3, and R4 requirements.

The second step is necessary for the unsatisfied requirements from the first step. In detail, the AAS DT developers realize manually the following features unsupported by P4M.

- The number of model elements of an AAS information model proposed by P4M is unchangeable at runtime, so it is impossible to add more model elements. This limitation impacts the R2 requirements, in which a running AAS AMR may receive multiple missions and tasks. As the solution, the developers code an extended module that stores queues of missions and tasks.
- Regarding TR4, even though P4M already supports an OPC UA communication protocol between assets and AAS DTs, it proposes no OPC UA connector for the side between AAS DTs and IT applications. Thus, the developers improve the Java project generated by P4M with an OPC UA client/server mode extension that allows AAS DTs to write data to an external OPC UA Server by using an identity token.
- To recall, TR6 requires that an AMR always initializes the communication with its AAS AMR. Therefore, in the WS client/server communication, the AMR plays the client role, and AAS AMR plays the server role. Since P4M only supports WS connector as the client mode by default, the developers implement a module to turn an AAS AMR into a WS server and wait for the connection to start from its corresponding AMR.

After having runnable AAS DTs, the AAS DTs development

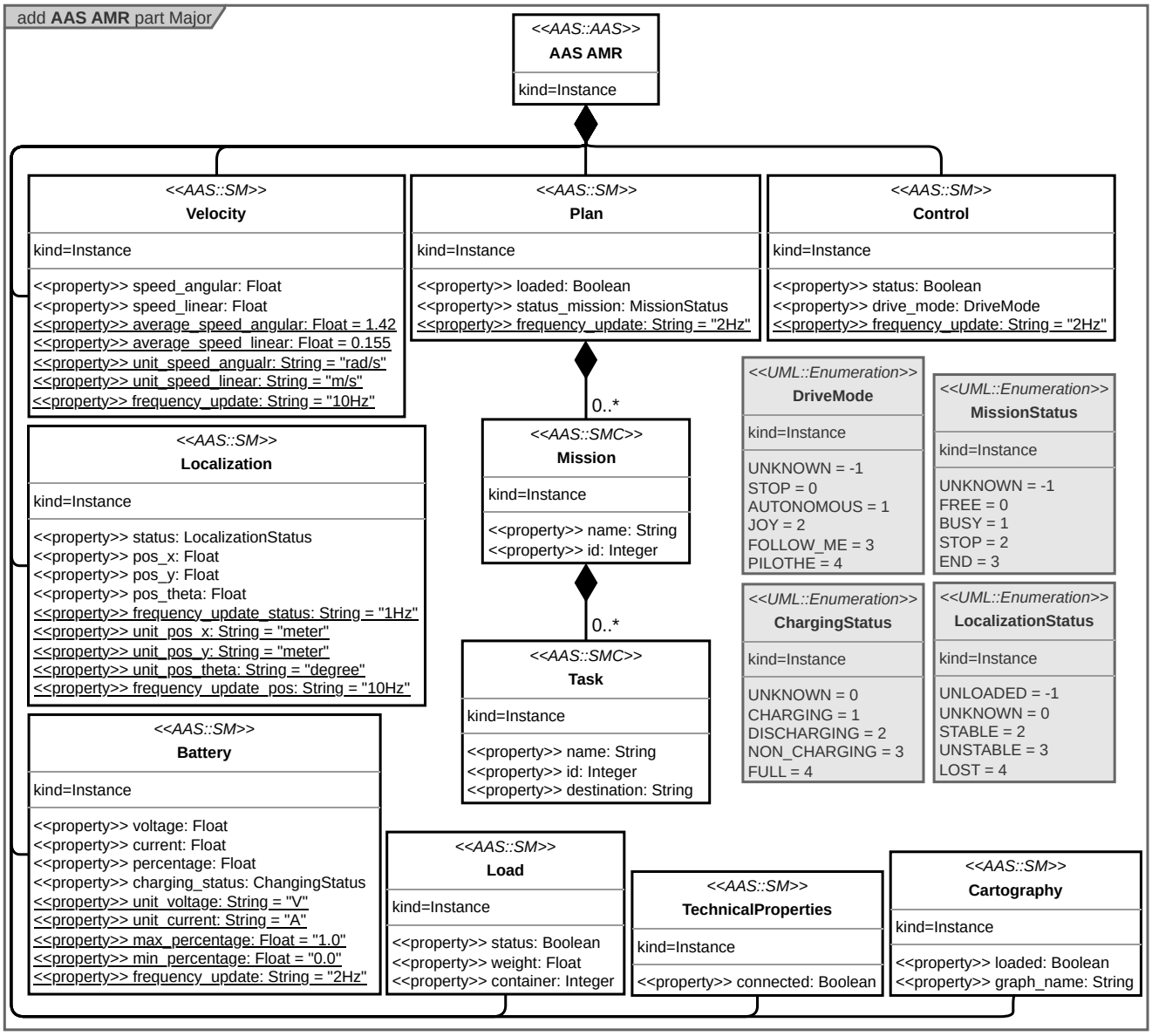

Figure 3: AAS design diagram of the major part of the AAS AMR's information model

team packages them as a docker[5] image to ensure they can run on different operating systems (Linux, Windows, and MacOS).

D. Testing

In the testing phase, the AAS DTs development team defines two strategies to validate the AAS DTs module. First, we prepare a system for unit testing the module, as shown in Figure 6. The system includes two computers. The first computer hosts six programs: (1) an AAS AMR, (2) an AAS ShopFloor, (3) a Mosquito[6] broker for the MQTT communication, (4) one Python application playing as an MQTT MG, (5) another Python application playing as a CG, and (6) another Python application playing as an AMR. The second computer hosts three programs: (1) Postman[7] to query data from the AAS AMR via HTTP, (2) an open62541[8] server as an OPC UA server to collect data from the AAS AMR via OPC UA communication, and (3) UaExpert[9] playing as an OPC UA client to query data from the OPC UA server. For unit testing, the Python applications on the first computer generate fake data that Postman and UaExpert can verify on the second computer. The AAS DTs development team tests each property in the AAS DTs' information model.

The second strategy to validate the AAS DTs module is integration testing. In detail, the AAS DTs development team uploads all resources related to the module, including docker

[5]https://www.docker.com/
[6]https://www.mosquitto.org/
[7]https://www.postman.com/
[8]https://www.open62541.org/
[9]https://unified-automation.com/products/development-tools/uaexpert.html

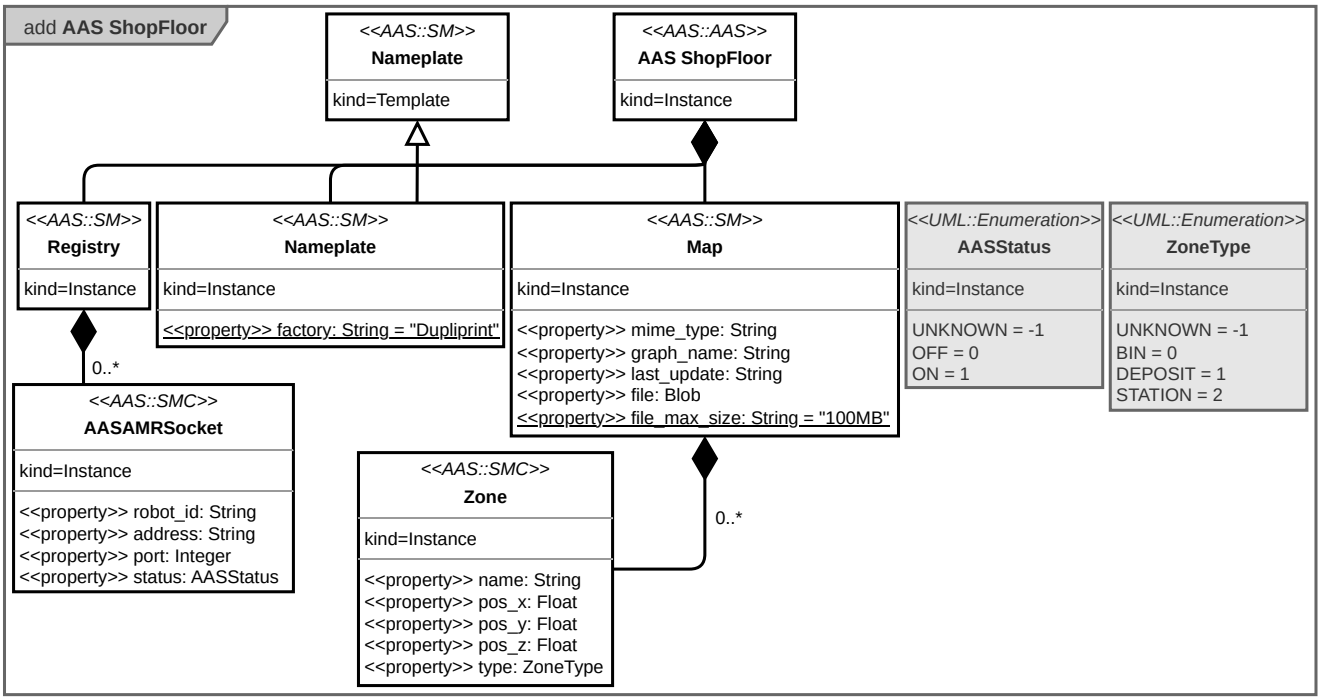

Figure 4: AAS design diagram of the AAS ShopFloor's information model

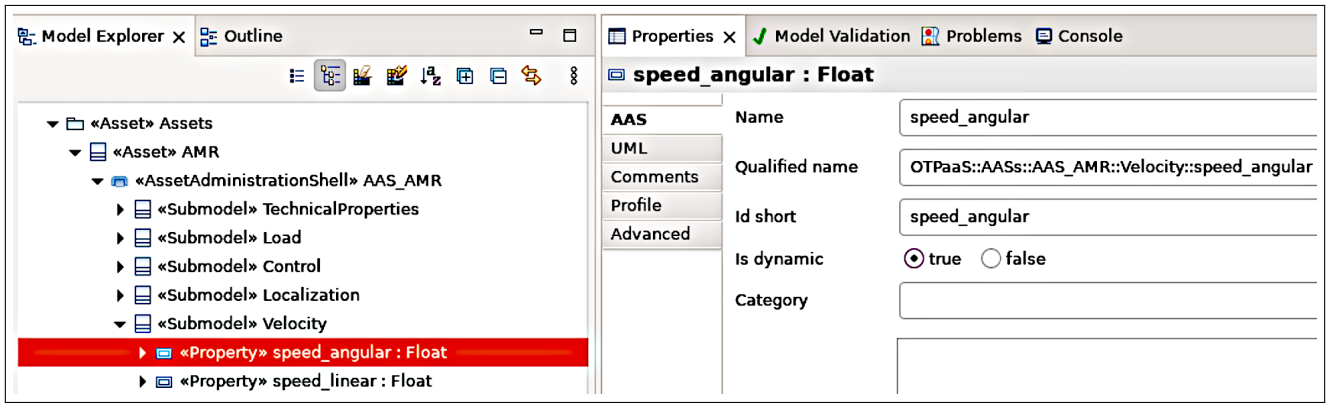

Figure 5: Screenshot of the P4M's design environment, including model explorer (left) and model editor (right) windows

containing the AAS DTs, a user guide, and supported codes, to a private cloud shared with other technical providers. Thus, the technical providers can test their modules with the AAS DTs module and return feedback. In the inverse sense, the technical providers also send their resources to the AAS DTs development team so we can test their modules on our side.

Note that for confidential reasons, the whole system testing is out of the scope of this paper.

E. Maintenance

In the maintenance phase, the AAS DTs development team waits for feedbacks from other technical providers and the use-case initiators. If there is new major requirements added, the AAS DTs development team will restart the loop of mini-waterfall methodology to produce a new version. If there is only minor requirements, the AAS DTs developers apply adequate modifications. At the end of the project, there are totally seven major versions. Each includes a docker image containing the AAS server and other necessary materials supporting the testing phase.

V. Discussion and Lesson Learned

Although the resulting AAS DTs and their engineering process were effective, there are still three points to discuss.

- The P4M has not yet supported the submodel element list, which is a part of the newest version (v3) of the AAS Part 1. This omission is understandable since the developments of the AAS standard and P4M are still ongoing. For this reason, in this project's context, the AAS DTs development team cannot profit from the P4M code generation feature to generate lists of missions and tasks but must manually implement them.

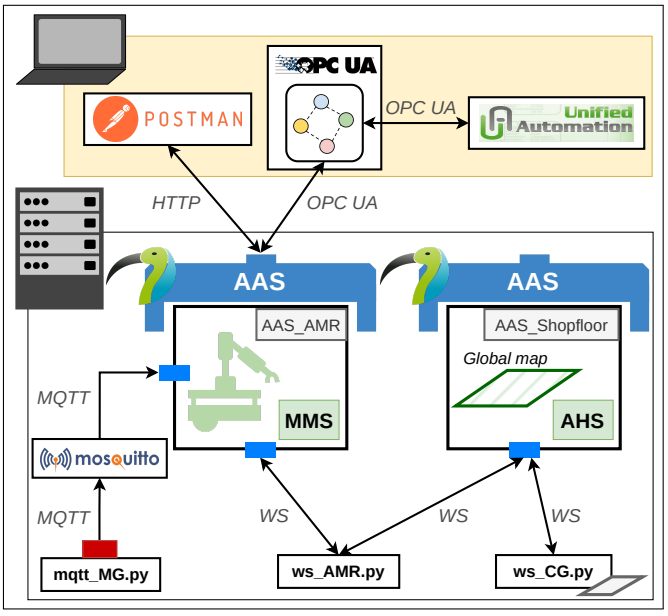

Figure 6: Architecture for unit testing

- The distributed concept of DTs—each AMR has one corresponding AAS AMR, and AAS AMRs can run on different host machines—can be a disadvantage for some modules in the system. For example, since AAS AMR manages the list of missions associated only with its AMR, it is more difficult for an MG to create the kind of mission involving multiple AMRs. Also, the control center may need to collect data from multiple hosts instead of a centralized server.
- As OTPaaS is not an open project, many details, such as uDC, CG and MG designs, cannot be shared in this paper without the permission of involved stakeholders. Also, the choice of communication protocols between AAS DTs and other modules depends on these stakeholders. For example, the OPC UA client/server communication mode is used between AAS AMRs and uDC, at the request of the control tower development team. In this case, uDC plays the role of OPC UA server, with an OPC UA information model built on the basis of our AAS AMR design and the I4AAS[10] companion specification.

Three useful lessons learned from using AAS DTs and MDE techniques in an industrial use case are as follows.

- Some special requirements, such as the security requirement TR6 presented in Section IV-A, demand the AAS DT development team to modify the original workflow of AAS DTs. To recall, instead of working as a WS client, an AAS DT must be changed to function as a WS server.
- In an industrial project with many stakeholders from various domains, developers should select a few basic and easily understandable MDE diagrams rather than trying to use them all. In practice, we found that the number and complexity of diagrams may lead to misunderstandings.

[10]https://opcfoundation.org/markets-collaboration/i4aas/

- The MDE automation feature is quite helpful in accelerating the production of the new product's version. Consequently, it can also speed up the work of the related stakeholders since they can practice integration tests sooner with their modules.

VI. Conclusion

This paper presents the practices of using the MDE approach in engineering AAS DTs for an industrial waste management case study. By leveraging the MDE abstraction and automation features, with the strong support of the P4M tool, the development team can provide multiple versions, thus effectively communicating with other stakeholders in the OTPaaS project. The resulting AAS DTs connect high-level applications and low-level assets, that is, to bridge IT and OT.

The AAS DTs development team has three plans to improve the P4M tool. The first is adding more network protocols, such as Profinet and Zenoh, so that P4M users can have more network connector choices. The second is to upgrade P4M to the latest AAS standard version. The third direction is to combine with other model types, and the tool can generate other forms of a target asset, such as a 3D visualization.

Regarding OTPaaS, the future step leverages federated learning on this current system to resolve industrial problems with the real-time data collected from AMRs and the other factory assets.

VII. Acknowledgments

This work was carried out in the scope of the OTPaaS project. This project has received funding from the French government as part of France 2030's "Cloud Acceleration Strategy" plan.

References

[1] International Society of Automation, "Enterprise-Control System Integration - Part 1: Models and Terminology," ANSI, North Carolina, American National Standard ANSI/ISA-95.00.01-2010, 2010.

[2] R. Heidel, M. Hoffmeister, M. Hankel, and U. Döbrich, *Industrie 4.0: The Reference Architecture Model RAMI 4.0 and the Industrie 4.0 component*. Germany: Standardization Council Industrie 4.0, 2019.

[3] International Electrotechnical Commission, "Digital twin - Concepts and terminology," International Organization for Standardization, International Standard ISO/IEC 30173:2023-11, Nov. 2023.

[4] Frédéric Sanchez, "Digital Twin: Leveraging the Digital Transformation of the Industry," Alliance Industrie du Future, France, Brochure, 2023.

[5] T. A. Abdel-Aty, E. Negri, and S. Galparoli, "Asset Administration Shell in Manufacturing: Applications and Relationship with Digital Twin," *IFAC-PapersOnLine*, vol. 55, no. 10, pp. 2533–2538, 2022.

[6] Industrial Digital Twin Association, "Specification of Asset Administration Shell - Part 1: Metamodel Schema," IDTA, Germany, Specification IDTA Number: 01001-3-0, Mar. 2023.

[7] Industrial Digital Twin Association, "Specification of Asset Administration Shell - Part 2: Application Programming Interfaces," IDTA, Germany, Specification IDTA Number: 01002-3-0, Jun. 2023.

[8] S. Dhouib, Y. Huang, A. Smaoui, T. Bhanja, and V. Gezer, "Papyrus4Manufacturing: A Model-Based Systems Engineering approach to AAS Digital Twins," in *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sinaia, Romania: IEEE, Sep. 2023, pp. 1–8.

[9] P.-A. Muller and N. Gaertner, *Modélisation objet avec UML*. Paris: Eyrolles, 2004.

[10] K. Schwaber and M. Beedle, *Agile software development with Scrum*, ser. Series in agile software development. Upper Saddle River, NJ: Prentice Hall, 2002.