



HAL
open science

Bridging the Gap between SysML and OPC UA Information Models for Industry 4.0

Fadwa Rekik, Saadia Dhouib, Quang-Duy Nguyen

► **To cite this version:**

Fadwa Rekik, Saadia Dhouib, Quang-Duy Nguyen. Bridging the Gap between SysML and OPC UA Information Models for Industry 4.0. *The Journal of Object Technology*, 2023, 22 (2), pp.1-15. 10.5381/jot.2023.22.2.a14 . cea-04169475

HAL Id: cea-04169475

<https://cea.hal.science/cea-04169475v1>

Submitted on 24 Jul 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0
International License

Bridging the Gap between SysML and OPC UA Information Models for Industry 4.0

Fadwa Rekik*, Saadia Dhouib*, and Quang-Duy Nguyen*

*Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

ABSTRACT

SysML is a UML profile used for multi-disciplinary systems engineering. The rise of Industry 4.0 modernizes industrial systems with new technologies and consequently demands the SysML language to be extended to cover new specific concepts. Among these new technologies, Open Platform Communication Unified Architecture (OPC UA) is a highly recommended standard for the interoperability of industrial systems. OPC UA offers a data modeling mechanism that can represent an industrial system's resources in the form of an OPC UA information model. Technically, the OPC UA information model is the combination of the basic information model proposed by the OPC Foundation and other companion specifications' information models related to the system's specific domains. When refining a SysML design for such an OPC UA-based system, it can be challenging to enrich the design model and integrate all the information from a companion specification. This paper aims to bridge the gap between SysML engineering environments and OPC UA information models with two contributions. First, we propose to extend SysML with a new UML profile corresponding to the OPC UA Robotics companion specification released by the joint working group between the OPC Foundation and the Mechanical Engineering Industry Association. Second, we share our approach to automatically generating OPC UA information models from high-level SysML design models. The two use cases in our robotic cell, also presented in this paper, show the importance of this research in practice.

KEYWORDS SysML, UML profiles, OPC UA information models, Model transformation, QVTo, Industry 4.0, VDMA Robotics.

1. Introduction

One of the prime requirements of Industry 4.0 is the interoperability between industrial systems. Interoperability means that a system must be able to communicate and collaborate with others, regardless of their differences in manufacturers or functions. Regarding this need, OPC UA (Mahnke et al. 2009), standing for the Open Platform Communication Unified Architecture, emerges as a strong solution. Indeed, it facilitates data exchange and management to meet all the requirements of today's automation without relying on individual suppliers.

OPC UA is a set of standards developed and maintained by

the OPC foundation. Its goal is to allow information to be reliably and securely exchanged between platforms from different vendors and to allow seamless integration of those platforms. In detail, it offers technical interoperability between industrial systems by defining a set of well-structured communication protocols. Also, OPC UA provides semantic interoperability with a common metamodel that can be used to model information of all industrial system types. The latest version of the OPC UA specification, version 1.05, has more than 24 parts¹, and each part reflects an aspect of the specification (OPC Foundation 2017). This paper focuses on the address space and information model parts. The address space part provides a standard way for a server to represent its industrial system's resources to clients in the form of OPC UA nodes. The information model part provides a basic OPC UA information model vocabulary, which contains the most fundamental OPC UA nodes for every OPC

JOT reference format:

Fadwa Rekik, Saadia Dhouib, and Quang-Duy Nguyen. *Bridging the Gap between SysML and OPC UA Information Models for Industry 4.0*. Journal of Object Technology. Vol. 22, No. 2, 2023. Licensed under Attribution - NonCommercial - No Derivatives 4.0 International (CC BY-NC-ND 4.0) <http://dx.doi.org/10.5381/jot.2023.22.2.a14>

¹ <https://reference.opcfoundation.org/>

UA information model. This basic OPC UA information model can be called the base information model. Moreover, the OPC Foundation and its partners provide many extensions for the base information model. The extensions are called Companion Specifications (CS). Each CS defines an extended vocabulary for specific industries, devices, or use cases. Technically speaking, a system's OPC UA information model is the combination of the base information model and the CSs' information models related to the system's specific domains. The OPC UA modeling language, as described in this paper, refers to the mechanism utilized to standardize an OPC UA information model. It encompasses the address space component and incorporates the vocabularies provided by the OPC Foundation and its partners, as well as the information model part and CSs.

The OPC UA information model is undeniably a core component of every OPC UA-based system. Unfortunately, when the complexity and number of industrial systems grow along with the rise of Industry 4.0, OPC UA information model implementation becomes challenging while applying the OPC UA standard. First, the OPC UA modeling language is complex and non-visualized; thus, it is uneasy to learn and use, especially for new modelers who lack experience in this language. Second, the resources of an industrial system may grow over time; consequently, its OPC UA information model must evolve. However, designing and deploying an OPC UA information model is time-consuming, and guaranteeing data model consistency between different design versions are sometimes challenging. Regarding the above challenges, using Model-Driven Engineering (MDE) and Domain-Specific Languages (DSL) is promising to reduce the complexity of modeling OPC UA information models (Goldschmidt & Mahnke 2012). MDE is a development methodology that proposes to use a DSL to analyze and model different aspects of a specific domain as conceptualized models (Stahl et al. 2006). Once the conceptualized models are stable, they can be converted into the other formats required by users.

The strong point of the MDE approach is that DSLs are usually simple and visualized, making them easily understandable to all stakeholders involved in the project. Two languages commonly used in MDE are Unified Modeling Language² (UML) and System Modeling Language³ (SysML). Both are standards developed and maintained by the Object Management Group (OMG). Technically, SysML is a profile of UML version 2 (UML 2) and is specified for system engineering. It provides a rich set of abstractions and notations that can be applied to a wide range of domains.

This paper follows MDE and uses SysML to create high-level design models for OPC UA-based systems. While OPC UA is a widely recognized standard primarily in the industry domain, UML and SysML are popular across all domains. Therefore, it is more convenient for a novice modeler to develop a SysML design model first and then produce an OPC UA information model rather than starting from scratch to learn the OPC UA modeling language. Moreover, the modeler can easily share their designs with others, thus, improving the data model consistency between different industrial systems or between different

versions of an industrial system. This work presents a novel approach for refining SysML models using OPC UA information models, thus bridging the gap between the two worlds. By leveraging the advantages of SysML, our approach enables a more structured and formal representation of the system components, information flows, interfaces, and relationships, resulting in a more precise and rigorous system modeling. In contrast, OPC UA models lack the global vision of the system and its interactions. Therefore, our approach can potentially lead to significant time and effort savings in the development of OPC UA models. These findings provide valuable insights for systems engineers who seek to optimize their modeling practices.

SysML is open, flexible, and well-supported; however, the vocabulary of SysML is insufficient to encompass all the concepts covered by OPC UA information models. Indeed, an OPC UA information model may contain one or several domain-specific CSs. An approach to resolve this problem is to develop UML profiles to extend SysML with new vocabulary and modeling rules for the CSs. This paper particularly focuses on the CS for Robotics (CS-Robotics) proposed by the joint working group between the OPC Foundation and the Mechanical Engineering Industry Association (VDMA) (VDMA 2019). Note that this choice is motivated by some actual projects at CEA List. In this sense, we propose to extend SysML models with the UML profile covering the metadata provided by CS-Robotics. Also, we share our strategy to automate the transformation of SysML models into OPC UA information models using model-to-model (M2M) transformation techniques.

The remainder of this paper is organized as follows. Section 2 captures the mandatory background of our research, which are UML, the profile SysML 1.6, and the OPC UA modeling language. Section 3 outlines our MDE approach to turn a SysML design model into an OPC UA information model. This section also reveals the steps to serialize an OPC UA information model into a binary file that runs on an OPC UA server. Section 4 introduces the UML profile for CS-Robotics. Section 5 describes our mapping and transformation rules from SysML models to OPC UA information models. Section 6 details the steps of our methodological approach. Section 7 presents the strategy to evaluate and maintain our work, including two use cases applying our contribution in practice. Next, some related works are discussed in Section 8. Finally, Section 9 concludes this paper with a discussion and an outlook on future works.

2. Background

The M2M transformation proposed by this paper receives SysML design models as input and produces OPC UA information models as output. First, the UML and its profile SysML 1.6 is the language to encode the input design models. Second, the OPC UA modeling language encodes the output results. This section respectively presents them in two following subsections.

2.1. UML and the Profile SysML 1.6

UML 2 identifies 14 diagrams to visualize, specify, construct, and document different aspects of the software. Each diagram has a set of elements, including the core elements and the ele-

² <https://www.omg.org/spec/UML/>

³ <https://www.omg.org/spec/SysML/>

ments specified for the diagram. Fig. 1 shows the UML metamodel including the core elements. UML has one metaclass **NamedElement** from which all other elements, such as **Class**, **Datatype**, **Operation**, and **Property**, are subtyped. The connections between the elements are UML Relationships. They are direct or indirect instances of the type **Relationship**. Their job is to connect elements and describe the relationship between elements. For example, there exists the UML Relationship **ownedOperation** between **Operation** and **Class**, and the UML Relationship **ownedAttribute** between **Property** and **Class**. Moreover, **MultiplicityElement** supports **Relationship** by defining the number of elements that participate in a relationship, the order, and the uniqueness.

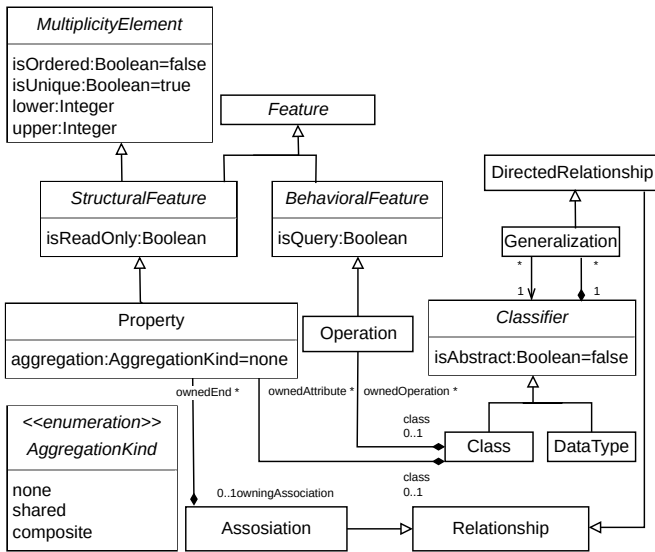


Figure 1 Excerpt of UML Metamodel

SysML is a profile of UML 2. On one hand, SysML inherits the relatively mature notation and widely-accepted semantics of UML. On the other hand, it has been intentionally designed to be less software-centric than UML and instead places greater emphasis on systems modeling. In detail, while UML involves modeling classes and addressing software engineering, SysML involves modeling blocks and provides the vocabulary for system engineering (Santos & Soares 2023). Technically, SysML reuses a subset of UML 2 and provides additional extensions addressing requirements for systems engineering. The subset is called UML4SysML, and the extensions are called SysML’s extensions to UML (Casse 2017). The latest version of SysML is 1.6. Fig. 2 illustrates the relation between UML 2 and its profile SysML 1.6. SysML supports designing, analyzing, and verifying complex systems, including software and hardware components. This language allows modeling various systems from different perspectives: behavioral, structural, para-metrical, or required views.

This work uses only two SysML diagrams. The first is the Block Definition Diagram (BDD), where components are modeled using the UML Stereotype Blocks. The second diagram is the Internal Block Diagram (IBD), where the internal structure of a Block is modeled using Ports, Parts, and Connections.

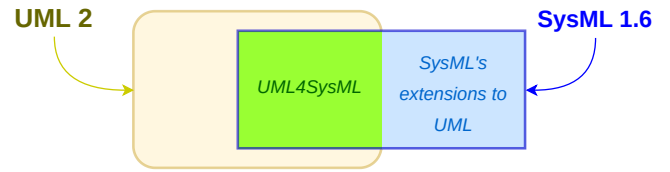


Figure 2 Relation between UML 2 and SysML 1.6

Fig. 3 illustrates the metamodel of Blocks and Ports. Blocks may include both structural and behavioral features, such as Properties and Operations. The relationships between Blocks can be modeled with UML Relationship instances of **Association** and **Composition**. Ports are connection points between a Block and its external entities. Also, in IBD, Blocks are extended to support **FlowProperties**, which specify the items exchanged between a Block and its environment. An example of such items is data flow.

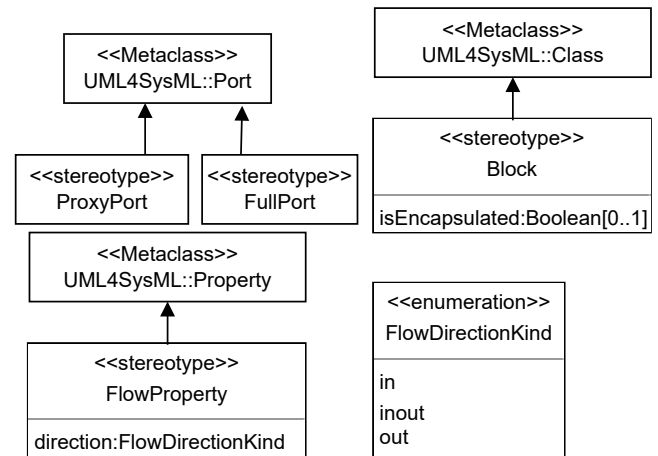


Figure 3 Block and Port metamodel of SysML 1.6

2.2. OPC UA Modeling Language

OPC UA provides a framework representing a system’s resources as a set of OPC UA nodes logically stored in an OPC UA address space. The set of OPC UA nodes is also called a nodeset. Each OPC UA node is an instance of a NodeClass. A NodeClass is a template containing information about attributes, references, and standard properties that its instances can represent. An attribute reveals a quality or characteristic of a node; a reference connects the node with another; and a standard property describes the node’s metadata. NodeID, BrowseName, DisplayName, and Description are four essential attributes of every NodeClass. The common NodeClass types are **Object**, **ObjectType**, **Variable**, **VariableType**, **Datatype**, and **ReferenceType**. **Variable** represents a value that can be read or written from the server. A **Variable** node has an association to a **Datatype** instance that defines the data type, such as a string, integer, and structure, of its attribute Value. **Object** and **Variable** respectively have a reference *HasTypeDefinition* to **ObjectType** and to **VariableType**. A reference is an instance of **ReferenceType**.

Technically, a nodeset is encoded in a NodeSet file with the XML format. Fig. 4 shows the base of the OPC UA metamodel encoded in the NodeSet file. In the encoding, all NodeClass have a prefix UA. For instance, the Object Type in the NodeSet file is `UAObjectType`. The root element is the `UANodeSetType`, which contains all nodes of the information model. All NodeClass types are direct or indirect subtypes of `UANode`. `UAObject` and `UAVariable` are a subtype of `UAInstance` and their instances are OPC UA instance nodes. Note that a NodeClass instance is not the same as an instance node. For example, a `UAObject` instance is an instance node; however, a `UAObjectType` instance is not an instance node.

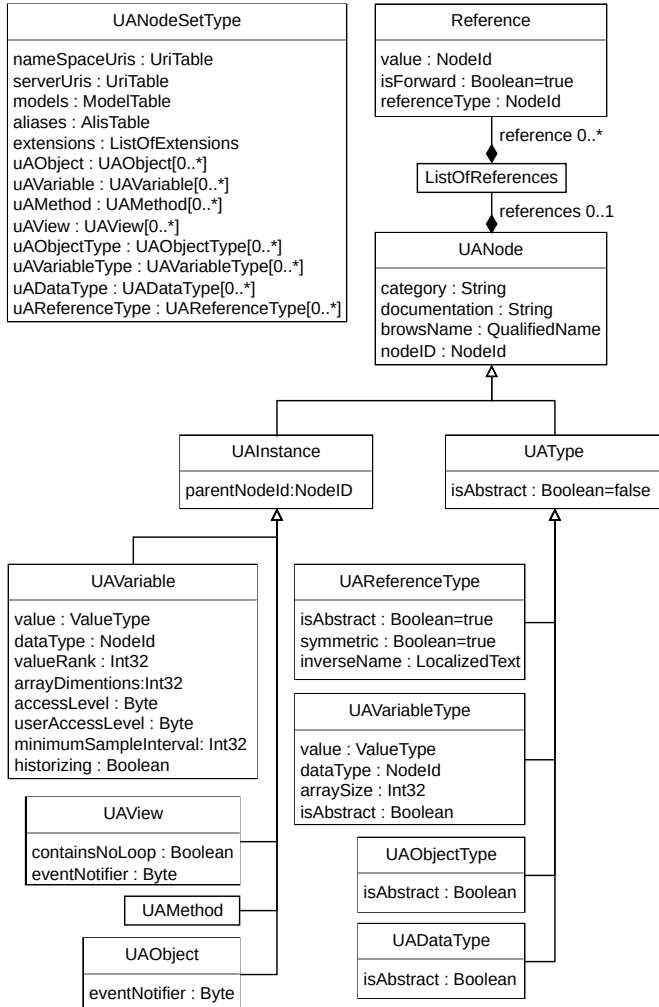


Figure 4 OPC UA metamodel in the NodeSet file

OPC UA Reference. The OPC Foundation proposes many built-in `ReferenceType` instances. A `ReferenceType` instance identifies the connection between a source node and a target node. The source node holds the `ReferenceType` instance and is the reference's starting point. The target node is the reference's ending point. For example, in the expression "X *HasTypeDefinition* Y": X is the source node that has the `ReferenceType` instance *HasTypeDefinition*, and Y is the target node. Note that the OPC UA modeling language does not

support the loop mechanism, which means that if the source node is X, thus the target node must be different from X. All `ReferenceType` instances are either direct or indirect subtypes of the instance *References* as shown in Fig. 5. Two types of references are *NonHierarchicalReferences* and *HierarchicalReferences*. Both of them are abstract. The subtypes of *NonHierarchicalReferences*, including *HasModelParent*, *HasTypeDefinition*, *GenerateEvent*, *HasEncoding*, *HasModelingRule*, and *HasDescription*, must be used by a node that cannot span a hierarchy. In other words, their target nodes have no references. *HierarchicalReferences* implies that its subtypes can span a hierarchy. Among these subtypes, only *HasChild* and *Aggregates* are abstract; thus, they cannot be used directly by a node. The rest are non-abstract. All `ReferenceType` instances have a different meaning and should be used in an appropriate context.

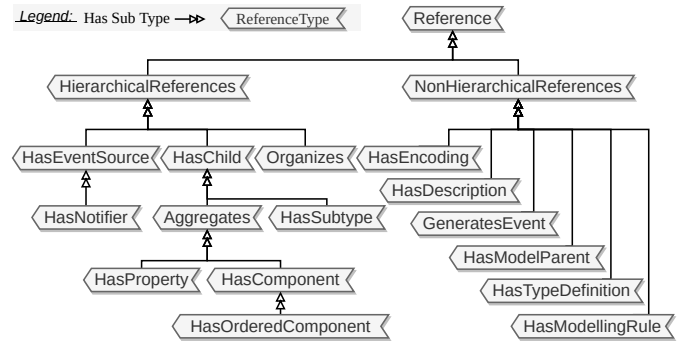


Figure 5 OPC UA ReferenceType instances

OPC UA Data Type. OPC UA defines 25 built-in `DataType` instances. They are subtypes of the instance `BaseDataType`, which also is an instance of `DataType`. Fig. 6 shows the hierarchy of all `DataType` instances. Among them, some correspond to primitive data types such as Boolean, Float, Integer, and String; some have a complex structure. For example, `NodeId` is a structured `DataType` instance composed of three elements: `NamespaceIndex`, `IdentifierType`, and `Identifier`.

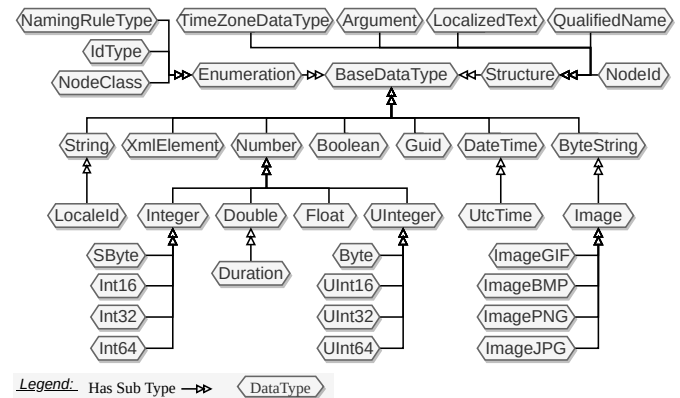


Figure 6 OPC UA DataType instances

OPC UA Namespaces. A namespace is a logical domain of OPC UA nodes. Using namespaces implies two advantages. First, it is practical when grouping all OPC UA nodes with

the same category into one namespace. Second, two OPC UA nodes, even with the same `BrowserName` and `Identifier`, are different if they are in two different namespaces. Thus, it prevents overlapping issues while combining different information models into a single coherence OPC UA information model. An OPC UA namespace has a globally unique string called a `NamespaceURI`, and a locally unique integer called a `NamespaceIndex`. The `NamespaceIndex` of the base information model is always 0, and its `NamespaceURI` is always <http://opcfoundation.org/UA/>. Other CSs of a system's OPC UA information model can have the `NamespaceIndex` value starting from 2. The index 1 is reserved for the OPC UA server.

3. A Model-Driven Engineering Approach for the OPC UA Information Model Development

MDE is a widely-adopted approach to fostering abstraction and coping with complexity. It is based on two principles: Abstraction and Automation (Mohagheghi & Dehlen 2008). First, Abstraction is about representing systems in the form of models. Models are representations on various abstract levels of a system's structure and behaviors. Each model is based on a formalism. The chosen formalism depends on the concerns and goals of the modeler, as well as on the system to model. Second, Automation means building software automatically from high-level models. The purpose is to transform a general-purpose system model into a platform-specific one. The automation generation must guarantee the conformance between the source and the target models.

Our MDE approach for Industry 4.0 systems is based on Abstraction and Automation. As shown in Fig. 7, this approach is divided into two main steps. The first step consists in using SysML as a modeling language for designing the systems. We choose SysML version 1.6 as a modeling language since it is a standard language recognized by academics and industry (Wortmann et al. 2020). These SysML models can be enriched with new UML profiles that cover specific domains. In this work, we implemented a UML profile to refine the SysML model with the notions specified for CS-Robotics. However, other CSs, such as Programmable Logic Controllers (PLC) and End of Arms tools, could be covered with the same approach.

The second step is the automatic code generation from SysML designs into OPC UA information models. It is a promising way to save time and cost for system development. The M2M transformation is performed using the Query/View/Transformation operational (QVTo) language (OMG 2011), a widely used standard for model transformation due to its powerful and flexible language, widespread tool support, and adoption by the OMG's Model-Driven Architecture (MDA) initiative (Höppner et al. 2022). First, we define rules to implement the mapping between source model elements into target model elements. Then, we develop a QVTo code generator that allows us to create OPC UA information models from SysML models automatically. Both the profile and the QVTo code generator are implemented as an extension to the open-source UML modeling tool Papyrus⁴.

⁴ <https://www.eclipse.org/papyrus/>

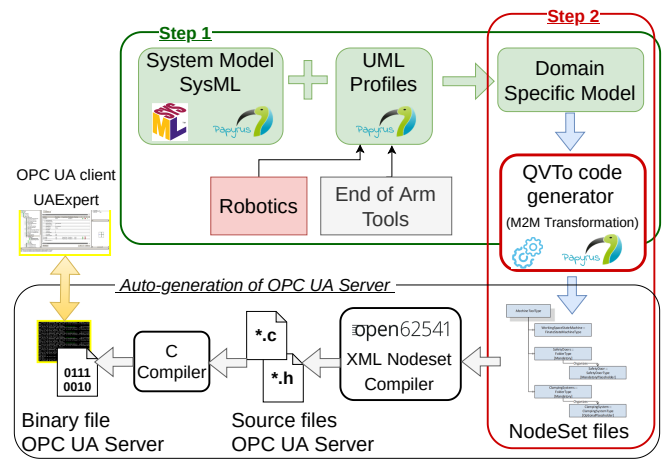


Figure 7 Model-driven tool-chain architecture

The Papyrus's extension is part of an automatic generation chain of the OPC UA server. This chain of automation can be called the server auto-generation process. Each time the SysML model is modified, such as adding a new component and variable, the OPC UA information model stored in a NodeSet file is automatically generated. As shown at the bottom of Fig. 7, the new NodeSet file is used as the input for the server auto-generation process. In our work, we use the open-source library open62541⁵ and its toolset to convert XML NodeSet format into C source code format files. Together with other existing C source code files, these files are then compiled to produce a binary file. Once launched, OPC UA clients, such as UaExpert⁶ can connect and access the OPC UA server.

4. UML Profile for CS-Robotics

This work focuses on the CS-Robotics depicted in Fig. 8. The goal of its authors, the OPC Foundation and VDMA, is to release CS-Robotics as a vocabulary that can describe various robot types in the industry. However, its current version, part 1, supports only robotic arms. It is composed of four groups of concepts: (1) the motion device or physical aspect of a robot, (2) the controller or the software aspect of a robot, (3) the safety state, and (4) the task control or working program of a robot. All of them constitute a motion device system. Technically, CS-Robotics reuses concepts from the CS for Devices (DI), a vocabulary to model industrial devices (OPC Foundation 2021). In turn, DI reuses concepts on the base information model. In other words, CS-Robotics depends on DI, and DI depends on the base information model. Moreover, the base information model extends the UML metamodel. Fig. 9 illustrates these dependencies. Note that the OPC UA profile in this figure corresponds to the base information model. Thus, a UML profile for CS-Robotics must inherit elements of the DI and the base information model profiles.

⁵ <https://open62541.org/>

⁶ The UaExpert is a general purpose test client supporting OPC UA features such as Data Access, Alarms & Conditions, and Historical Access. It is available at: <https://www.unified-automation.com/downloads/opc-ua-clients.html>

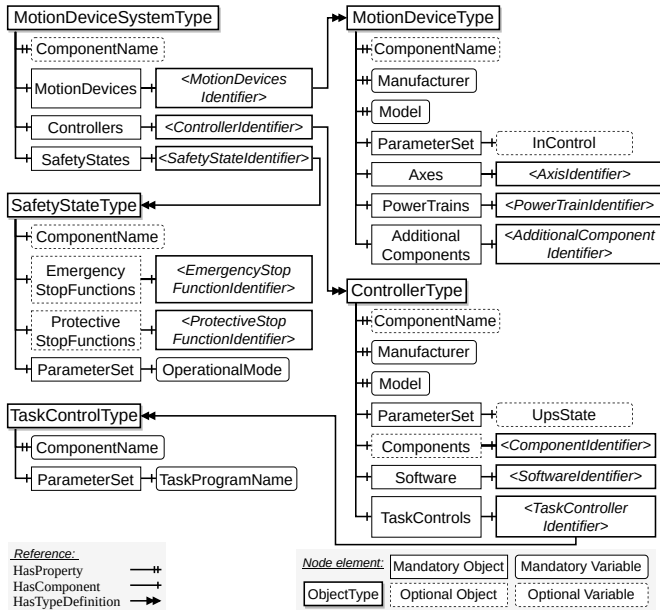


Figure 8 The basic concepts of CS-Robotics

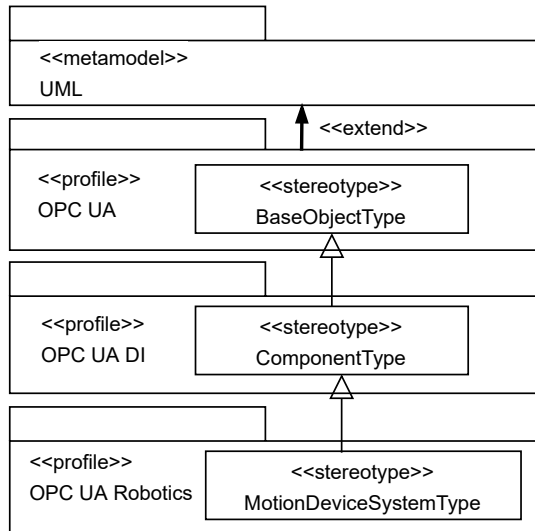


Figure 9 Profiles Architecture of CS-Robotics

The UML profile for CS-Robotics is depicted in Fig. 10. This profile extends the SysML model with specific information from CS-Robotics. The following points are remarkable.

First, each `ObjectType` instance is translated into a corresponding UML Stereotype. Also, all relationships between `ObjectType` instances are respected and translated into relevant UML Relationships. The following example clarifies this point. `MotionDeviceSystemType` is a subtype of `ComponentType`, which is an `ObjectType` instance from DI. This relationship is translated into a UML Generalization between the UML Stereotype `MotionDeviceSystemType` and `ComponentType`.

Second, the `ParameterSet` of an `ObjectType` instance which contains a flat list of parameters, is translated into a UML `DataType`. For example, in CS-Robotics, the `AxisType` *Has*-

Component `ParameterSet`. This `ParameterSet` is translated into the UML `DataType` `ParameterSetAxisType`.

Third, each Enumeration data type in CS-Robotics is translated into a UML Enumeration. For example, the `DataType` instance `AxisMotionProfileEnumeration` is translated into the UML Enumeration `AxisMotionProfileEnumeration`.

Finally, the UML Multiplicity concept is used to describe the `Object` instances that have a reference *HasTypeDefinition* pointing to `ModellingRuleType` and `FolderType` in OPC UA modeling language. The multiplicity of a UML Property can have a lower and high bound. The lower bound equals 1 when an `Object` instance has the reference *HasModellingRule* pointing to an `Object` instance `Mandatory`. The lower bound equals 0 when an `Object` instance has the reference *HasModellingRule* pointing to an `Object` instance `Optional`. For example, the `PowerTrainType` in CS-Robotics has the reference *HasComponent* pointing to `Motor` and `Gear`. Since `Motor` is mandatory; thus, the UML Stereotype `PowerTrainType` has a property `Motor` with the low bound equaling 1. However, `Gear` in CS-Robotics is optional; thus, the property `Gear` in UML has the low bound equaling 0. The high bound equals "many" (*) to represent in CS-Robotics a `FolderType` instance containing multiple `Object` instances with the same type. For example, in CS-Robotics, `MotionDeviceType` has the reference *HasComponent* pointing to `Axes`, which has the type definition `FolderType`. Thus, the UML Stereotype `MotionDeviceType` has a property `Axes` with a high bound "many".

5. QVTo Transformation

This section describes the M2M transformation from SysML design models to OPC UA information models using QVTo. We use QVTo⁷ as transformation language because it is compatible with UML. Also, it is widely adopted in both academia (Lee et al. 2017; Commission et al. 2007) and industry (Gerpheide et al. 2016). QVTo transforms models conforming to one or several metamodels. The transformation declaration specifies the source and target metamodels. Fig. 11 illustrates the QVTo transformation process. It involves two levels: the metamodel level and the model level. The metamodel level defines a set of transformation rules that works with the mappings between the input and the output metamodels⁸. In detail, it takes as input the UML, SysML, and CS-Robotics metamodels. The output is an OPC UA information model and a CSV metamodel. The CSV metamodel contains a simple list of the generated nodes with their corresponding `NodeId` and `NodeClass` information encoding in the comma-separated values format. After defining the mappings between the input and the output metamodels, the transformation is executed at the model level to generate XML and CSV files from SysML design models.

This section has two subsections. We first describe the transformation from a SysML model to an OPC UA information model. Then, we depict the transformation of a SysML model enriched by the profile CS-Robotics.

⁷ <https://www.omg.org/spec/QVT>

⁸ Ecore is a metamodel included in the Eclipse Modeling Framework (EMF). It is most often used as a meta-language to define languages.

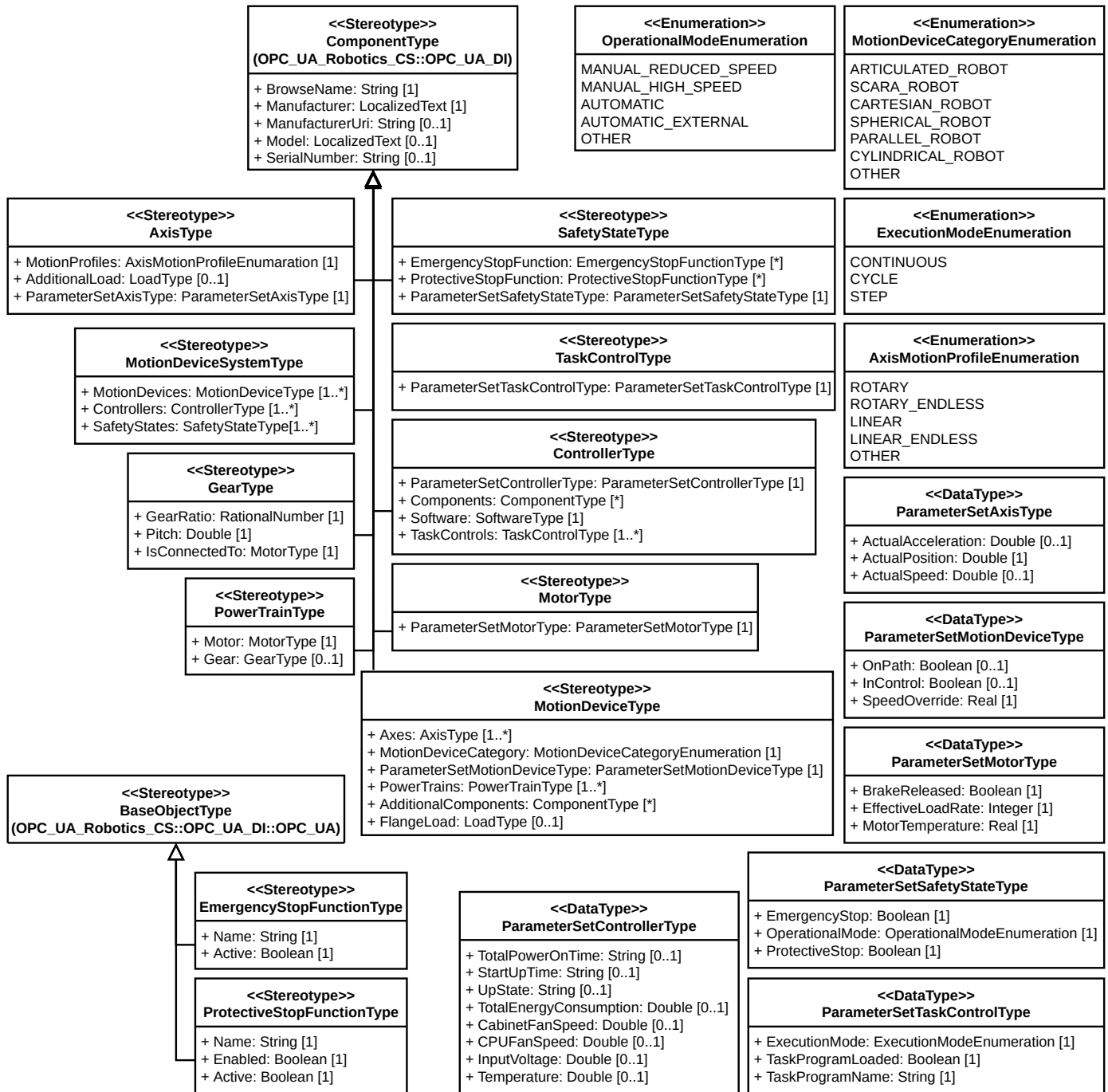


Figure 10 Excerpt of the UML profile for CS-Robotics

5.1. Transformation Rules from SysML to the OPC UA Information Model

SysML Model to UANodeSetType. Fig. 12 shows the details of the transformation. The transformation begins from the root model element or any selected UML Element. In both cases, we map the selected element to an instance of `UANodeSetType`. Then, we create the `namespaceUris` defining an OPC UA `UriTable`, and add the `NamespaceURI` of the base information model to the table by default. If the CS-Robotics profile is applied to the model, we must also add the DI and CS-Robotics

`NamespaceURI`. The resulting XML code is as follows.

```

1 <NamespaceUris>
2 <Uri>OrganicArchitecture</Uri>
3 <Uri>http://opcfoundation.org/UA/DI/</Uri>
4 <Uri>http://opcfoundation.org/UA/Robotics/</Uri>
5 </NamespaceUris>
  
```

`UANodeSetType` has an attribute containing the URIs and the version of the information model. Thus, we create an OPC UA `ModelTable` and add a `ModelTableEntry` element to it. The element has `modelUri`, a `publicationDate` of type

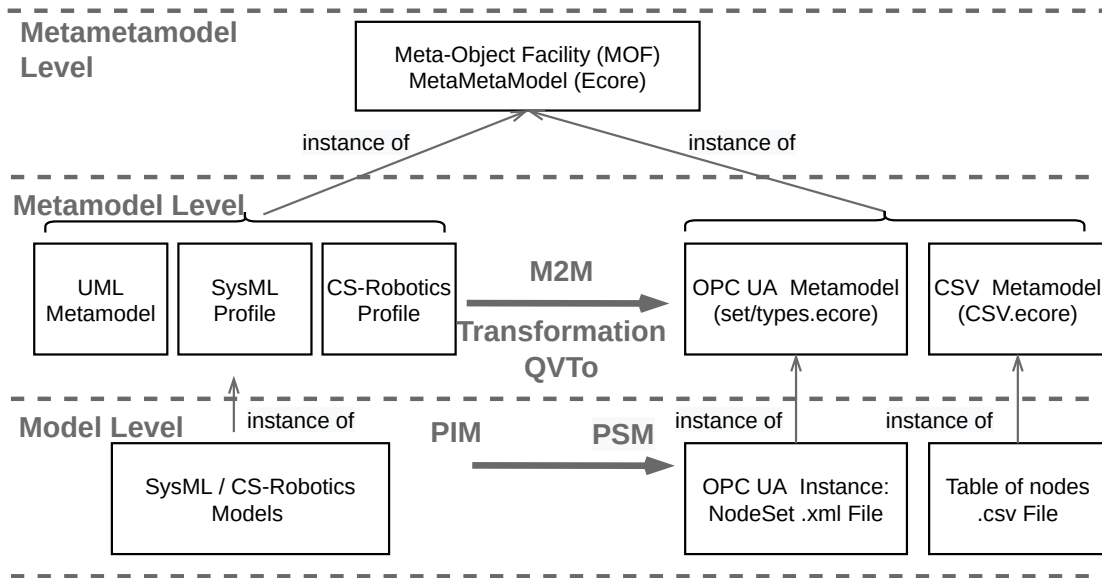


Figure 11 Transformation from SysML to OPC UA information model artifacts using QVTo

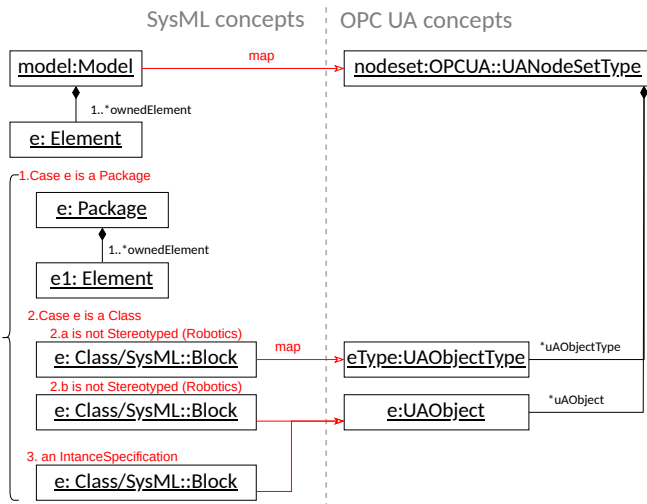


Figure 12 Transformation of a SysML Model

DateTime, and a version. The modelUri is assigned to be the namespace of the first SysML::Package, or a SysML::Model reached when we traverse the element tree upwards. Next, we add the required ModelTableEntry to the Model. The ModelTableEntry of the base information model is included by default. If the CS-Robotics Profile is applied, then the DI and the CS-Robotics ModelTableEntry are further added as required models. The following example of the QVTo code adds the CS-Robotics ModelTableEntry, where createDateTime is a method to create a DateTime from a String.

```

1 modelTableEntry.requiredModel+= object
2 OPCUA::ModelTableEntry{
3   modelUri:="http://opcfoundation.org/UA/Robotics/";
4   publicationDate:=createDateTime("2019-01-23T00:00:00Z")
5   .oclAsType(DateTime);
6   version:="0.93" };

```

The resulting XML code of the transformation is as follows. Note that "OrganicArchitecture" is the name of the super package of the selected element.

```

1 <Models>
2 <Model ModelUri="OrganicArchitecture"
3   PublicationDate="2022-01-17T15:44:15.200+01:00"
4   Version="1.0.0">
5   <RequiredModel ModelUri="http://opcfoundation.org/UA/"
6     PublicationDate="2020-07-15T00:00:00Z" Version="1.04.7"/>
7   <RequiredModel ModelUri="http://opcfoundation.org/UA/DI/"
8     PublicationDate="2020-06-02T00:00:00Z" Version="1.02.2"/>
9   <RequiredModel ModelUri="http://opcfoundation.org/UA/
10     Robotics/"
11     PublicationDate="2019-01-23T00:00:00Z" Version="0.93"/>
12 </Model>
</Models>

```

As shown in Fig. 12, we have different mapping options corresponding to different types of the selected element.

- If the selected SysML element is a Package, then map the owned elements of the package.
- If the selected element is a Block or UML Class and is not stereotyped by CS-Robotics profile, then map the element to a UAObjectType.
- If the selected element is a Block and is stereotyped by the CS-Robotics profile: there is already a predefined UAObjectType for this element in the CS-Robotics, then consider the Block as an instance of this type. Thus, we map the stereotyped element to a UAObject.
- If the selected element is an InstanceSpecification, then map to a UAObject.

SysML Block to UAObjectType. Fig. 13 details the transformation from SysML Block to UAObjectType. This transformation is also applicable to UML Classes. To determine the supertype of the output, we need to check if the Block has a

UML Generalization instance. If it is the case, then the generalization is mapped to a *HasSubType* reference, and the supertype is the corresponding referenced *UAObjectType* instance. Otherwise, the supertype is *BaseObjectType*. Next, the properties of a *Block* are mapped as follows.

- If the property is typed by a UML *PrimitiveType* or an *Enumeration*, then it is mapped to a *UVariable* with a *HasComponent* reference pointing to the *UAObjectType* corresponding to the property's *Block*. Table 1 shows the mappings from UML to OPC UA data types. OPC UA defines several types that do not exist in UML, such as *Int16*, *UInt16*, or *Int32*. Thus, we implement a *SyML* library containing the required OPC UA primitive data types to enable the transformation.
- If the property is typed by a *Block* or if the association is a UML *Aggregation* instance, then the property is mapped to an *UAObject* instance, and the type is mapped to *UAObjectType* instance. A *HasComponent* reference is generated between the *UAObject* instance and the *UAObjectType* instance.
- If the property is typed by a *Block* and its *associationKind* is none, then a new reference is generated between the parent *UAObject* instance and the *UAObject* instance mapped from this *Block* typing the property. The generated reference is a subtype of *NonHierarchicalReferences*.

Table 1 Mapping of UML dataTypes

UML	OPC UA
Real	Double
Integer	UInteger
String	String
Enumeration	Enumeration
DataType	DataType

SysML Enumeration to OPC UA Enumeration. Each enumeration is mapped into a *UDataType* and a *UVariable*. The reason for this mapping relies on the mechanism of modeling enumeration of OPC UA modeling language. Indeed, an enumeration *UDataType* is defined with a list of field names, but they are just fixed attributes. The enumeration also needs to have a *HasProperty* reference to a *UVariable* *EnumerationStrings* containing the *ListOfLocalizedText* representing the list of fields to select. Note that this list contains the same items or less than the list of field names defined with enumeration. As shown in Fig. 14, the generated *DataType* has *Enumeration* as *HasSubType* reference and *HasProperty* reference to the generated *UVariable*. The generated *DataType* contains *DataTypeDefinition*, which contains fields (*DataTypeField*), each of which is the result of a mapping of a UML *EnumerationLiteral*. The value of the field is the value of the *literalInteger*. As same, we map each *EnumerationLiteral* to a *LocalizedText* inside the *ListOfLocalizedText*, modeling the variable's value.

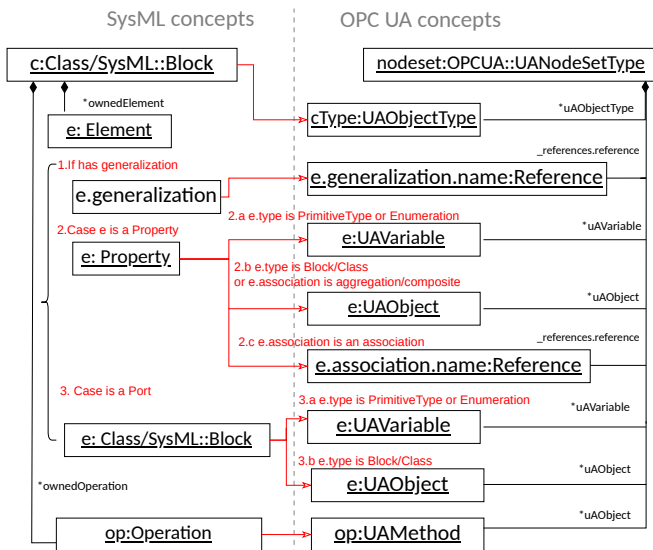


Figure 13 Transformation of UML Class

Ports is *Block*'s connection points that specify features available to the external entities via connectors. The type of a *Port* represents the type of data flowing through the *Port*. Ports inside a *Block* are mapped like properties to an *UVariable* if they are typed by a *PrimitiveType* or an *Enumeration*. If the *Port* is typed by another *Block*, then it is mapped to an *UAObject* instance, and a *HasComponent* reference is generated between this *UAObject* and its parent *UAObject*. Finally, *Operations* are mapped to *UAMethods*.

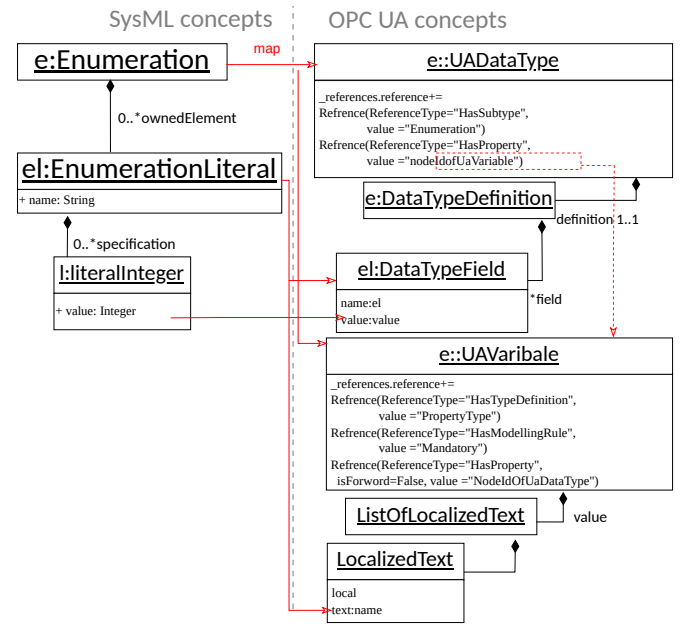


Figure 14 Transformation of UML Enumeration

5.2. Transformation from UML profile for CS-Robotics to OPC UA Information Model

Beforehand, it is worth noting that this section uses the example of a 6-axis robotic arm to illustrate the mapping from the UML profile CS-Robotics to an OPC UA information model. It is still correct for other robotic arms with more or fewer axes.

In CS-Robotics, `MotionDeviceSystemType` is the root element to model a motion device system, such as a robotic arm. The system modeler needs to create a Block stereotyped by `MotionDeviceSystemType` as shown in Fig. 15. A `MotionDeviceSystemType` instance contains at least one property typed by `MotionDeviceType`, `ControllerType`, and `SafetyStateType`. We apply the stereotype `MotionDeviceType` to the property typed by a `MotionDeviceType`, and we can assign corresponding values to its elements as in Fig. 16. If we have two different motion devices, we assign different properties for each one. Then for each `MotionDeviceType`, we need to specify the Axes, the PowerTrains, and other sub-components. These sub-components are added as properties. The 6-axis robot in Fig. 17 has six axes, six powertrains, and six motors. Since the profile has the same structure as the information model provided by CS-Robotics, the mapping is one-to-one.

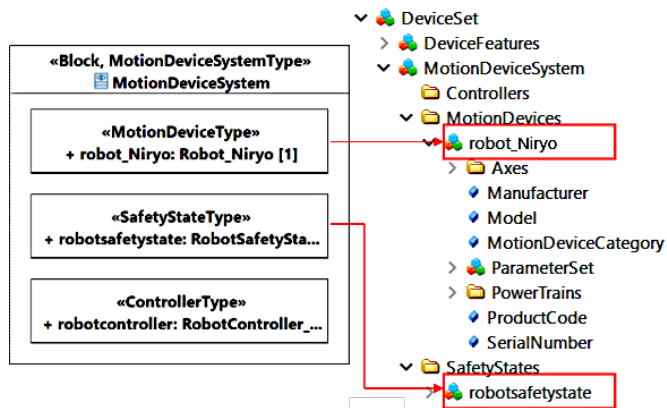


Figure 15 Mapping of MotionDeviceSystemType

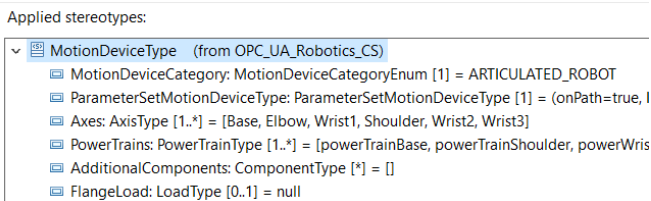


Figure 16 MotionDeviceType properties

6. OPC UA Modeling Rules

When using the SysML language to model the OPC UA connectivity information of an Industry 4.0 system, a system architect should follow a methodological approach that combines modeling rules coming from the SysML language and modeling rules coming from the OPC UA domain.

6.1. Block Definition Diagrams

The first step of the methodological approach is to create a BDD diagram and populate it with the SysML blocks that correspond to the main types of components composing the Industry 4.0 system. Blocks created in the BDD diagram can be simple SysML Blocks that will be transformed into `UAObjectTypes` as shown in Fig. 12. If system architects decide to refine the SysML Block by applying a stereotype coming from a CS UML profile, then modeling rules coming from the CS specification will be automatically applied since the UML profile implements these rules. For example, in the CS for Robotics, a `GearType` has three mandatory attributes: `GearRatio`, `Pitch`, and `isConnectedTo`. This constraint is captured in the profile (see Fig. 10). At the stereotype `GearType`, the attributes `GearRatio`, `Pitch` and `isConnectedTo` have a cardinality of exactly 1. By applying the stereotype `GearType`, and assigning values to the stereotype attributes, the Block will be correct by construction. If the system architect does not assign exactly one value to each attribute, it will get an error after triggering the model validation feature. The SysML model validation is required before executing the model transformation into an OPC UA NodeSet model.

6.2. Internal Block Diagrams

The second step of the methodological approach is to create an IBD diagram that reflects the internal structure of a SysML Block that was initially created in the BDD. If system architects decide to follow an OPC UA CS to model the internal structure, they have to follow the modeling rules provided in the specification. For example, in the CS for Robotics, a `MotionDeviceSystemType` is composed of at least a `MotionDeviceType`, a `SafetyStateType`, and a `ControllerType`. The IBD must follow this rule as shown in Fig. 15. For this, the system architects need to follow a user guide that could be implemented as an eclipse cheat sheet in the Papyrus Modeling tool. Additionally, all the structural modeling rules imposed by the CS have to be implemented as OCL constraints and triggered when validating the SysML model. The OCL constraint corresponding to the modeling rule of the `MotionDeviceSystemType` is shown as follows.

```

1 context CSRobotics::MotionDeviceSystemType inv
2   MotionDeviceSystemTypeComposition
3   let properties: Set(UML::Element) = self.base_Class.attribute
4   in properties->notEmpty()
5   implies
6     (properties->exists(p |
7       let appliedStereotypes = p.type.getAppliedStereotypes() in
8       appliedStereotypes ->exists(s | s.name = 'MotionDeviceType')
9     )) and
10    (properties ->exists(p |
11      let appliedStereotypes = p.type.getAppliedStereotypes() in
12      appliedStereotypes ->exists(s | s.name = 'SafetyStateType')
13    )) and
14    (properties ->exists(p |
15      let appliedStereotypes = p.type.getAppliedStereotypes() in
16      appliedStereotypes ->exists(s | s.name = 'ControllerType')
17    ))

```

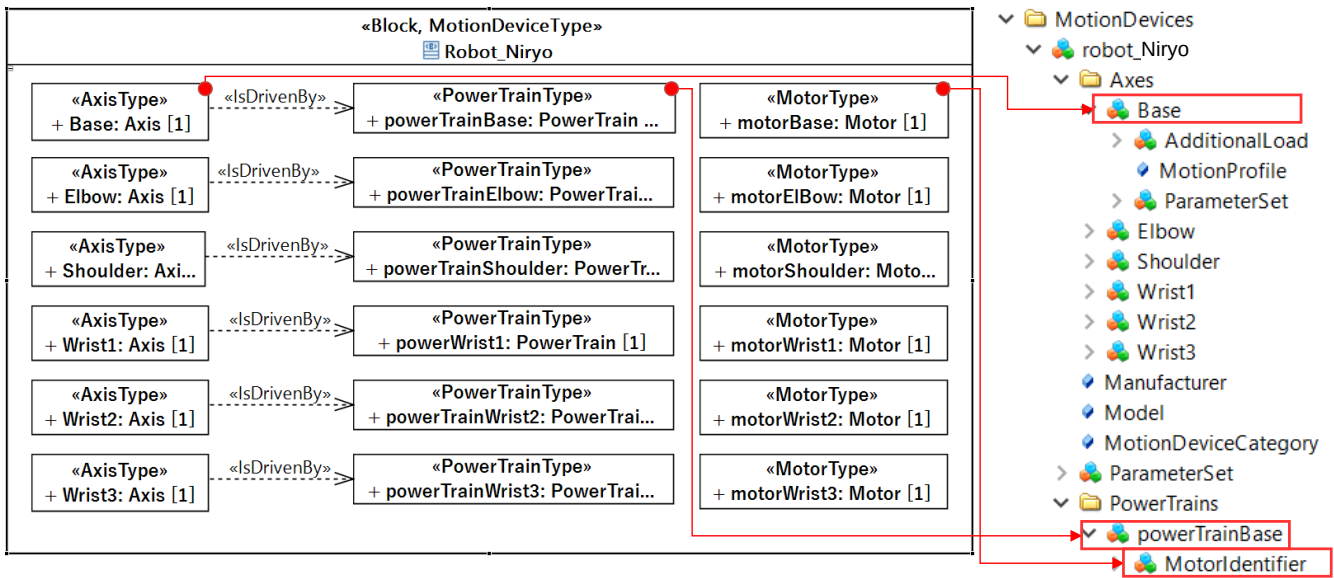


Figure 17 Mapping of MotionDeviceType

7. Testing and Maintenance

The strategy to evaluate our contributions has four levels: unit testing, integration testing, system testing, and issue tracker. While the first three testing methods are standard in traditional software development, the last is a part of the modern approach to maintaining and improving a product over time. The following subsections present them respectively.

7.1. Unit Testing of the Model Transformation

To test and validate our transformation, we prepare different input models:

- Models containing Blocks with Properties and Ports. In each test case, the input model has different input data types, which can be UML PrimitiveTypes, such as String, Double, Boolean, Enumeration, DataType, or OPC UA DataType, such as Int16 and Int32.
- Models with CS-Robotics stereotypes are Niryo Ned⁹ and UR5¹⁰. Both are 6-axis robotic arms.

For each input model, we first manually construct an expected output model using the OPC UA Modeler¹¹. Then we compare the generated information model with the expected one using existing XML comparison tools. We verify that the resulting model complies with the OPC UA specification; for example, all the ReferenceTypes are present. Also, the generated information model perfectly reflects the input SysML model. The expected and generated models have minor differences, such as the NodeId of the generated UAObject instances and the order of the XML tags in the NodeSet files. Such differences have no impact on the correctness of the tests.

⁹ <https://niryo.com/products-cobots/ned-six-axis-robot-arm/>

¹⁰ <https://www.universal-robots.com/>

¹¹ <https://www.prosysopc.com/products/opc-ua-modeler/>

7.2. Integration Testing with an OPC UA Server

The integration testing takes an OPC UA information model as the input and produces a running OPC UA server as the output. Our developers prepare a script, called UaServerGenerator, to automate the generation of the OPC UA server as the auto-generation process presented in Section 3. The script relies on the open62541 library and its tool NSCompiler. It runs only when the input OPC UA information model has a correct format; otherwise, an error appears. In other words, this tool helps to validate the OPC UA information model's syntax. Regarding the OPC UA information model's semantics, the OPC UA experts in our laboratory use UaExpert to manually visualize and evaluate the OPC UA information model.

Since the input of the integration testing is OPC UA information models from the unit testing, the number of integration test cases equals the number of unit test cases. To trace back if an error is from the OPC UA information model or another module of the OPC UA server, our developers read the log after running the script UaServerGenerator. Technically, the script's log relies on the log generated by NSCompiler and the open62541 library. Fig. 18 includes two logging error samples. The upper one shows an error in the information model. The second error relates to the network module of the OPC UA server.

7.3. System Testing: Two Use Cases in Testbed

The extension for Papyrus developed from this research is a strategic tool to develop the LocalSEA testbed (Nguyen, Rezik, et al. 2022), a robotic cell managed by CEA List. One fundamental component of the testbed is an OPC UA server, in which its OPC UA information model can be changed frequently according to the use case deployed in the testbed. The extension allows system developers to design OPC UA information models via a drag-and-drop user interface. Moreover, they can save their designs as different versions and reload them from one project to another.

Error from the OPC UA Information Model module

```
en62541/tools/nodeset_compiler/nodeset.py", line 137, in
    sanitize
    raise Exception("Reference " + str(ref) + " has an unknown target")
Exception: Reference ns=5;i=5081-- [ns=0;i=47]--ns=4;i=50810 has an unknown target
```

Error from another module of the OPC UA server

```
[2023-04-18 18:00:30.131 (UTC+0200)] error/server PubSub
Connection creation failed. Requested transport layer not found.
```

Figure 18 Logging error samples after running the UaServerGenerator script

This subsection presents two use cases using our contributions in its development. The first use case is to build a 3D digital twin, and the second is to deploy a product assembly line (PAL) monitoring system.

3D Digital Twin. The goal is to synchronize the movement of a physical robot with its 3D digital version. The robot used in this use case is Niryo Ned. Our developers created a tool, called RvizUA¹², to reuse the 3D Niryo Ned object proposed by the Niryo company to make a 3D digital twin. Between the physical and digital twins is an OPC UA server. It represents the resources of the physical Niryo Ned, including its six joints' position and rotation speed data, and publishes them to the 3D Niryo Ned. Based on such information, the 3D Niryo Ned can move synchronously with the physical Niryo Ned. Fig. 19 illustrates the architecture of the use case.

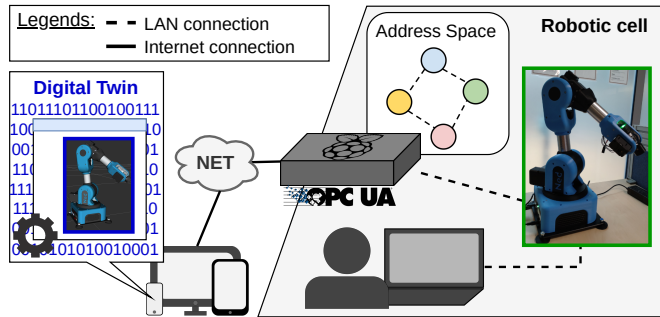


Figure 19 Architecture of the use case 3D Digital Twin

The development starts with analyzing the use case and producing the system functional design using SysML. As mentioned in 2.1, this work focuses only on the two diagrams IBD and BDD. The illustration on the left of Fig. 20 presents a global view of the robotic cell and its digital twin. The Block Cell represents the robotic cell, and the Block DigitalTwin represents the digital twin. Cell exchanges with DigitalTwin using connectors representing the data exchange between the physical and digital worlds. The illustration on the right of Fig. 20 presents the model of the robotic cell. It contains a sub-Block Robot_Niryo representing Niryo Ned. The connectors between the sub-Block and the Cell represent the data

exchange between the robot inside and components outside the cell. In this case, Robot_Niryo has six ports to exchange the actual_position data and six other ports to exchange the actual_speed data of the six joints.

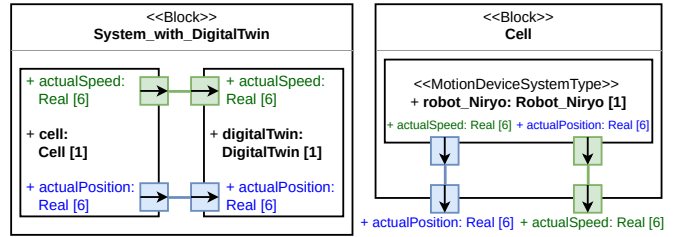


Figure 20 IBDs of the case study and the robotic cell

To recall, the SysML IBD model for Niryo Ned, presented in Section 5.2, is already illustrated in Fig. 17. Using the QVTo transformation presented in Section 5, our system modelers automatically generate the OPC UA information model from the SysML model. The information includes the actual_speed and actual_position, which are properties of the DataTypeParameterSetAxisType. They correspond to the data type of the UML PropertyParameterSetAxisType of the UML Stereotype AxisType in the UML profile CS-Robotics, as shown in Fig. 10. Next, the OPC UA information model serialized in a NodeSet file becomes the input of the script UaServerGenerator to generate the OPC UA server automatically. The OPC UA server runs on a Raspberry Pi¹³ and manages an address space containing OPC UA nodes corresponding to the available resources of Niryo Ned.

Users control Niryo Ned in the local network using Niryo Studio¹⁴, a tool to control Niryo's robots proposed by the Niryo company. The Niryo Ned communicates with the OPC UA server using the OPC UA PubSub communication mechanism, of which Niryo Ned is a publisher, and the server is a subscriber. All information regarding the robot arm, such as the actual position and actual speed data of the robot's joints, is up-to-date in the OPC UA server. Then, other devices can connect to the server via an Internet connection, collect updated data, and use them for any application.

Different stakeholders in the project use RvizUA to retrieve data from the OPC UA server; then to visualize and evaluate the synchronization between the physical and digital Niryo Neds. They give feedback so we can fix bugs related to the system in general or our M2M transformation specifically.

PAL Monitoring System. PAL is a manufacturing process in which "the bill-of-material parts and components are attached one by one to a unit sequentially by a series of workers to create a finished product" (Thomopoulos 2014). This use case includes three components: one robotic arm, one mobile robot, and one conveyor belt. The robotic arm is Niryo Ned. The mobile robot is TurtleBot3 Waffle Pi¹⁵. Fig. 21 illustrates the working scenario of this use case. The scenario is simple: Niryo Ned, at the first workstation, picks and places red covers on the conveyor

¹² RvizUA is a wrapper of Rviz, a 3D visualization tool for the Robot Operation System (ROS), with an extension feature that supports reading data from OPC UA nodes. The origin Rviz is available at <http://wiki.ros.org/rviz>.

¹³ <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>

¹⁴ <https://niryo.com/fr/robots-collaboratifs/niryo-studio-controler-les-robots/>

¹⁵ <https://robotis.us/turtlebot-3-waffle-pi/>

belt. It picks and places green covers on TurtleBot3. The two transporters TurtleBot3 and conveyor belt, deliver covers from the first workstation to the second one. At the second workstation, human workers assemble a cover with a box to produce a final product. In this use case, an OPC UA server must represent all resources of the PAL so users can access them through OPC UA clients. The goal of this use case is twofold. The first is to monitor all components of the PAL. Second, it is to count the number of objects transported by the mobile robot and conveyor belt.

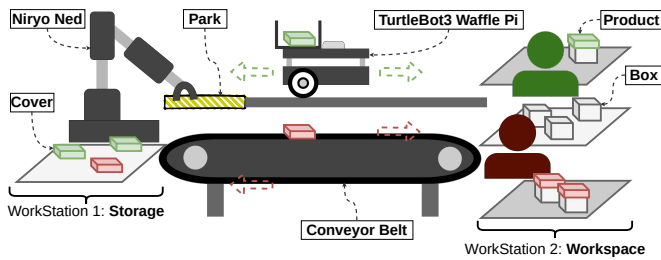


Figure 21 PAL for box manufacturing deployed in LocalSEA

The requirements of this use case are detailed in our previous publication (Nguyen, Dhoub, et al. 2022). To recap, the information models for Niryo Ned and TurtleBot3 must be the core of the PAL system’s OPC UA information model. The other details, such as the conveyor belt and the counters for red and green covers, can be considered additional components to the core. Thus, they can be added manually to the OPC UA information model as soon as the core is available. There is no difference in generating an information model for Niryo Ned as in the first use case. However, it requires some changes to generate the information model for TurtleBot3. Indeed, the current version of CS-Robotics only supports robotic arms: it proposes the concepts to model robots’ joints and their related components. A mobile robot like TurtleBot3 has no joints but wheels instead. To profit from our contributions, we need to modify and produce a SysML model that replaces Axes with Wheels, MotionDevices with MobileRobots, and MotionDeviceSystemType with MobileRobotSystemType. Then, the generated information model for Niryo Ned combined with the generated information model for TurtleBot3 is serialized in a Nodeset file. Using OPC UA modeling language, our modelers encode the necessary details into the file to form the final OPC UA information model for the PAL system. Next, system developers use the script UaServerGenerator to generate and run the OPC UA server.

The stakeholders in the project use UaExpert to observe the data updated in the OPC UA server and compare them with the running system. They give us feedback so we can evaluate the use case and fix bugs related to the system.

7.4. Maintenance and Issue Tracker

To facilitate research and further evaluation of our work, we have released our code on the Eclipse Foundation under the EPL open-source license. The transformation is implemented as a set

of Eclipse plugins constituting the Papyrus OPC UA Designer¹⁶. It can be installed using an update site¹⁷. Otherwise, users can use it through the Rich Client Platform (RCP) version available on the Papyrus for Manufacturing’s site¹⁸.

Users who find a bug in our contribution can contact us through the issue tracker Bugzilla¹⁹ also proposed by the Eclipse Foundation. It is a practical channel for us to receive feedback from the community.

8. Related work

Some MDE approaches for generating OPC UA address space and information modeling already exist. The most relevant work is mapping UML to OPC UA information model (Lee et al. 2017; Pauker et al. 2018). Lee et al. (Lee et al. 2017) present a metamodeling approach for transforming OPC UA address spaces into UML models. It is a bidirectional transformation between OPC UA information models and UML class diagrams. They also use the Eclipse Modeling Framework and the Query View Transformation (QVT) to implement their generator.

Rohjans et al. (Rohjans et al. 2010, 2013) also present an MDE approach for transforming IEC 61850 and CIM into OPC UA information models. They are the standard data models in the energy domain described in UML. They also developed a tool for automatic model transformation called CIMBAT. However, this approach is not generic. It addresses the energy domain and the UML model of the energy domain standards; thus, it is difficult to be adopted to other domains.

Compared to the approaches of Lee et al. and Rohjans et al., Pauker et al. (Pauker et al. 2016) present a general approach based on MDE. They propose a process model in addition to the transformation. Their transformation deals with the use case and the state-machine diagrams in addition to the class diagram. However, many transformations are done manually, and no sustainable linkage is given. In a second paper (Pauker et al. 2018), a transformation is described from class diagrams to OPC UA information model with additional OCL (Object Constraint Language)²⁰ constraints to ensure the construction of UML models that can be translated to OPC UA information models. In this work, the transformation is implemented based on Eclipse Modeling Framework (EMF)²¹ and the Atlas Transformation Language (ATL)²².

In (Friedl et al. 2020), the authors focus on the creation of the OPC UA information model and its description in a text document. They implement a graphical tool to model OPC UA information models based on the Eclipse Modeling Framework (EMF) and also use an MDE approach for the generation.

To the best of our knowledge, there is no existing work addressing the transformation of SysML models to OPC UA information models. The generation of information models

¹⁶ <https://wiki.eclipse.org/Papyrus/customizations/manufacturing/opcu>

¹⁷ <https://download.eclipse.org/modeling/mdt/papyrus/components/manufacturing/OpCUA/latest/>

¹⁸ <https://www.eclipse.org/papyrus/components/manufacturing/downloadopcu.html>

¹⁹ <https://bugs.eclipse.org/bugs/>

²⁰ <https://www.omg.org/spec/OCL>

²¹ <https://www.eclipse.org/modeling/emf/>

²² <https://www.eclipse.org/at/>

compatible with CSs has not been covered so far. Additionally, while there has been previous work done on transforming UML models to OPC UA information models, such as the work done by (Pauker et al. 2018), this work does not address the specific challenge of generating information models compatible with companion specifications.

Our work builds upon the plugin developed by (Pauker et al. 2018), which we have extended to enable the transformation of SysML models into OPC UA. This is significant because SysML offers a more structured and formal approach to system modeling than UML and is better suited for modeling Industry 4.0 systems (Lin et al. 2019; Holt & Perry 2008). Additionally, SysML is more system-centric and compact, which simplifies communication between SysML and OPC UA modelers. This enables a more precise and rigorous representation of the components, information flows, functions, interfaces, and relationships between the different elements of the system.

Another advantage of using SysML is that it provides a global view of the system and its interactions, which seem to be limited in OPC UA information models. By incorporating a SysML model, this gap can be filled, resulting in a more comprehensive representation of the system.

9. Summary and Future Directions

This paper presents an MDE approach to generating OPC UA information models from SysML models. The approach includes two steps. The first step is to enrich SysML with UML profiles for specific domains corresponding to CS's information models. In this paper, we specifically contributed a new UML profile for CS-Robotics. The second step is to convert the output SysML models of the first step into OPC UA information models. We shared our experiences of using QVTo transformation to do this job. The two use cases deployed in our testbed in CEA List prove the applicability and feasibility of our contributions. Moreover, the positive feedback from the stakeholders of both use cases allows us to use this approach to contribute to a demonstration at the showroom of List Tech Days 2023²³.

There are some further points to discuss. First, the paper suggests a UML profile for CS-Robotics, but the same approach can be applied to develop new UML profiles for other CSs as well. The library of CSs is officially available on the OPC Foundation's document site²⁴. The generation of the profiles from the NodeSets could be automatized since the Foundation provides the NodeSets and Other Supporting Files²⁵ which may facilitate the automation.

Second, the OPC UA modeling language proposes many specific datatypes that are non-existent in the SysML vocabulary. For example, for the integer number, the OPC UA modeling language proposes Int16, UInt16, Int32, UInt32, Int64, and UInt64, while SysML has only UML PrimitiveType Integer. It is logical since the OPC UA standard addresses low-level hardware and software components that always demand the optimization of resources. However, SysML inherits many

concepts from UML's vocabulary, which concerns high-level software-centric components. The high-level software focuses on user-friendly designs and hides the optimization in the background. Thus, to fully support the OPC UA information model, it is necessary to have a UML library covering all OPC UA DataTypes. This library is out of the scope of this current work but is in our future work plan.

Third, one challenge of this approach is that sometimes the generated OPC UA information model cannot fulfill the requirements of the expected system. One reason for this challenge is that the CSs library is still developing, and its current version cannot cover all industrial components. For example, the second use case presented in Section 7.3 requires modeling many components out of the scope of the UML profile for CS-Robotics, such as a mobile robot, a conveyor belt, and two counters. Our solution to overcome this obstacle relies on two points. First, modelers modify the UML profile if possible. Second, modelers can use the standard SysML language (BDD, IBD) to create system models that will contain all the details of the specific components of the model. Then the OPC UA information model will be generated from this SysML model.

After this research, we plan some near future works. First, the modified CS-Robotics for mobile robots presented in Section 7.3 are not mainstream. VDMA declares to update CS-Robotics with the concepts for other robot types in subsequent parts released in the future. Thus, we are waiting to improve our UML profile accordingly. Second, there are new use cases in the LocalSEA testbed that require the collaboration of multiple robotic arms. We plan to apply this research's contributions to deploy these use cases. We also plan to include more detailed evaluations of our approach with larger and more diverse sets of use cases, as well as incorporating specific research questions and metrics to evaluate the performance of our model generation approach. Finally, we continue to share our future results and updates with the community by maintaining the project under Eclipse Papyrus.

References

- Casse, O. (2017). Sysml: Object management group (omg) systems modeling language. *SysML in Action with Cameo Systems Modeler*, 1–63.
- Commission, I. E., et al. (2007). Power systems management and associated information exchange, data and communications security. *IEC International Standard IEC, 62351*.
- Friedl, S., von Arnim, C., Lechler, A., & Verl, A. (2020). Generation of opc ua companion specification with eclipse modeling framework. In *2020 16th ieee international conference on factory communication systems (wfcs)* (p. 1-7). IEEE. doi: 10.1109/WFCS47810.2020.9114448
- Gerpheide, C. M., Schifferers, R. R., & Serebrenik, A. (2016). Assessing and improving quality of qvto model transformations. *Software Quality Journal*, 24(3), 797–834.
- Goldschmidt, T., & Mahnke, W. (2012). Evaluating domain-specific languages for the development of opc ua based applications. *IFAC Proceedings Volumes*, 45(2), 860–865.


²³ <https://list.cea.fr/fr/event/list-tech-day-2023/>


²⁴ <https://opcfoundation.org/developer-tools/documents>


²⁵ <https://github.com/OPCFoundation/UA-NodeSet>

- Holt, J., & Perry, S. (2008). *Sysml for systems engineering* (Vol. 7). IET.
- Höppner, S., Haas, Y., Tichy, M., & Juhnke, K. (2022). Advantages and disadvantages of (dedicated) model transformation languages: A qualitative interview study. *Empirical Software Engineering*, 27(6), 159.
- Lee, B., Kim, D.-K., Yang, H., & Oh, S. (2017). Model transformation between opc ua and uml. *Computer Standards & Interfaces*, 50, 236–250.
- Lin, S.-W., Miller, B., Durand, J., Bleakley, G., Chigani, A., Martin, R., ... Crawford, M. (2019, June). *The Industrial Internet of Things Volume G1: Reference Architecture* (Tech. Rep.). Industrial Internet Consortium.
- Mahnke, W., Leitner, S.-H., & Damm, M. (2009). *Opc unified architecture*. Berlin Heidelberg: Springer.
- Mohagheghi, P., & Dehlen, V. (2008). Where is the proof?-a review of experiences from applying mde in industry. In *European conference on model driven architecture-foundations and applications* (pp. 432–443).
- Nguyen, Q.-D., Dhouib, S., Suri, K., & Rekik, F. (2022). From requirement specification to opc ua information model design: A product assembly line monitoring case study. In *2022 IEEE 20th international conference on industrial informatics (indin)* (p. 306-311). doi: 10.1109/INDIN51773.2022.9976157
- Nguyen, Q.-D., Rekik, F., Huang, Y., & Dhouib, S. (2022). Early lessons learned from the development of a local opc ua-based robotic testbed for research. In *2022 IEEE 31st international symposium on industrial electronics (isie)* (p. 615-618). doi: 10.1109/ISIE51582.2022.9831484
- OMG. (2011, January). *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1* (Tech. Rep.). Object Management Group.
- OPC Foundation. (2017). *OPC Unified Architecture - Part 1: Overview and Concepts* (Industry Standard Specification No. OPC 10000-1). OPC Foundation.
- OPC Foundation. (2021). *OPC Unified Architecture - Part 100: Devices* (Industry Standard Specification No. OPC 10000-100). OPC Foundation.
- Pauker, F., Frühwirth, T., Kittl, B., & Kastner, W. (2016). A systematic approach to opc ua information model design. *Procedia CIRP*, 57, 321-326. (Factories of the Future in the digital environment - Proceedings of the 49th CIRP Conference on Manufacturing Systems)
- Pauker, F., Wolny, S., Fallah, S. M., & Wimmer, M. (2018). Uml2opc-ua transforming uml class diagrams to opc ua information models. *Procedia CIRP*, 67, 128–133.
- Rohjans, S., Piech, K., & Lehnhoff, S. (2013). Uml-based modeling of opc ua address spaces for power systems. In *2013 IEEE international workshop on intelligent energy systems (iwies)* (pp. 209–214). IEEE.
- Rohjans, S., Usler, M., & Appelrath, H. J. (2010). Opc ua and cim: Semantics for the smart grid. In *IEEE PES T&D 2010* (pp. 1–8). IEEE.
- Santos, T. L. S., & Soares, M. S. (2023, March). A survey on what users think about SysML. *Systems Engineering*, sys.21663. doi: 10.1002/sys.21663
- Stahl, T., Völter, M., & Czarnecki, K. (2006). *Model-driven software development: technology, engineering, management*. John Wiley.
- Thomopoulos, N. T. (2014). *Assembly Line Planning and Control*. Cham: Springer International Publishing.
- VDMA. (2019). *OPC UA for Robotics - Part 1: Vertical Integration* (Industry Standard Specification No. VDMA 40010-1). Mechanical Engineering Industry Association.
- Wortmann, A., Barais, O., Combemale, B., & Wimmer, M. (2020). Modeling languages in industry 4.0: an extended systematic mapping study. *Software and Systems Modeling*, 19(1), 67–94.

About the authors

Dr. Fadwa REKIK  was formerly a postdoctoral researcher at CEA List, where she was engaged in pioneering projects related to Service Oriented Architecture (SOA), Industry 4.0, and the Internet of Things (IoT). Now, she is recently employed as a research engineer at Softeam. You can contact the author at fadwa.rekik@softeam.fr.

Dr. Saadia DHOUIB  is a senior researcher and a project manager at CEA List. She is working on model driven engineering methodologies and tools applied to the Robotics, Industry 4.0 and IoT domains. She is currently leading the open source Eclipse project Papyrus4Manufacturing. You can contact the author at saadia.dhouib@cea.fr.

Dr. Quang-Duy NGUYEN  is a young and motivated postdoctoral researcher at CEA List. Obtaining an engineer's degree in computer engineering, a master's degree in computer networking, and a Ph.D. in computer science, he has solid skills both in engineering and research. He is interested in building complex systems applied with novel technologies from the domains of IoT, OPC UA, Ontology, and AAS. You can contact the author at quang-duy.nguyen@cea.fr.