Enabling Component Reuse in Model-based System Engineering of Cyber-Physical Production Systems

Stefano Spellini, Sebastiano Gaiardelli, Michele Lora, Franco Fummi
Department of Computer Science
University of Verona
name.surname@univr.it

***Abstract*—Manufacturing lines are evolving into complex cyber-physical production systems. However, their growth in complexity is not matched by the development of structured modeling and design methodologies. In particular, approaches exploiting both models typical of the manufacturing domain and models used by computer engineers are still missing.**

In this work, we outline a design flow contemplating the reuse of already existing manufacturing lines' models, while designing novel advanced production systems. To enable such a flow, we propose a methodology extracting System Modeling Language (SysML) structural diagrams from AutomationML descriptions. Then, we propose to design the system functionalities on top of the produced diagrams.

The paper shows the application of the methodology to a concrete manufacturing line, the structure of which was originally modeled using AutomationML. To exemplify the advantages of the methodology, we exploit the models being generated to automatically extract a digital twin for the production system transportation line. The resulting digital twin is compliant with a well-known plant simulation tool.

I. Introduction

Today's manufacturing trends are constantly enriching traditional production systems with computational and communication infrastructures, transforming manufacturing lines into every day more complex systems. These recent trends are framed within the so-called "fourth industrial revolution" [1], *i.e.*, the transformation of production lines into *Cyber-Physical Production Systems (CPPSs)*. Indeed, such a transformation introduces unprecedented challenges in the design of manufacturing systems [2], as it requires the ability of designing systems while considering either the production processes, as well as the complex computational infrastructure monitoring and controlling the production processes.

The ongoing transformation of production lines into CPPSs is particularly problematic for Small and Medium Enterprises (SMEs). While a large manufacturing corporation may consider redesigning their production plants from scratch to incorporate novel technologies, SMEs are often forced to gradually introduce intelligence in their already existing lines. Furthermore, companies must be able to evaluate in advance the impact of re-designing their production lines. As such, it is necessary to develop modeling and design flows such as the one summarized in Figure 1, based on the Platform-Based Design (PBD) paradigm [3]. The methodology must support

This work is partially supported by the *European Commission through the project DeFacto (grant n. H2020-MSCA-IF-2019-894237)*, and by the *Veneto Regional project VIR2EM (grant. POR FESR Azione 1.1.4)*.

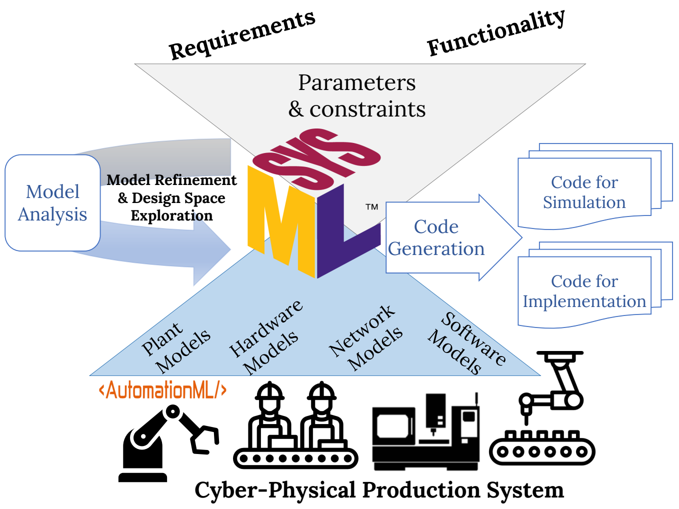


Figure 1: Conceptual view of the proposed design flow for CPPSs, and placement of the proposed model reuse methodology (cyan triangle) within the design flow.

both the *top-down modeling of requirements and functionalities*, as well as the *bottom-up reuse of components* already existing in the system and available to designers. Core to the methodology should be a language able to capture concepts belonging to both "cyber" and "physical" concepts of today's production systems. We chose to rely on the System Modeling Language (SysML) [4] for such a task, as it provides a variety of heterogeneous diagrams able to capture structure, behavior and requirements of systems. Once models encapsulate all this information, they enable system refinement and design space exploration through model analysis, as well as implementation and simulation exploiting code generation.

SysML is central in the described approach as it acts as a unified representation and modeling language for the entire system design process. Indeed, it is well suited for the top-down phase due to its focus on modeling and user-friendliness. However, it falls short when used to carry information about already existing components. For this reason, in the past, methodologies have been developed to convert and import already existing domain-specific models into SysML models. This task has been already carried out extensively for software [5] and hardware [6] components, and network infrastructure [7] enabling modeling of Cyber-Physical System (CPS). To enable the design of CPPSs, it is necessary allowing to import models of already existing machines.

For this reason, we propose a methodology to extract information from Automation Markup Language (AutomationML) [8] models of production lines to produce SysML structural diagrams to be used for the design of CPPSs. AutomationML has been introduced in 2006 as a data exchange language to store models useful at describing different aspects of manufacturing systems, becoming soon a standard de-facto for the modeling of production lines [9]. Then, the obtained SysML models are integrated through a top-down modeling phase which is meant to describe the aspects that AutomationML specifications failed to capture about the system. Finally, the produced SysML models are exploited to build a digital twin of the production system.

Section II presents the state of the art in CPPS modeling, and the background on AutomationML. Then, throughout the paper, the methodology is applied to the transportation subsystem of a full-size production line introduced in Section III, along with the guidelines of the intended modeling approach. Section IV details the methodology producing SysML models by reusing AutomationML specifications. Section V shows how the generated models may act as a library of components and as a platform for the top-down modeling phase of PBD. Then, in Section VI we exploit the produced models to generate the industrial plant's digital twin. Finally, we draw some conclusions in Section VII.

II. Preliminaries

AutomationML is an XML-based data format to exchange information describing manufacturing system [10]. Different standards are intertwined within AutomationML to describe multiple aspects of production plants: from topology to the logic controlling machines' microcontrollers.

The Computer Aided Engineering Exchange (CAEX) (IEC 62424) standard provides to AutomationML the features required to represent a topological view of the system, *i.e.*, relations between objects, such as types of machinery and materials. It is object-oriented [11] as it provides system objects' semantics using roles which are defined by *role class* library. Role classes express the abstract functionality representation of objects, without specifying their implementations. As an example, a "resource" is a role for an object, and can be further detailed to represent a piece of equipment or material. Concrete resource instances are typically specified by *system unit classes* usually containing vendor-specific AutomationML objects. Relations between objects are specified within the *interface classes* library. AutomationML descriptions hierarchically organized within a AutomationML description. In particular, AutomationML's core is the *instance hierarchy* storing the hierarchy of components and sub-components composing the system.

Concerning the system behavior, the COLLAborative Design Activity (COLLADA) interchange standard, incorporated in AutomationML, allows specifying information about three-dimensional shapes and the kinematics of the system. Thus, it is focused on the mechanical behavior of the system. PLCopen is the standard embedded within AutomationML to describe the logic behavior of the system. It allows specifying simple behaviors expressed by impulse diagrams, sequence function charts, logic networks, state charts, Gantt and Pert charts.

A. State of the art in production system modeling

In the context of manufacturing and CPPSs, Model-based System Engineering (MBSE) is a quite popular approach to support the design and the development lifecycle. Authors in [12] propose a specialization of SysML to support the development of automation software (*i.e.*, Programmable Logic Controller (PLC) software) for manufacturing systems. Then, they suggest a methodology to automatically generate control software from SysML models of the production plant. [13] proposes to use an AutomationML-based model to create "Plug-and-Produce" facility components, which integrate information about their production capabilities. In this regard, the production resources composing the manufacturing system are modeled in terms of their ability to produce a particular product or to implement a required production "Skill". As such, AutomationML includes the necessary constructs to model such a system viewpoint. The work proposed in [14] presents an extension of Unified Modeling Language (UML) to support the modeling of mechatronic components and Internet-of-Things (IoT) production environments. Therefore, manufacturing system components are adjoined with IoT interfaces at a modeling stage, facilitating their latter integration.

A systematic review [9] of modeling languages for manufacturing highlighted a gap between the involved research communities. In particular, system engineering and knowledge-representation languages are widely used in this field, however rarely combined. The survey also found out that AutomationML and the UML (which is the root of SysML) are quite popular in the field. However, they are rarely used together and the reuse of existing models is hardly tackled.

To the best of our knowledge, the only approach combining AutomationML models and SysML has been proposed in [15]. The authors investigate the commonalities and differences between the structural modeling provided by the two languages. They also propose a SysML profile and an AutomationML metamodel, to boost languages' interoperability. While their objective of reusing models is common to this work, our approach exploits AutomationML models reuse in a PBD framework. As such, it considers also other modeling and system's aspects, such as behavioral and parametric facets, implemented in corresponding SysML diagrams. Furthermore, the reuse of AutomationML models is implemented using a direct one-to-one mapping between AutomationML and SysML standard elements. As such, it avoids using a SysML custom profile and additional stereotypes.

III. Motivations and Overview

When dealing with today's production systems, the bare manufacturing plant is not the only aspect to consider. For this reason, concerning modeling, the most obvious limitation of AutomationML is the lack of expressiveness to model advanced functionalities making plants smart. Neither the computational infrastructure nor the information flowing through the system's machinery can be represented by AutomationML

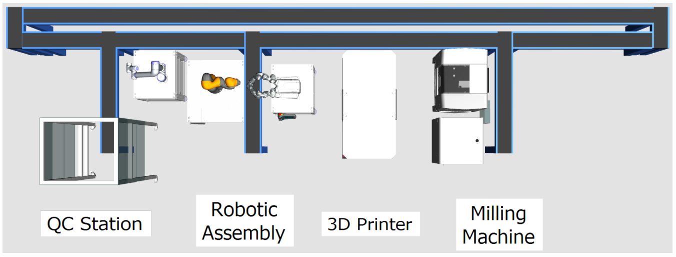


Figure 2: Structure of the advanced manufacturing production line used as a case study in this work. The machines are connected through an articulated software-controlled transportation mechanism.

models. Indeed, PLCopen allows modeling control logic. However, its constructs are not suitable to model anything more complex than control software running on PLCs.

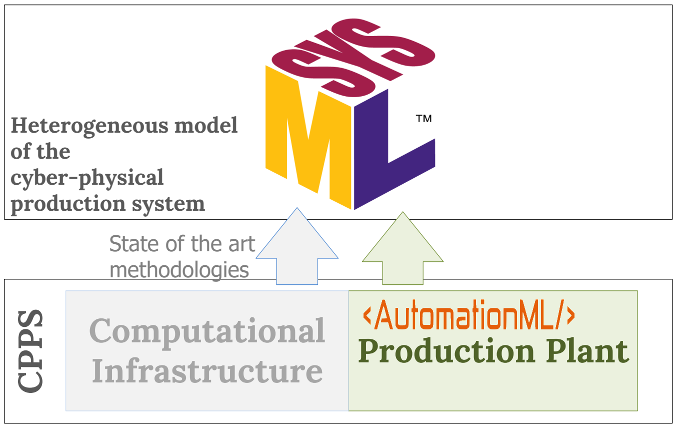
AutomationML expressiveness is limited also when specifying manufacturing functionalities. While it supports structural modeling through CAEX and kinematic modeling through COLLADA, it does not provide constructs useful to specify actions and primitives provided by the single machines composing the systems. The lack of primitives to express machinery's behavior in terms of actions makes it difficult to specify production processes recipes. While it is possible specifying which products, resources and processes are related, AutomationML does not allow specifying how a process is structured to transform a resource into a product.

For all these reasons, design flows for CPPS require more expressive languages than AutomationML. However, manufacturing systems engineers are already confident with AutomationML. Furthermore, many AutomationML descriptions are already available for existing manufacturing systems, and the standard is popular also among researchers [9], [16]. Thus, while searching for novel design and modeling approaches, we advocate the importance of letting system designers and engineers continue using their languages of choice. Such a feature has already been investigated when dealing with CPSs [17]. However, the reconciliation to a single language of the production viewpoint of CPPSs has never been proposed. For this reason, we aim at enabling the integration of existing AutomationML models within SysML-based design flows. AutomationML descriptions of existing manufacturing systems are analyzed and used to generate SysML models. Then, generated models can be used by designers to carry on the top-down phase of designing a CPPS. In this way, the designers can model novel functionalities and refine them onto the model of the already existing architecture. The proposed methodology enable better modeling features for designer to tackle advanced manufacturing systems design, while preserving already existing models of systems.

The paper proposes a methodology implementing the bottom-up phase and shows the application of the generated models. The presentation of the methodology is paired with its application to the design of a complex *conveyors system*.

A. Case study: conveyors system

The case study is based on the materials and products transportation subsystem of the production line available at our


Heterogeneous model of the cyber-physical production system
SysML™
State of the art methodologies
CPPS
Computational Infrastructure
<AutomationML/>
Production Plant
Figure 3: Bottom-up reuse of existing models. While the computational components of the system are reused by applying state-of-the-art methodologies, the production plant model, originally expressed as a AutomationML description, is incorporated into the SysML model.

research facility[1]. The production line structure is depicted in Figure 2. It is composed by multiple machines connected by multiple conveyor belts, controlled by computational devices connected through a communication network.

The transportation system is made by a closed-loop main conveyor belt. Multiple conveyor bays are linked to the main belt in order to move the materials from the transportation system to the machines and back. The passage of material between the main belt and each bay is managed by a switching mechanism that is guided by sensors detecting and identifying the minipallets moving around the production system.

IV. Reusing manufacturing systems models

AutomationML models, built upon the CAEX standard, depict the manufacturing system's structure: the components composing the production line and their relations. As such, they can be exploited for the bottom-up phase of the proposed PBD approach, to construct the structural part of SysML models. As depicted in Figure 3, the methodology hereby described creates the production plant model alongside the computational infrastructure model, within the same SysML description. This is a fundamental feature, enabling models' reuse and boosting the design process for complex systems. To accomplish such a task, we propose a mapping between the two languages. Table I reports the mapping: the AutomationML elements are categorized in libraries, instances, classes, objects and relations. The table reports, for each AutomationML element, the corresponding SysML elements.

The main component in an AutomationML model is the *class*: it is typically organized in libraries, depending on the concept or component it represents. In this regard, AutomationML defines three different libraries: the *SystemUnitClassLib*, the *RoleClassLib* and the *InterfaceClassLib*. The *SystemUnitClassLib* represents system components and their relationships. The *RoleClassLib* encapsulates semantic classes that are associated to objects, while the *InterfaceClassLib* allows defining the type of communication between elements.

[1]https://www.icelab.di.univr.it/

Table I: Mapping of AutomationML elements to SysML objects.

	AML Element	SysML Element
Libraries	SystemUnitClassLib	Block Definition Diargam
	RoleClassLib	
	InterfaceClassLib	
Instances	InstanceHierarchy	Internal Block Diagram
Classes	SystemUnitClass	Block
	RoleClass	
	InternalElement	
	InterfaceClass	InterfaceBlock
Objects	Attribute	Property
	Internal Link	Connector
	ExternalInterface	Port
Relations	BaseClass $\rightarrow$ Class	Generalization
	RoleRequirements $\rightarrow$ Class	Realization
	SupportedRoleClass $\rightarrow$ Class	Realization

A AutomationML library is directly mapped to a SysML Block Definition Diagram (BDD), characterizing the hierarchy of system components (*i.e.*, blocks) and their relations. AutomationML classes frequently enclose attributes to further specify components' properties or parameters. As such, an AutomationML attribute is mapped to a SysML *block property*, typed accordingly to the attribute type. Assigning the correct type to an attribute is not trivial: AutomationML provides a small set of native types (*i.e.*, *integer*, *real*), but in case a physical dimension (*i.e.*, velocity or temperature), a workaround is needed to specify their units of measure.

AutomationML implements different relationships between classes: a AutomationML *system unit class* might be either a base or a specialized class inheriting attributes from a superclass. The inheritance relation is mapped and represented in SysML through the *Generalization* connector. In addition, at least one role class must be associated with a class or to an *InternalElement*, which is a class instance. As such, the *RoleRequirement* construct represents the requirement that a specific role must be associated with the element. Furthermore, an element associated with multiple roles includes multiple *SupportedRoleClass* AutomationML relationships. In this context, SysML handles roles as abstract classes. Therefore, the association of a role class to a class or an internal element is mapped to the *realization relation* between SysML blocks.

The AutomationML *ExternalInterface* defines an interface to an external object or class. An interface is typed by an *InterfaceClass*, which encapsulates user-defined attributes to characterize the communication. The InterfaceClass is implemented as an *InterfaceBlock* in SysML, which belongs to the BDD defining the model's interfaces. The *InterfaceBlock* is used to type a *Port* object directly acquired from ExternalInterfaces composing the class or the specific InternalElement.

The *InstanceHierarchy* is a central section of AutomationML: it structurally defines class instances and object connections between ExternalInterfaces. We correlate this concept to the Internal Block Diagram (IBD) of SysML, where instances of classes are related to each other through *Ports*. As such, AutomationML ExternalInterfaces typed by InterfaceClasses and connected using *InternalLinks*, are mapped to SysML Ports, typed by InterfaceBlocks and connected with *Connectors*. An important limitation of AutomationML InternalLinks is the lack of expressivity: they do not allow specifying the kind of information passing through the ports being connected. As such, they only represent the connection between elements, without providing any further information.

A. Basic Components

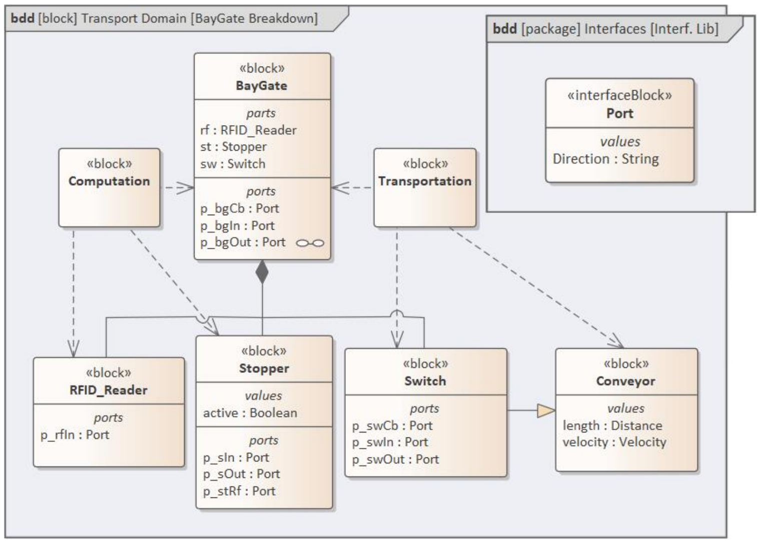
Figure 4 illustrates the components of a portion of our conveyor system: a conveyor gate, which stops the minipallet, reads its id and deviates it in the conveyor bay whether necessary. Otherwise, the pallet is unlocked and can continue its journey through the main conveyor.

The gate is composed of a minipallet stopper, locking the minipallet in a specific position of the main conveyor. Then, an RFID reader reads a tag fixed on the minipallet frame and, once the supervisor produces a response, the stopper deactivates. Then, a special conveyor, moving along two axes, configures itself to let the unit in the conveyor bay or to let the minipallet flow throughout the gate. As depicted in Figure 4b, the SystemUnitClassLib *BayGate Breakdown* consists of a SystemUnit class for each component. It includes a set of base classes of simple components: the *Stopper* and the *RFID_Reader*. The library also defines a subclass: the *Switch* class, which is a specialization of the *Conveyor* base class, inheriting the *length* and the *velocity* attributes. Exploiting such a relationship between classes allows the extension of the subclass with additional class members, such as external interfaces. In particular, the *Switch* class extends the *Conveyor* class by defining three ExternalInterface objects. The ExternalInterface type is defined by the *Port* InterfaceClass specified in an InterfaceClassLib. Finally, the *BayGate*, which is the primary class in the hierarchy, includes port-typed external interfaces and instances of components as class members. Each class has one or multiple associated roles, defined in a RoleClassLib. In this scenario, three role classes are sufficient to represent the semantic category of each object composing the system: *Computation*, *Manipulation* and *Transportation*.

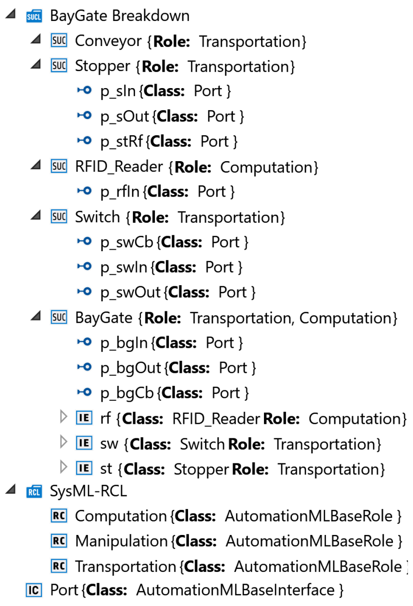
Each library defined in AutomationML is translated to a SysML BDD. For the sake of clarity, in Figure 4a only the "BayGate Breakdown" and "Interface Lib." diagrams are depicted, omitting the "SysML-RCL" BDD. The entire set of classes defined in AutomationML is mapped to blocks related to each other according to inheritance and composition rules. The block "Conveyor" is, in fact, in a Generalization relationship with the "Switch" class, that is, in turn, associated to the "BayGate" block in a membership relationship. Therefore, the "Switch" class becomes a property "part" of the "BayGate". Class attributes are transformed to block properties (which become "Values" properties once typed) and ExternalInterfaces become "Ports", typed by an interfaceBlock. Role classes are related to other classes using the "Realization" connector. As an example, the "BayGate" class is associated to both the "Computation" and the "Transportation" roles.

B. System Structure

The AutomationML *InstanceHierarchy* is a hierarchical structure of objects called *internalElements*, which are instances of classes specified in the AutomationML libraries.

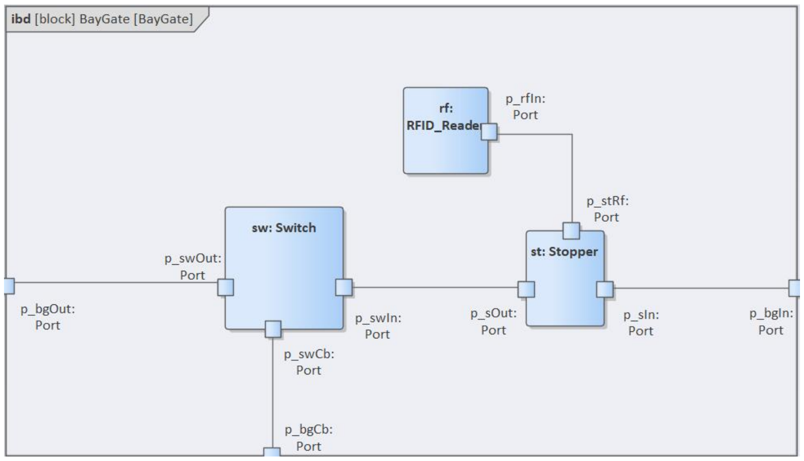


(a)

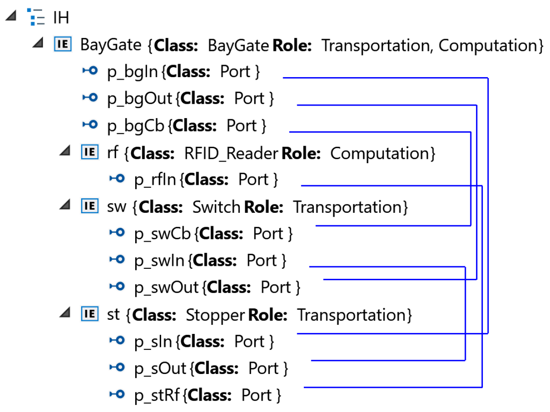

- BayGate Breakdown
  - Conveyor {**Role:** Transportation}
  - Stopper {**Role:** Transportation}
    - p_sIn {**Class:** Port }
    - p_sOut {**Class:** Port }
    - p_stRf {**Class:** Port }
  - RFID_Reader {**Role:** Computation}
    - p_rfIn {**Class:** Port }
  - Switch {**Role:** Transportation}
    - p_swCb {**Class:** Port }
    - p_swIn {**Class:** Port }
    - p_swOut {**Class:** Port }
  - BayGate {**Role:** Transportation, Computation}
    - p_bgIn {**Class:** Port }
    - p_bgOut {**Class:** Port }
    - p_bgCb {**Class:** Port }
    - rf {**Class:** RFID_Reader **Role:** Computation}
    - sw {**Class:** Switch **Role:** Transportation}
    - st {**Class:** Stopper **Role:** Transportation}
- SysML-RCL
  - Computation {**Class:** AutomationMLBaseRole }
  - Manipulation {**Class:** AutomationMLBaseRole }
  - Transportation {**Class:** AutomationMLBaseRole }
- Port {**Class:** AutomationMLBaseInterface }

(b)

Figure 4: Visual representation of the mapped objects between AutomationML and SysML, to define the system components. (b) describes AutomationML libraries of SystemUnit classes, roles and interfaces, while (a) depicts the BDD derived from such object classes.



(a)


- IH
  - BayGate {**Class:** BayGate **Role:** Transportation, Computation}
    - p_bgIn {**Class:** Port }
    - p_bgOut {**Class:** Port }
    - p_bgCb {**Class:** Port }
    - rf {**Class:** RFID_Reader **Role:** Computation}
      - p_rfIn {**Class:** Port }
    - sw {**Class:** Switch **Role:** Transportation}
      - p_swCb {**Class:** Port }
      - p_swIn {**Class:** Port }
      - p_swOut {**Class:** Port }
    - st {**Class:** Stopper **Role:** Transportation}
      - p_sIn {**Class:** Port }
      - p_sOut {**Class:** Port }
      - p_stRf {**Class:** Port }

(b)

Figure 5: The system and components structure represented in an IBD (b), derived from the AutomationML InstanceHierarchy (a).

Each object's semantics is defined by associated roles. Furthermore, the object's interfaces to other objects are concretely used to represent object to object connections, other than simply declared. Regarding the conveyor gate example, the structure of the "BayGate", depicted in Figure 5b, is consistent with the class defined in the SystemUnitClassLib. However, it is also further characterized by defining the connections between interfaces of its sub-components. In this regard, *InternalLinks* are used to connect ports of the gate components set. The instantiation of a class in the hierarchy also includes assigning a value to its set of attributes.

Parts, *ports* and *values* are specialization of the type *Property* in SysML. For this matter, the purpose of the IBD is to define the internal structure of a Block by means of properties and relationships between properties. The IBD depicted in Figure 5a is built upon the BayGate internalElement. Since the internalElement is an instance of a class, it carries the same structure of the related block defined in the BDD. Consequently, class members such as properties, parts and ports are instantiated and are associated with the corresponding block. In the IBD, InternalLinks between externalInterfaces become connectors between ports.

V. Top-down modeling

The SysML models generated from AutomationML descriptions are barebones: they delineate the basic structure of objects with no functionality attached. Furthermore, the communication between objects is not exhaustively defined. As discussed, the reason for such a shortcoming is related to the lack of expressivity of the AutomationML language. On the other hand, SysML does provide multiple constructs to enrich the diagrams obtained from the bottom-up phase. As

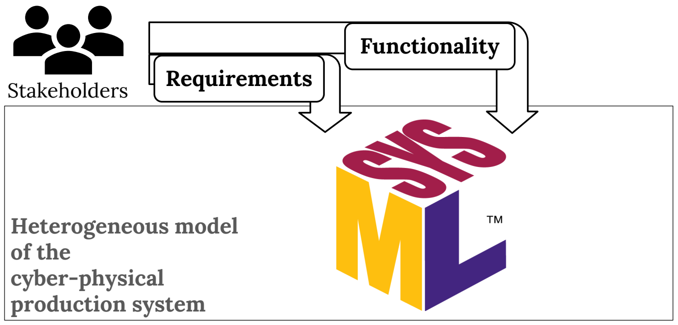


Figure 6: Top-down phase: the proposed methodology allows building the "platform" on top of which is possible specifying the functionalities and the requirements of the system being designed.

such, as summarized by Figure 6, the barebones models acts as a *platform* to be refined by the designer, by expressing a larger set of information regarding the system to specify intended functionalities and requirements. Furthermore, SysML provides other diagrams useful to model many more details of different systems' aspects and viewpoints. For example, behavior diagrams such as the *activity diagram* focus deeply on the system dynamic, specifying sequences of events. In this Section, we show the applicability of the models generated in the previous section to support a complete CPPSs design flow.

A. Components Communication Modeling

Figure 7b reports the manually refined IBD of Figure 5a: *item flows* have been added above simple connectors. This step enables specifying the type of information or object flowing between ports. Thus, it allows defining a piece of functionality performed by the system architecture. In fact, by determining the kind of objects passing through a connection, the connection itself is refined from an abstract concept to a concrete object. As an example, a communication network is very different from a physical object that connects two points in the physical space. However, AutomationML does not have the required expressivity to detail such a set of information that is crucial in today's production systems.

The diagrams in Figure 5a model exclusively the architecture of the gate component, without specifying object flows between one sub-component and another. The Figure 7b depicts a refined version of the original IBD, defining the type of communication other than outlining the direction of the flow of objects. Applying this modeling approach, it is possible to discriminate between physical objects and information. In the example, the *MiniPallet* block represents the object moving through the physical components of the conveyor system. Meanwhile, the *RFID_Tag* is the unique identifier of a MiniPallet object. The main difference between these two objects is that they belong to different domains: the minipallet is a physical object, while the RFID tag carries a piece of information that is purely digital. As such, the channel on which these entities move is substantially different. The ability to define components has been enabled by moving the modeling of the system from AutomationML to SysML.

The *Stopper* modeled in Figure 7b acts as the contact point between the plant and the computational platform, carrying out two types of communication. The inward object flow in its input port is typed by a MiniPallet block, which represents a physical object. Then, it has two output ports: the *p_stRf* and the *p_sOut* ports. While both output ports share the same type, they accept different flows of objects: the first accepts a digital RFID_Tag object (highlighted in red in Figure), while the second accepts the flow of a physical MiniPallet object.

B. Components Behavior Modeling

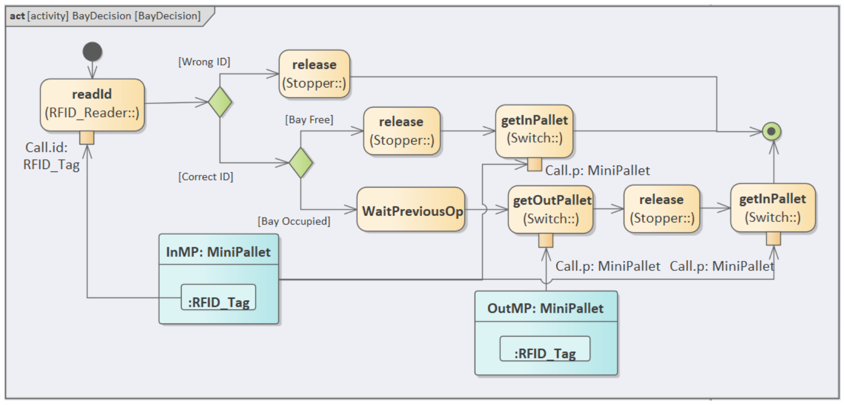
Behaviors can be determined in SysML by associating an *operation* to a block of a BDD. Each operation is conceptually similar to a function and is represented as a class method. Therefore, an operation is defined by a set of input parameters and a return value and the model of its behavior. A parameter can be typed by a native type or by another block. The activity diagram depicted in Figure 7a represents the sequence of actions following the arrival of a minipallet at the gate and the decision process that establishes whether it has to be pulled inside the attached bay or it has to be released. The activity diagram separates the types of flows, outlined by arrows between entities, depending on the kind of information they carry: if the flow represents the transfer of control between actions, it is called "control flow", while if the flow serves as the movement of objects, it is called "object flow". Each action in the diagram is a call to an operation defined in the corresponding block of the BDD. As an example, the *readId* action, which is the first action of the activity, takes as an input parameter an *RFID_Tag* object. As such, the flow between the *inMP* object (*i.e.*, an instance of the MiniPallet block definition) and the *readId* action is an object type.

The minipallet destination is decided on the ID of the RFID tag and is implemented through a couple of decision nodes, connected with control flows: if the ID is the wrong one, the pallet is released on the main belt, otherwise, the control passes to another decision node. This entity is in charge of determining whether the bay is free. In such a case, the stopper is released and the pallet is pulled in the bay by the *getInPallet* function of the *Switch* component. In the other scenario, the ingoing minipallet waits for the completion of the previous production operation and the clearance of the bay from the *OutMP* minipallet. Then the ingoing minipallet is pulled in.

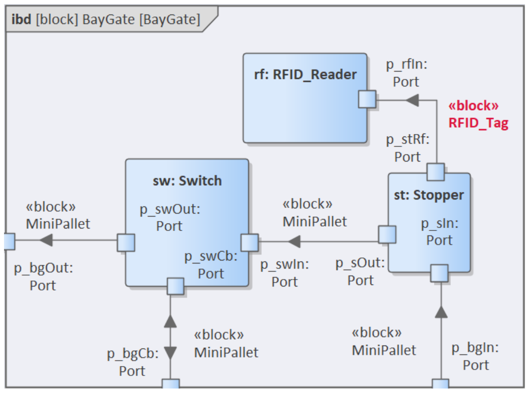
While this diagram portrays a simple scenario for demonstration purposes, the complexity of such a kind of modeling constructs enables the specification of intricate behaviors. The expressiveness guaranteed by SysML allows to overcome the limitations of the AutomationML structure and, thus, empowers the modeling of modern intelligent production systems.

VI. Exploiting models for CPPS design

The underlying heterogeneity of SysML is well-suited to represent and model CPPSs. Modeling tools based on SysML, other than providing a clear and intuitive environment, are easily extensible, thus enabling customized generation of code for any simulator of choice. In this Section, we report our experience in generating the code necessary to simulate the case study within *Tecnomatix Plant Simulation*, a well-known production line simulation tool distributed by Siemens.



(a)



(b)

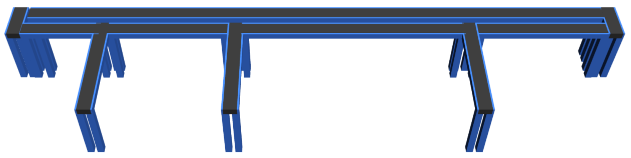
Figure 7: Refined SysML models defining functionalities and overcoming AutomationML expressiveness limitations. (a) *activity diagram* in SysML characterizing components' behaviors. (b) The refined IBD with additional information about object flows and directions.

The automatic instantiation of a simulation within Plant Simulation is a two-phased task:

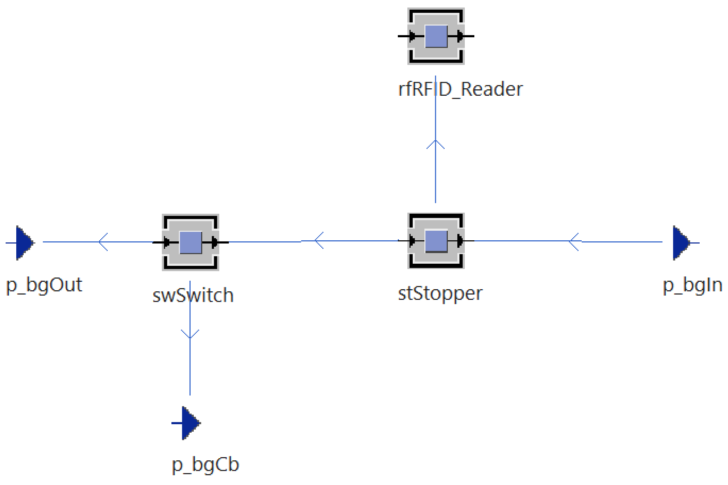
- *Structural information extraction*: the XMI storing the SysML model is parsed to capture the plant topology, *i.e.*, the line's components, their positions in space and interactions. The acquired information is used to produce a SimTalk script instantiating the components inside the simulation environment and connected with each other.
- *Behavioral information extraction*: code generators for the most popular programming languages (*i.e.*, C/C++, Java and Python) are provided by all the UML/SysML editors available today. We exploit the automatic C code generation to translate the behavioral diagrams into a C implementation. The implementation is then compiled into a dynamic linked library and imported within Plant Simulation using an Application Programming Interface provided by the simulator.

Figure 8 shows the Plant Simulation model being generated using the models produced by applying the proposed methodology. Figure 8a shows the 3D model of the transportation system being simulated within Plant Simulator. Meanwhile, Figure 8b depicts the internal model of a single conveyor gate of the transportation system. The latter is automatically generated by applying the proposed methodology to the SysML IBD in Figure 7b, which has been automatically generated by extracting the information contained in the AutomationML description reported by Figure 5b. The protocol used by the objects depicted in Figure 8b is automatically synthesized from the activity diagram in Figure 7a. Thus, the simulation of each conveyor gate evolves accordingly to the model specified during the top-down phase of the presented approach.

To understand the size of the problem, Table II characterizes the modeling dimensions of the case study. It reports the characteristics of the original *AutomationML* description in terms of lines of code, defined classes and internal links, and the characteristics of the *SysML* extracted from the *AutomationML* (*i.e.*, bottom-up). It also reports the number of BDD, IBD and lines of XMI code in the generated model, the number of activity diagrams and state machine diagrams manually



(a)



(b)

Figure 8: The simulation models in the Plant Simulation environment, derived from SysML. (a) The 3D model of the transportation system. (b) The internal structure of a conveyor gate.

specified by the designer in SysML (*i.e.*, top-down). Finally, it details the number of lines of C++ code automatically generated from the model.

Models generated through the proposed methodology may be used for many other purposes within the design flow of industrial production systems that we summarizes hereby.

a) Verification and Validation: to evaluate the correctness of requirements and behaviors by models is crucial for mission-critical applications, such as industrial systems [18]. So far, methods for the verification and validation of SysML models rely either on formal methods or simulation [19].

Table II: Characteristics of the system original descriptions, and of the models generated by applying the proposed approach.

AutomationML	Lines of Code	13372
	Libraries	8
	Classes	33
	Internal Links	47
SysML (bottom-up)	Block Definition Diagrams	8
	Internal Block Diagrams	7
	XMI Lines of Code	6326
SysML (top-down)	Activity diagrams	3
	State Machine diagrams	1
	XMI Lines of Code	5115
Generated C++	Lines of Code	167

b) System Analysis and Optimization: accurate system models may be used to perform in-depth analysis and optimizations. Design-space exploration is intrinsic concept of any PBD flow, that may be performed on top of sufficiently expressive SysML models [20]. Optimization problems and formal models can be built on top of SysML models [21], [22] and then resolved exploiting existing solvers.

c) Code Generation and Implementation: SysML expressiveness is not limited to software: it is also efficient at capturing features of hardware components and their interactions with the software components. Thus, it may become a fundamental tool to support hardware-software integration. For instance, SysML models may come in handy while integrating a Manufacturing Execution System into a production line [23].

SysML also allows the generation of control software starting from diagrams composing a system model. In particular, SysML diagrams can be used to generate the templates for PLC software consistent with the IEC 61131-3 standard [24] to be later deployed on the system.

VII. Conclusions

This paper identified some limitations of the current state of the practice on modeling for manufacturing systems. In particular, the paper focused on the limitations of a prominent language for the description of production systems (*i.e.*, AutomationML). Then, while we recognize the importance of allowing designers to keep using the languages of their choice, we advocate the necessity of introducing more sophisticated modeling languages and structured methodology for the design of advanced production systems based on CPPSs.

Thus, we proposed the use of PBD as the methodological framework, and SysML as specification language. Then, we presented a methodology to automatically produce SysML models from AutomationML descriptions. We presented the mapping, and we applied it to a real-world industrial transportation system to generate its SysML model from its AutomationML description. Then, we extended the model to specify its functionalities. Finally, we applied automatic code generation to produce simulation models for a widely used industrial processes simulator, and presented other possible uses of the models produced by the presented methodologies.

References

[1] M. Hermann *et al.*, "Design principles for industrie 4.0 scenarios," in *Proc. of HICSS*, 2016, pp. 3928–3937.

[2] L. Ribeiro, "Cyber-Physical Production Systems' Design Challenges," in *Proc. of IEEE ISIE*, 2017, pp. 1189–1194.

[3] K. Keutzer, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: orthogonalization of concerns and platform-based design," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 19, no. 12, pp. 1523–1543, 2000.

[4] S. Friedenthal *et al.*, *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[5] E. Korshunova, M. Petkovic, M. Van Den Brand, and M. R. Mousavi, "CPP2XMI: Reverse engineering of UML class, sequence, and activity diagrams from C++ source code," in *2006 13th Working Conference on Reverse Engineering*. IEEE, 2006, pp. 297–298.

[6] N. Bombieri, E. Ebeid, F. Fummi, and M. Lora, "On the reuse of heterogeneous IPs into SysML models for integration validation," *Journal of Electronic Testing*, vol. 29, no. 5, pp. 647–667, 2013.

[7] E. Ebeid, F. Fummi, and D. Quaglia, "Model-driven design of network aspects of distributed embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 4, pp. 603–614, 2015.

[8] AutomationML Consortium, "AutomationML," 2006. [Online]. Available: https://www.automationml.org/o.red.c/home.html

[9] A. Wortmann, O. Barais, B. Combemale, and M. Wimmer, "Modeling languages in Industry 4.0: an extended systematic mapping study," *Software and Systems Modeling*, vol. 19, no. 1, pp. 67–94, 2020.

[10] R. Drath, "Let's talk AutomationML what is the effort of AutomationML programming?" in *Proc. of IEEE ETFA*, Sep. 2012, pp. 1–8.

[11] N. Schmidt and A. Lüder, "AutomationML in a Nutshell," *AutomationML eV*, 2015.

[12] B. Vogel-Heuser, D. Schütz, T. Frank, and C. Legat, "Model-driven engineering of manufacturing automation software projects – a sysml-based approach," *Mechatronics*, vol. 24, no. 7, pp. 883 – 897, 2014, 1. Model-Based Mechatronic System Design 2. Model Based Engineering.

[13] P. Danny, P. Ferreira, N. Lohse, and M. Guedes, "An automationml model for plug-and-produce assembly systems," in *Proc. of IEEE INDIN*, 2017, pp. 849–854.

[14] K. Thramboulidis and F. Christoulakis, "UML4IoT—A UML-based approach to exploit IoT in cyber-physical manufacturing systems," *Computers in Industry*, vol. 82, pp. 259 – 272, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016636151630094X

[15] L. Berardinelli, S. Biffl, A. Lüder, E. Mätzler, T. Mayerhofer, M. Wimmer, and S. Wolny, "Cross-disciplinary engineering with AutomationML and SysML," *at - Automatisierungstechnik*, vol. 64, pp. 253 – 269, 2016.

[16] S. Spellini, R. Chirico, M. Panato, M. Lora, and F. Fummi, "Virtual Prototyping a Production Line using Assume-Guarantee Contracts," *IEEE Transactions on Industrial Informatics*, 2020.

[17] M. Lora, S. Vinco, and F. Fummi, "Translation, Abstraction and Integration for Effective Smart System Design," *IEEE Transactions on Computers*, vol. 68, no. 10, pp. 1525–1538, 2019.

[18] R. Baduel, M. Chami, J.-M. Bruel, and I. Ober, "SysML Models Verification and Validation in an Industrial Context: Challenges and Experimentation," in *European Conference on Modelling Foundations and Applications*. Springer, 2018, pp. 132–146.

[19] M. Debbabi, F. Hassaine, Y. Jarraya, A. Soeanu, and L. Alawneh, *Verification and validation in systems engineering: assessing UML/SysML design models*. Springer Science & Business Media, 2010.

[20] M. Mura, L. G. Murillo, and M. Prevostini, "Model-based design space exploration for RTES with SysML and MARTE," in *Proc. of IEEE/ECSI FDL*, 2008, pp. 203–208.

[21] P. Leserf, P. de Saqui-Sannes, and J. Hugues, "Multi domain optimization with SysML modeling," in *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, 2015, pp. 1–8.

[22] S. Kanthabhabhajeya and B. Lennartson, "Modeling and optimization of synchronous behavior for packaging machines," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 1684 – 1691, 2014.

[23] L. Piétrac, A. Lelevé, and S. Henry, "On the use of sysml for manufacturing execution system design," in *ETFA2011*, 2011, pp. 1–8.

[24] M. Jamro and B. Trybus, "An approach to SysML modeling of IEC 61131-3 control software," in *2013 18th International Conference on Methods Models in Automation Robotics (MMAR)*, 2013, pp. 217–222.