From requirement specification to OPC UA information model design: A product assembly line monitoring case study

Quang-Duy Nguyen, Saadia Dhouib, Kunal Suri, Fadwa Tmar

From Requirement Specification to OPC UA Information Model Design: A Product Assembly Line Monitoring Case Study

Quang-Duy NGUYEN, Saadia DHOUIB, Kunal SURI and Fadwa REKIK
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
Email: quang-duy.nguyen@cea.fr, saadia.dhouib@cea.fr, kunal.suri@cea.fr, fadwa.tmar@cea.fr

***Abstract*—Open Platform Communication Unified Architecture (OPC UA) has emerged as a highly-demanded standard in building industrial systems. One factor in this success is the concepts of OPC UA address space and OPC UA information model. OPC UA address space provides a mechanism to represent the resources of an OPC UA server and its field devices as OPC UA nodes. OPC UA information model structures OPC UA nodes as a schema. Once other devices and systems understand the schema, they can interact appropriately with the industrial system. Thus, designing the OPC UA information model is an undeniable and essential practice of OPC UA-based industrial system engineering. However, both academia and industry suffer from a lack of shared experiences on this subject. This gap motivated us to share our experiences on the development of an OPC UA information model. These experiences are based on an actual product assembly line monitoring case study developed at CEA LIST. Overall, in this paper, we aim to throw some light on the steps involved in the transformation of a requirement specification into an OPC UA information model.**

***Index Terms*—OPC UA, Information model, Design, Product assembly line, Robotic cell, System engineering**

I. Introduction

Open Platform Communication Unified Architecture (OPC UA) is a widely-used standard in the industry. It is developed and maintained by the OPC Foundation, a group of more than 850 industry members. OPC UA's newest version, v1.05, consists of more than twenty-three specifications for building an industrial system with reliability, security, and interoperability. Its specifications part 3 and part 5 are the keys to semantic interoperability. Part 3, Address Space, provides the definition of OPC UA nodes and implies a mechanism to represent the resources of an OPC UA server and its field devices as OPC UA nodes [1]. Part 5, Information Model, defines a vocabulary containing the most basic OPC UA nodes and implies a mechanism to structure OPC UA nodes as an OPC UA information model [2]. In other words, an OPC UA information model is a schema of OPC UA nodes. This paper considers the vocabulary presented in the first twenty-three specifications as the standard information model. All other OPC UA information models rely on the standard information model. An OPC UA-based industrial system always has a domain-specific OPC UA information model and an OPC UA address space to store OPC UA nodes. Often, an OPC UA server manages the OPC UA address space. Other devices and systems can understand the system by learning from its OPC UA information model. Then, they can access its resources through the OPC UA address space lying in the OPC UA server. Figure 1 illustrates an OPC UA-based industrial system.

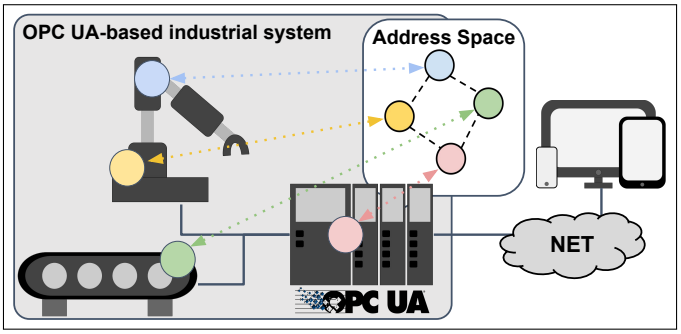

Fig. 1. Example of an OPC UA-based industrial system

OPC UA information model is an undeniable and essential part of an OPC UA-based information system. Therefore, designing the OPC UA information model must be part of OPC UA-based information system engineering. However, it still lacks sharing experiences on this subject. Instead of focusing on the design step, some available research works describe steps for generating, serializing, and implementing the OPC UA information model and address space [3], [4]. The partners of the OPC Foundation often propose companion specifications that include their domain-specific OPC UA information model and examples of how to use them. However, such examples are usually minimal. While in other domains, such as ontology-based system engineering, there exists research that focuses on the reason and steps to analyze data from requirements to produce ontological schema design [5]. Their sharing is useful to system developers in the domain.

In addressing the above issue, this paper aims to share experiences transforming a requirement specification into an OPC UA information model design. This sharing is meaningful and compelling for two reasons. First, the experiences are from the actual case study of developing the Product Assembly Line (PAL) monitoring system deployed in the LocalSEA robotic cell of CEA LIST [6]. The PAL is also used as a testbed for the ongoing work in context of Eclipse Papyrus for Manufacturing development environment and related toolset[1]. Second, this

[1]https://www.eclipse.org/papyrus/components/manufacturing/

case study's OPC UA information model is complex. Indeed, it models hardware, software, and abstract components. Also, it deals with different complexity of data: real-time data, historical data, and summary data. Each complexity of components and data requires a different strategy to model.

The development methodology applied in this case study is mini-waterfall. It contains five phases: specification, design, implementation, testing, and maintenance [7]. However, the scope of this paper is only the two first phases.

The organization of this paper is as follows. Section II introduces the case study, which includes the context and motivation of this research. Section III focuses on analyzing the requirement specification. Then, Section IV describes in detail the transformation from the analysis into an OPC UA information design. Finally, a brief conclusion sums up this paper and opens a discussion.

II. Case Study: Product Assembly Line Monitoring System

PAL refers to *"a manufacturing process where the bill-of-material parts and components are attached one-by-one to a unit in a sequential way by a series of workers to create a finished product"* [8]. Note that a worker can be a robot or a human in the modern industry. A workstation is a point in the series where one or several workers stay and do the same job. The PAL system deployed in LocalSEA illustrates box manufacturing. In which, a product is composed of two parts: a box and its cover. While a cover's color can be red or green, all boxes are the same. The PAL system is composed of two workstations. The first workstation, called storage, stores all covers. The worker in the storage is a Niryo Ned[2] robotic arm that can pick up one cover and drops it into a carrier. This procedure is called a pick-n-drop cycle. Then the carrier can transport it to the second workstation. There are two carriers: a conveyor belt carries red covers, and a mobile robot carries green covers. The mobile robot is a TurtleBot3 Waffle Pi[3]. The second workstation is a workspace where a human assembles boxes with red covers, and another assembles boxes with green covers. They produce two different products. Figure 2 illustrates the PAL system. In this scenario, there are two further remarks. First, in practice, Niryo Ned controls and supplies energy directly to the conveyor belt. Thus, it is possible to consider the conveyor belt as a component of Niryo Ned. Second, the park is a new component, representing the position where the mobile robot waits to carry a new cover. It is neither hardware nor software but an abstract component.

PAL monitoring involves getting real-time and historical data from the PAL system and viewing such data in a human-understandable interface. Thus, the PAL monitoring system requires a server to connect to field devices, collect data from them, record data when necessary, and expose such data to external clients. The server in this case study is an OPC UA server. The field devices are the two robots (Niryo Ned and TurtleBot3 Waffle Pi) and the conveyor belt.

[2]https://niryo.com/product/ned-education-research-cobot/

[3]https://robotis.us/turtlebot-3-waffle-pi/

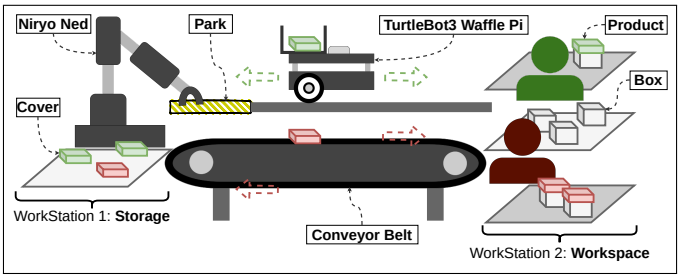

Fig. 2. Product Assembly Line for box manufacturing deployed in LocalSEA

There are three stakeholders with different interests in data outputs. The first stakeholder plans to visualize the position and movement of Niryo Ned on an external computer in real-time. The second one needs to record the periods when the two robots work. Finally, the third stakeholder demands the answer to the following questions:

- At any given time, how many covers have been handled by Niryo Ned? How many red covers? How many green covers?
- How many covers are on the storage before each pick-n-drop cycle?
- At any given time, what is the total energy consumption of Niryo Ned and TurtleBot3 Waffle Pi?
- At any given time, are the conveyer belt and TurtleBot3 Waffle Pi still running?

III. Requirement Specification Analysis

In system development, the specification is the output from the specification phase. It contains all the information related to the case study and the requirements needed to deploy the next phase. The specification of the PAL monitoring case study includes mainly the eight groups of information:

(1) Information for project management.
(2) Description of Niryo Ned, TurtleBot3 Waffle Pi, and the conveyor belt.
(3) Description of the available network infrastructure and communication method.
(4) Description of the role of human workers.
(5) Description of the final products.
(6) Description of the working procedure of the PAL.
(7) Description of the expected output data.
(8) Description of the services that the system can provide.

Among the above, only information groups (2), (3), (6), and (7) are necessary for the OPC UA information model design. Note that the information group (7) is translated from the demands of the stakeholders as follows.

- Concerning the first stakeholder's demand, the expected data is *the rotation angles of the robot arm's six joints*. The unit of each joint's value should be radian (rad) or milliradian (mrad). The manufacturer defines the limitation of each joint. For example, the angle that the first joint can rotate is between -2970 and 2970 mrad.
- Concerning the second stakeholder's demand, the information about when the two robots are "on" or "off" is not informative to record in the historical database. Thus,

system designers propose to record *the temperature of the two robots' control unit processors (CPU)* instead. The unit of each value is degree Celsius (°C). They have the same meaning in this case: when a robot is on, the CPU temperature of the robot's controller is above 0°C.

- Concerning the third stakeholder's demand, the questions turn into the following information:
  - *The number of (red) covers transported via conveyor belt.* The data type is an integer.
  - *The number of (green) covers transported via TurleBot3 Waffle Pi.* The data type is an integer.
  - *The number of covers that Niryo Ned's camera has found in the storage.* The data type is an integer.
  - *The energy consumption of Niryo Ned.* The unit of energy consumption is Joule (J). The data type of energy consumption is double.
  - *The energy consumption of TurtleBot3 Waffle Pi.*
  - *The status of Niryo Ned.*
  - *The status of TurtleBot3 Waffle Pi.*
  - *The status of the conveyor belt.*

For technical reasons, the information about *the status of the park* is also required. Indeed, it enables Niryo Ned to recognize whether the mobile robot is ready for transportation.

There are some remarks from the above information. First, the expected data from the first and second stakeholders can be obtained directly from data sources of the PAL system's devices without further processing. Thus, they are raw data. While the first stakeholder demands real-time raw data, the second is interested in historical data. Unlike the above, the expected data from the third stakeholder requires summary data. Such complex data usually are aggregated and processed by one or several software components. For example, no module works as an energy calculator integrating inside Niryo Ned. Thus, system designers can design a software component representing this module, and system developers must develop a program that calculates energy consumption and links the result to the software component.

System designers study the above information and extract five groups of requirements corresponding to five elements of the PAL: Niryo Ned, TurtleBot3 Waffle Pi, the storage, the conveyor belt, and the park. There are two types of requirements: (1) **object** and (2) `variable`. They all should be described with at least a title and a description. Each `variable` requirement must be associated with a specific data type.

R1. **Niryo Ned** is an object requirement to represent the Niryo Ned robotic arm.

R1 `Status`: is a variable requirement to represent the status of Niryo Ned. It should have at least three states: available, busy, and offline.

R1.1. **Controller**: is an object requirement to represent the Raspberry Pi 4 that controls Niryo Ned.

R1.1.1. **CPU**: is an object requirement to represent a microprocessor of the Raspberry Pi 4.

R1.1.1 `Temperature`: is a variable requirement to represent the CPU temperature. All `temperature` variables should be described with the unit °C. Since the data is periodically recorded in a historical database, it should associate with a valid interval representing the recording frequency.

R1.2. **Joint base**: is an object requirement to represent the joint that connects the base and shoulder of Niryo Ned.

R1.2 `Actual position`: is a variable requirement to represent the rotation angle of the joint base. All `actual position` variables should be described with the unit rad or mrad, and the range of motion associated with the minimum and maximum rotation angle values.

R1.3. **Joint shoulder**: is an object requirement to represent the joint that connects the shoulder and arm.

R1.3 `Actual position`: is a variable requirement to represent the rotation angle of the joint shoulder.

R1.4. **Joint elbow**: is an object requirement to represent the joint that connects the arm and elbow.

R1.4 `Actual position`: is a variable requirement to represent the rotation angle of the joint elbow.

R1.5. **Joint forearm**: is an object requirement to represent the joint that connects the elbow and forearm.

R1.5 `Actual position`: is a variable requirement to represent the rotation angle of the joint forearm.

R1.6. **Joint wrist**: is an object requirement to represent the joint that connects the forearm and wrist.

R1.6 `Actual position`: is a variable requirement to represent the rotation angle of the joint wrist.

R1.7. **Joint hand**: is an object requirement to represent the joint that connects the wrist and hand.

R1.7 `Actual position`: is a variable requirement to represent the rotation angle of the joint hand.

R1.8. **Red cover counter**: is an object requirement to represent a software component that counts the number of covers transported via the conveyor belt.

R1.8 `Count number`: is a variable requirement to represent the number counted by the red cover counter.

R1.9. **Green cover counter**: is an object requirement to represent a software component that counts the number of covers transported via TurtleBot3 Waffle Pi.

R1.9 `Count number`: is a variable requirement to represent the number counted by the green cover counter.

R1.10. **Energy calculator**: is an object requirement to represent a software component that calculates the energy consumption of Niryo Ned during a period.

R1.10 `Energy consumption`: is a variable requirement to represent the calculation result. All `energy consumption` variables should be described with the unit joule (J).

R2. **TurtleBot3 Waffle Pi:** is an object requirement to represent TurtleBot3 Waffle Pi.

R2 `Status`: is a variable requirement to represent the status of TurtleBot3 Waffle Pi. It should have at least three states: available, busy, and offline.

R2.1. **Controller**: is an object requirement to represent the Raspberry Pi 4 that controls TurtleBot3 Waffle Pi.

R2.1.1. **CPU**: is an object requirement to represent a microprocessor of the Raspberry Pi 4.

R2.1.1 `Temperature`: is a variable requirement to represent the

CPU temperature.

R2.2. **Energy calculator**: is an object requirement to represent a software component that calculates the energy consumption of TurtleBot3 Waffle Pi during a period.

R2.2 `Energy consumption`: is a variable requirement to represent the calculation result.

R3.1. **Storage**: is an object requirement to represent the physical storage. It is optional.

R3.1.1. **Storage counter**: is an object requirement to represent a software component that counts the total of covers at the storage.

R3.1.1 `Count number`: is a variable requirement to represent the number of covers at the storage.

R4. **Conveyor belt**: is an object requirement to represent the conveyor belt of this case study.

R4 `Status`: is a variable requirement to represent the status of the conveyor belt. It has only two states: on and off.

R5. **Park**: is an object requirement to represent the position where the TurtleBot3 Waffle Pi waits to carry a cover. It is an abstract component.

R5 `Status`: is a variable requirement to represent the state if the TurtleBot3 Waffle Pi is available in the park.

IV. OPC UA Information Model Design

The strategy to design our domain-specific OPC UA information model for PAL monitoring is to reuse the existing vocabulary before defining new concepts. This approach enables our designers to profit from well-defined concepts of industrial organizations and also improves the interoperability between OPC UA-based industrial systems. The group of new-defined concepts is called the custom-specific information model.

A. Vocabulary Selection

The vocabulary to study is the standard information model and the list of 40 companion specifications approved by the OPC Foundation. Among them, the OPC UA information model part 100 Devices (CS-DI) is one of the most important. It provides generic fundamental concepts for the topology of an industrial system, including all devices, hardware and software components, networks, and features [9]. It is the skeleton for many other companion specifications to extend.

Next, OPC UA information for Robotics (CS-Robotics) is another companion specification information model necessary for this case study. It is presented by the working group of the Mechanical Engineering Industry Association (VDMA), and OPC Foundation [10]. CS-Robotics also relies on CS-DI. The working group of VDMA and OPC Foundation aim to propose a vocabulary for multiple industrial robot types; however, CS-Robotics is still developing, and its current version only supports robotic arms. It is possible to divide this vocabulary into four groups related to four main objects: motion device system, controller, motion device, and safety state. Logically, a motion device system contains the three others.

Other companion specifications can be used to model some requirements in this case study. The companion specification for Process Automation Devices is worth mentioning. It includes an extension of analog measurement variables that can detail *BaseAnalogItemType* of CS-DI, such as the concept of *TemperatureMeasurementVariableType* can cover the CPU temperature requirements. However, their impact on this case study is too small compared to the cost of loading the whole vocabulary. Thus, our designers decide to reuse only the standard information model, CS-DI, and CS-Robotics and define new custom-specific concepts when necessary.

B. Requirement Coverage

Requirement coverage is the step to selecting the concepts that can model requirements. In OPC UA information model language, the concept is a type such as ObjectType and VariableType. Table I presents the relation between the requirements listed in Section III and the concepts to cover them. In this table, each row represents the coverage of a requirement. The first column identifies the requirement. The second column contains the exact concept covering the requirement. The representation format of a concept comprises two parts: **domain** and **Concept**. The part **domain** is the namespace of the vocabulary that holds the concept. In detail, ns0 is for the standard information model, ns2 is for CS-DI, ns3 is for CS-Robotics, and ns4 is for the custom-specific information model. The part **Concept** is the name of concept.

As dedicated to robotic arms, many concepts from CS-Robotics can cover the group of requirements R1 related to Niryo Ned. However, there are some exceptions. First, no concept in our selected information models is specifically for the CPU. Since the generic *ns2:DeviceType* is abstract, our designers cannot use it directly but need to define the subtype *ns4:ComputerComponentType* to cover the CPU requirements. Fortunately, the CPU temperature variable requirements can be covered by *ns0:AnalogUnitType*. Second, concerning the software components, including counters and energy calculators, our designers can just use *ns2:SoftwareType* to cover them. However, their associated variables are summary data. Thus, to explicitly represent such complexity, it is necessary to define the *ns4:AggregatedItemType* variable type, as the subtype of *ns0:DataItemType*, dedicated to summary and processed data.

Concerning the group of requirements R2, there is not yet any vocabulary for mobile robots. Thus, our designers copy the schema from CS-Robotics and produce another version for mobile robots with some replacements. In detail, the new concepts related to mobile robots replace the concepts related to motion devices. Also, the new concepts related to wheels replace the concepts related to axes. Since the two types of robots share many identical characteristics, it is practical and easy to reuse well-defined concepts from CS-Robotics, such as *ns3:ControllerType*, with mobile robots. The schema of mobile robots is a part of the custom-specific information model.

Group R3 contains three elements. First, the storage is a component of the system, but it should not be modeled as a *ns2:DeviceType* or *ns2:SoftwareType*. Our designers then defines *ns4:PlaceType* as a subtype of the *ns2:ComponentType* object type to cover storage. On the other hand, the storage counter can be covered by *ns2:SoftwareType*, and the

TABLE I
REQUIREMENTS COVERAGE BY THE STANDARD INFORMATION MODEL (NS0), CS-DI (NS2), CS-ROBOTICS (NS3), AND CUSTOM-SPECIFIC (NS4)

Requirement	domain:Concept
R1. Niryo Ned	ns3:MotionDeviceSystemType
R1 Status	ns0:MultiStateDiscreateType
R1.1. Controller	ns3:ControllerType
R1.1.1. CPU	ns4:ComputerComponentType
R1.1.1 Temperature	ns0:AnalogUnitType
R1.2. Joint base	ns3:AxisType
R1.2 Actual position	ns0:AnalogUnitType
R1.3. Joint shoulder	ns3:AxisType
R1.3 Actual position	ns0:AnalogUnitType
R1.4. Joint elbow	ns3:AxisType
R1.4 Actual position	ns0:AnalogUnitType
R1.5. Joint forearm	ns3:AxisType
R1.5 Actual position	ns0:AnalogUnitType
R1.6. Joint wrist	ns3:AxisType
R1.6 Actual position	ns0:AnalogUnitType
R1.7. Joint hand	ns3:AxisType
R1.7 Actual position	ns0:AnalogUnitType
R1.8. Red cover counter	ns2:SoftwareType
R1.8 Count number	ns4:AggregatedItemType
R1.9. Green cover counter	ns2:SoftwareType
R1.9 Count number	ns4:AggregatedItemType
R1.10. Energy calculator	ns2:SoftwareType
R1.10 Energy consumption	ns4:AggregatedItemType
R2. TurtleBot3 Waffle Pi	ns4:MobileRobotSystemType
R2 Status	ns0:MultiStateDiscreateType
R2.1. Controller	ns3:ControllerType
R2.1.1. CPU	ns4:ComputerComponentType
R2.1.1 Temperature	ns0:AnalogUnitType
R2.2. Energy calculator	ns4:SoftwareType
R2.2 Energy consumption	ns4:AggregatedItemType
R3.1. Storage	ns4:PlaceType
R3.1.1. Storage counter	ns2:SoftwareType
R3.1.1 Count number	ns4:AggregatedItemType
R4. Park	ns4:PlaceType
R4 Status	ns0:TwoStateDiscreteType
R5. Conveyor belt	ns4:ConveyorBeltType
R5 Status	ns0:TwoStateDiscreteType

count number can be covered by *ns4:AggregatedType*. The park of group R4 can be covered by *ns4:PlaceType*. To cover the conveyor belt of the group R5, our designers define *ns4:ConveyorBeltType* as the subtype of *ns4:DeviceType*. The variables of the park and conveyor belt are status representing two states. Thus, both can be covered by *ns0:TwoStateDiscreteType*. Back to the two robots Niryo Ned and TurtleBot3 Waffle Pi, their status represents three states, so they must be covered by *ns0:MultiStateDiscreteType*.

C. Schema Forming

Forming a schema is the step that brings the semantic to a model by making relations of all elements. The concepts in the OPC UA information model language are linked due to references. Three basic reference types in our schema are *HasComponent*, *Organizes*, and *HasProperty*. Moreover, the references *HasTypeDefinition* and *HasSubType* appear in Subsection IV-A to define a new object or variable type. By default, information models provide not only the vocabulary, but also a schema with relations between the concepts. Thus, when reusing a schema, system designers have only three jobs: (1) define instances from the concepts, (2) put the defined instances in appropriate positions in the schema, and (3) add details to complete it. Since the skeleton of the OPC UA information model for the PAL monitoring system is CS-DI and CS-Robotics, our designers can adopt the above strategy.

Figure 3 is an excerpt of the blueprint of our final OPC information model design. The `DeviceSet` instance predefined by CS-DI is the root for all device instances. In this case, there are two instances: `Niryo_Ned` representing Niryo Ned and `TurtleBot3_Waffle_Pi` representing TurtleBot3 Waffle Pi. As every MotionDeviceType instance, `Niryo_Ned` has three groups of components which are `Controlers`, `MotionDevices`, and `SafetyStates`. Our designer defines the instance `Raspberry_Pi_4` as the controller of Niryo Ned. It has a microprocessor represented by the `CPU` instance. `Temperature` is one of the variable of the CPU's `ParameterSet`. In the `MotionDevices` group, our designers define `Robot` to represent the mechanical and electronic parts of the Niryo Ned. The instance has many sub-elements, of which three are essential for this scenario. First, the `ParameterSet` has the component which is the `Status` variable. Second, the `Axes` contains all the joint instances of Niryo Ned. Third, the `AdditionalComponents` can contain all additional modules of Niryo Ned. In this sense, all software components and their corresponding variables are in this folder. The organization of the sub-elements of `TurtleBot3_Waffle_Pi` shares the same design logic as the one of `Niryo_Ned`, as explained in Section IV-B. Also in this design, our designers consider the conveyor belt and the storage as additional components of Niryo Ned, and the park as an additional component of TurtleBot3 Waffle Pi.

All variables in our schema are described with properties except `CountNumber`. Every `ActualPosition` associates with an `EngineeringUnits` and a `EURange`. The former property holds the value which is the unit identification corresponding to mrad. The latter one holds the minimum and maximum rotation angle values. Every `Status` initiated by *ns0:TwoStateDiscreteType* has two properties of two states `FalseState` and `TrueState`. However, the `Status` initiated by *ns0:MultiStateDiscreteType* links to a property `EnumStrings` which associates with three integer values corresponding to available, busy, and offline states. `Temperature` and `EnergyConsumption` variables have `EngineeringUnits`, with the units respectively are °C and J. One essential addition to `Temperature` variables are a new property, called `RepeatIntervalMs`, which implies the frequency that the variables is updated. Our designers define this property to distinguish historical data from other real-time data.

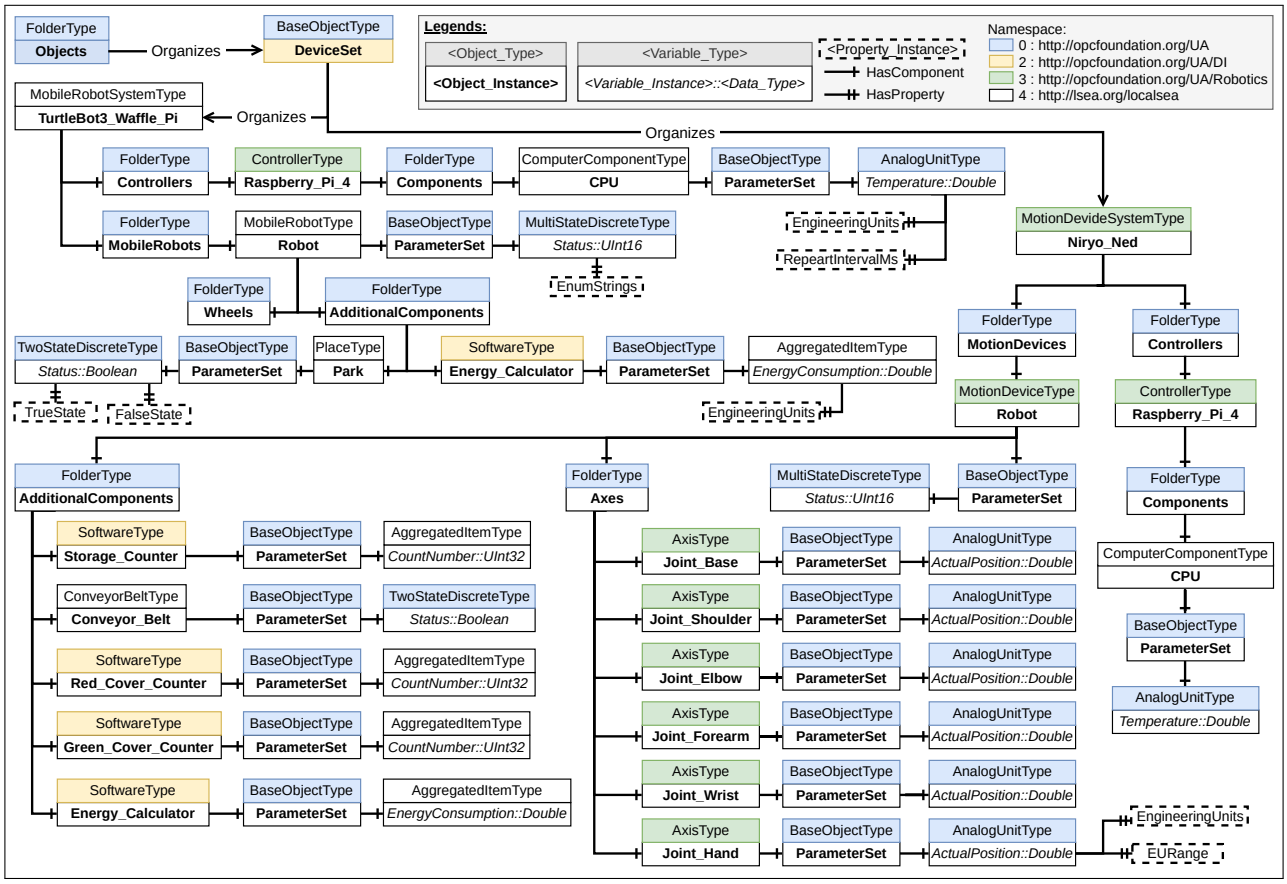

Fig. 3. Excerpt of the blueprint of the OPC UA information system for the PAL monitoring case study

V. Conclusion and Discussion

This paper presents the experiences of designing an OPC UA information model from a requirement specification. The presented PAL monitoring system relies on a case study deployed in the robotic testbed LocalSEA. The vocabulary used to model the system relies mainly on the standard information model, CS-DI, and CS-Robotics. The sharing experiences are helpful to system developers who are dealing with a complex OPC UA-based industrial system engineering. Unfortunately, limited by the case study, this paper has no chance to share experiences in modeling OPC UA methods and views.

There are three further points to discuss. First, it is worth recalling that the blueprint in Subsection IV-C is only a part of the complete OPC UA information model. Since this paper focuses on highlighting the transformation from requirements into design, therefore, avoids presenting other unrelated parts.

Second, the conveyor belt is an additional component of Niryo Ned as in our design, or can be modeled as an independent device. The choice depends on the design style and the case study. For example, the second solution is better when multiple robotic arms share the same conveyor belt.

Third, the serialization from the OPC UA information model design into a system modeling format is out of the scope of this paper. In reality, our designers develop and use an extension for Eclipse Papyrus to encode the design into a SysML-based model and then generate code for OPC UA servers. We plan to share this extension with the public in the future.

References

[1] OPC Foundation, "OPC Unified Architecture - Part 3: Address Space Model," Industry Standard Specification OPC 10000-3, 2017.

[2] OPC Foundation, "OPC Unified Architecture - Part 5: Information Model," Industry Standard Specification OPC 10000-5, 2017.

[3] F. Pauker, T. Frühwirth, B. Kittl, and W. Kastner, "A Systematic Approach to OPC UA Information Model Design," *Procedia CIRP*, vol. 57, pp. 321–326, 2016.

[4] S. Rohjans, K. Piech, and S. Lehnhoff, "UML-based modeling of OPC UA address spaces for power systems," in *2013 IEEE International Workshop on Inteligent Energy Systems (IWIES)*. Vienna, Austria: IEEE, Nov. 2013, pp. 209–214.

[5] M. Poveda Villalón, Q.-D. Nguyen, C. Roussey, C. de Vaulx, and J.-P. Chanet, "Ontological Requirement Specification for Smart Irrigation Systems: A SOSA/SSN and SAREF Comparison," in *9th International Semantic Sensor Networks Workshop, ISWC 2018*, vol. 2213, Monterey, United States, Oct. 2018, pp. 1–16.

[6] Q.-D. Nguyen, F. Tmar, Y. Huang, and S. Dhouib, "Early lessons learned from the development of a local OPC UA-based robotic testbed for research," in *IEEE 31st International Symposium on Industrial Electronics*, Anchorage, Alaska, United States, Jun. 2022, pp. 1–4.

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

[8] N. T. Thomopoulos, *Assembly Line Planning and Control*. Cham: Springer International Publishing, 2014.

[9] OPC Foundation, "OPC Unified Architecture - Part 100: Devices," Industry Standard Specification OPC 10000-100, 2021.

[10] Mechanical Engineering Industry Association (VDMA), "OPC UA for Robotics - Part 1: Vertical Integration," VDMA, Industry Standard Specification VDMA 40010-1, 2019.