10

Open Semantic Meta-model as a Cornerstone for the Design and Simulation of CPS-based Factories

Jan Wehrstedt[1], Diego Rovere[2], Paolo Pedrazzoli[3], Giovanni dal Maso[2], Torben Meyer[4], Veronika Brandstetter[1], Michele Ciavotta[5], Marco Macchi[6] and Elisa Negri[6]

[1]SIEMENS, Germany
[2]TTS srl, Italy
[3]Scuola Universitaria Professionale della Svizzera Italiana (SUPSI), The Institute of Systems and Technologies for Sustainable Production (ISTEPS), Galleria 2, Via Cantonale 2C, CH-6928 Manno, Switzerland
[4]VOLKSWAGEN, Germany
[5]Università degli Studi di Milano-Bicocca, Italy
[6]Politecnico di Milano, Milan, Italy
E-mail: janchristoph.wehrstedt@siemens.com; rovere@ttsnetwork.com; pedrazzoli@ttsnetwork.com; dalmaso@ttsnetwork.com; torben.meyer@volkswagen.de; veronika.brandstetter@siemens.com; michele.ciavotta@unimib.it; marco.macchi@polimi.it; elisa.negri@polimi.it

A key enabler towards the fourth industrial revolution is the ability to maintain the digital information all along the factory life cycle, despite changes in purpose and tools, allowing data to be enriched and used as needed for that specific phase (digital continuity). Indeed, a fundamental issue is the lack of common modelling languages, and rigorous semantics for describing interactions – physical and computational – across heterogeneous tools and systems, towards effective simulation. This chapter describes the definition of a semantic meta-model meant to describe the functional characteristics of a CPS, which are relevant from its design and simulation for its integration and coordination in an industrial production environment.

Actually, digital continuity needs to be empowered by a standardized, open semantic meta-model capable of fully describing the properties and functional characteristics of the CPS simulation models, as a key element to empower multidisciplinary simulation tools. The hereby described meta-model is able to provide a cross-tool representation of the different specific simulation models defining both static information (3D models, kinematics chains, multi-body physics skeletons, etc.) and behavioural information (observable properties, inverse kinematics processors, motion-low computation functions, resource consumption logics, etc.).

10.1 Introduction

In order to empower simulation methodologies and multidisciplinary tools for the design, engineering and management of CPS-based (Cyber Physical Systems) factories, we need to target the implementation of actual digital continuity, defined as the ability to maintain digital information all along the factory life cycle, despite changes in purpose and tools.

A Semantic Data Model for CPS representation is the foundation to achieve digital continuity, because it provides a unified description of the CPS-based simulation models that different simulation tools can rely on to operate.

Cyber Physical Systems are engineered systems that offer close interaction between cyber and physical components. CPS are defined as the systems that offer integrations of computation, networking, and physical processes, or in other words, as the systems where physical and software components are deeply intertwined, each operating on different spatial and temporal scales, exhibiting multiple and distinct behavioural modalities, and interacting with each other in a myriad of ways that change with context [2, 3]. From this definition, it is clear that the number and complexity of features that a CPS data model has to represent are very high, even if limited to the simulation field. Moreover, many of the aspects that concur to define a CPS for simulation (3D models, kinematics structures, dynamic behaviours, etc.) have been already investigated and formalized by many well-established data models that are, or can be considered, to all extents data exchange standards.

For these reasons, the goal of an effective CPS Semantic Data Model is providing a gluing infrastructure that refers existing interoperability standards and integrates them into a single extensible CPS definition. This approach reduces the burden on the simulation software applications to access the new data structures because they mainly add a meta-information level whereas data for specific purposes is still available in standard formats.

AutomationML [1] is a standard technology that is based on this "Integration philosophy" and defines the semantics of many elements of the manufacturing systems so that it is suitable to be adopted as the foundation of our CPS Semantic Data Model.

10.2 Adoption of AutomationML Standard

The meta-data model needs basis on which data is saved and processed. The goal of AutomationML is to interconnect engineering tools in their different disciplines, e.g. mechanical plant engineering, electrical design, process engineering, process control engineering, HMI development, PLC programming, robot programming, etc. It is a standard focused on data exchange in the domain of automation engineering, defined in four whitepapers that focus each on one of the following aspects:

1. Architecture and general requirements;
2. Role class libraries;
3. Geometry and kinematics;
4. Logic.

The data exchange format defined in these documents is the Automation Markup Language (AML), an XML schema-based data format and has been developed in order to support the data exchange in a heterogeneous engineering tools landscape for the production.

Engineering information is stored following the Object-Oriented Paradigm, and physical and logical plant components are modelled as data objects encapsulating different aspects. An object may consist of other sub-objects and may itself be part of a larger composition or aggregation. Typical objects in plant automation comprise information on topology, geometry, kinematics and logic, whereas logic comprises sequencing, behaviour and control. Therefore, an important focus in the data exchange in engineering is the exchange of object-oriented data structures, geometry, kinematics and logic.

AML combines existing industry data formats that are designed for the storage and exchange of different aspects of engineering information. These data formats are used on an "as-is" basis within their own specifications and are not branched for AML needs. The core of AML is the top-level data format CAEX that connects the different data formats (e.g. COLLADA for geometries or PLCOPEN-XML for logic). Therefore, AML has an inherent distributed document architecture. The goals and basic concepts of AutomationML are well aligned with our objectives, and it can be used as the base of the semantic meta data model; nonetheless, it is mainly a specification for data exchange only and it falls short when it comes to describe some operational aspects of simulation. For these reasons, we decided to extend AML aiming at targeting a more integrated connection between real/digital CPS and simulation tools.

10.3 Meta Data Model Reference

This chapter documents the *Meta Data Model* developed. It is organized into eight sections that correspond to the eight main semantic areas in which the data model is organized:

1. *Base Model (§10.3.1)*: documents low-level utility classes that are used for the definition of high-level classes of the other sections.
2. *Assets and Behaviours (§10.3.2)*: documents, classes and concepts related to the possibility of using external data sources to define additional resource models.
3. *Prototypes Model (§10.3.3)*: introduces the concepts of resource prototypes and resource instances that are at the basis of the model reuse paradigm and documents the classes defining the resource model prototypes.
4. *Resources Model (§10.3.4)*: documents all the classes related to representation of intelligent and passive resources constituting the model of a manufacturing plant.
5. *Device Model (§10.3.5)*: documents all the classes related to the representation of the data connection with the physical devices, including the definition of all the relevant I/O signals that are exchanged with the digital counterpart.
6. *Project Model (§10.3.6)*: documents all the classes that represent complex multi-disciplinary simulation projects and that enable simulation tools to share plant models and results.
7. *Product Routing Model (§10.3.7)*: documents all the classes related to the definition of a discrete product, of the manufacturing processes and of the production plans that should be used for plant simulation.
8. *Security Model (§10.3.8)*: documents the classes that are related to the access control and that define the authentication and authorization levels needed to work on a certain resource.

Each section is introduced with a diagram view (based on UML Class Diagram) that contains only the classes composing that specific data model area and their relationships with the main classes belonging to the other data model areas. Therefore, it is possible to find the same class representation (e.g. Property class) in many different diagrams, but each class is documented only once in the proper semantic section.

10.3.1 Base Model

This section documents some low-level and general-purpose classes that are shared by other higher-level models described in the following sections. In particular, the classes related to the possibility of modelling generic, simple and composite properties of plant resources are documented (Figure 10.1).

10.3.1.1 Property

Property is an abstract class derived by IdentifiedElement and represents runtime properties of every resource and prototype. These properties are relevant information that can be dynamically assigned and read by the simulation tools.

10.3.1.2 CompositeProperty

CompositeProperty is a class derived by Property and represents a composition of different properties of every resource and prototype. This composition is modelled to create a list of simple properties of the resource, or even a multilevel structure of CompositeProperty instances. Figure 10.2 shows a possible application of the base model classes to represent properties, meta information and documentation of a sample CPS. A resource (in this case, CPS4) can have many properties instances associated to it and these properties can be simple (as ToolLength, EnergyConsumption and TempCPS4) or composite that allow creating structured properties (CurrProd).

10.3.2 Assets and Behaviours

The goal of the CPS Semantic Data Model is providing a gluing infrastructure that refers existing interoperability standards and integrates them into a single extensible CPS definition. For this reason, the implemented model includes the mechanisms to reference external data sources (Figure 10.3).

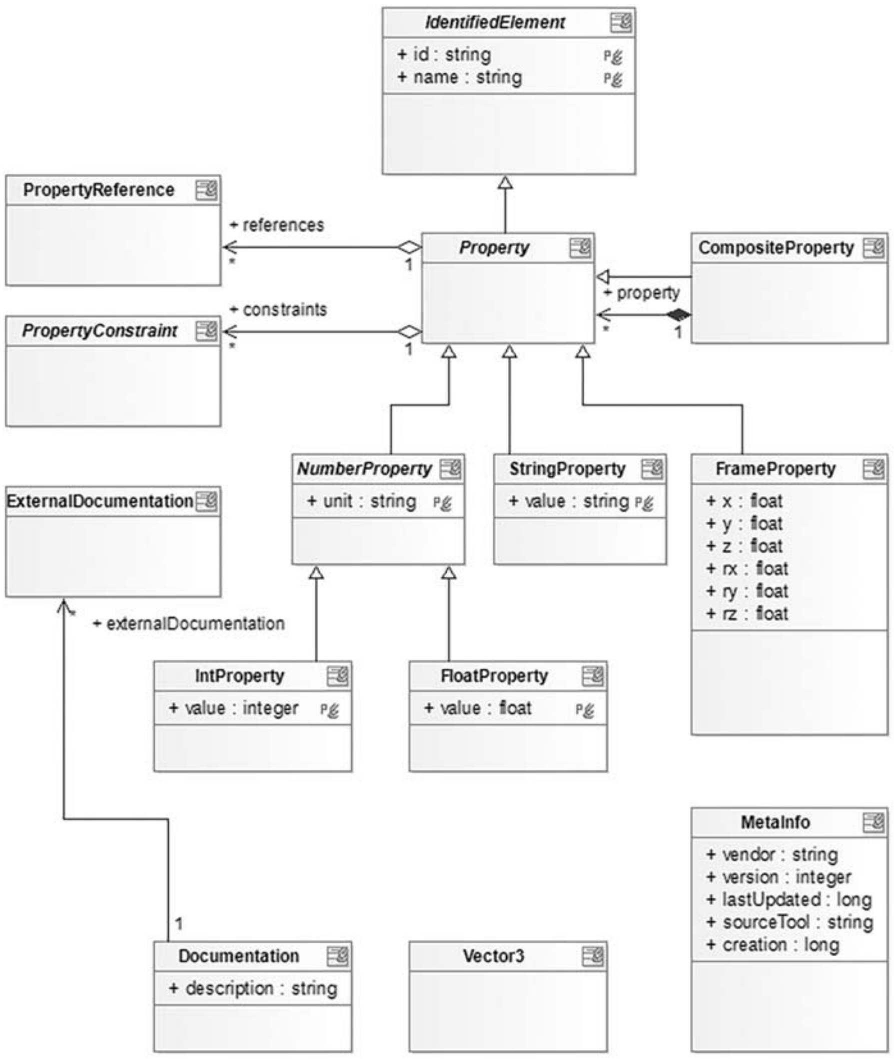

Figure 10.1 Class diagram of the base classes.

10.3.2.1 ExternalReference

ExternalReference is abstract and extends IdentifiedElement. This class represents a generic reference to a data source that is external to the *Meta Data Model* (e.g. a file stored on the Central Support Infrastructure (CSI, see Chapter 13)). The external source can contain any kind of binary

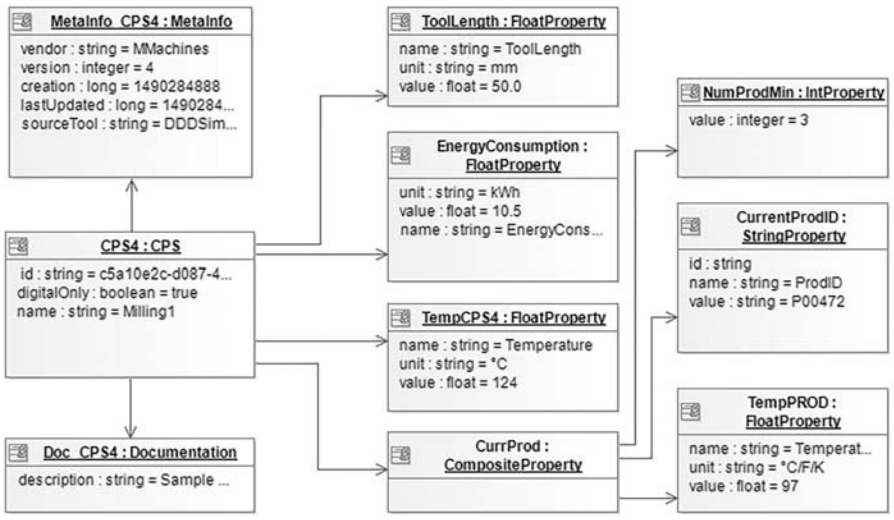
MetaInfo_CPS4 : MetaInfo
vendor : string = MMachines
version : integer = 4
creation : long = 1490284888
lastUpdated : long = 1490284...
sourceTool : string = DDDSim...
CPS4 : CPS
id : string = c5a10e2c-d087-4...
digitalOnly : boolean = true
name : string = Milling1
Doc_CPS4 : Documentation
description : string = Sample ...
ToolLength : FloatProperty
name : string = ToolLength
unit : string = mm
value : float = 50.0
EnergyConsumption : FloatProperty
unit : string = kWh
value : float = 10.5
name : string = EnergyCons...
TempCPS4 : FloatProperty
name : string = Temperature
unit : string = °C
value : float = 124
CurrProd : CompositeProperty
NumProdMin : IntProperty
value : integer = 3
CurrentProdID : StringProperty
id : string
name : string = ProdID
value : string = P00472
TempPROD : FloatProperty
name : string = Temperat...
unit : string = °C/F/K
value : float = 97

Figure 10.2 Object diagram of the base model.

data in proprietary or interoperable format, depending on the type of resource. Using external references allows avoiding re-defining data models and persistency formats for all the possible technical aspects related to a certain resource. The approach that has been adopted is like AutomationML one, where additional data is stored in external files using already existing standards (e.g. COLLADA for 3D models or PLCopen for PLC code).

10.3.2.2 Asset

Asset is an extension of ExternalResource. This class represents a reference to an external relevant model expressed according to interoperable standard or binary format that behavioural models want to use. An important feature that the CPS data model should support is the possibility to create links between runtime properties and properties defined inside assets and between properties defined by two different assets. Assets can be considered static data of the CPS because they represent self-contained models (e.g. 3D Models) that should be slowly changing.

10.3.2.3 Behaviour

Behaviour is an extension of ExternalResource. This class represents a reference to runnable behavioural models that implement: (i) functionalities and operative logics of the physical systems and (ii) raw data stream aggregation

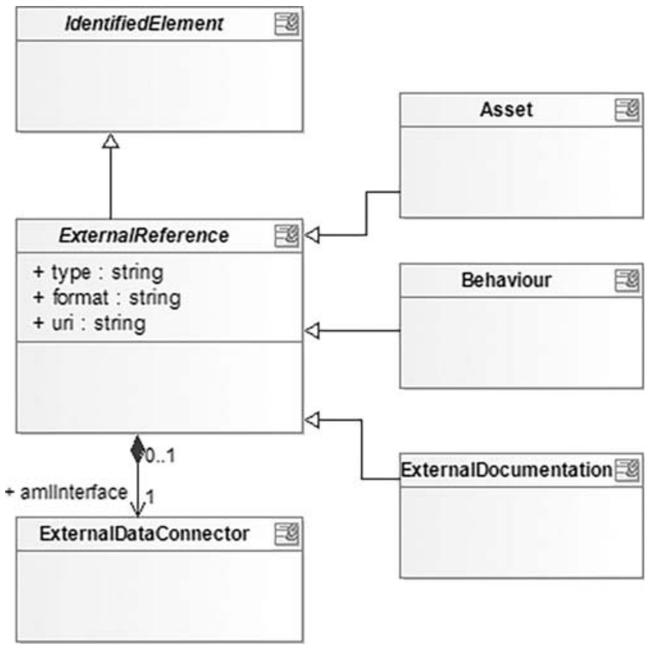

Figure 10.3 Class diagram for assets and behaviours.

and processing functions. Simulation Tools should be able to use directly the former to improve reliability of simulations, whereas the latter should run inside the CSI to update the runtime properties of the CPS model.

10.3.3 Prototypes Model

This section is meant to describe the classes and concepts related to the definition of prototype resources that can be defined once and reused many times to create different plant models.

10.3.3.1 Prototypes and instances

One of the most exploited features of manufacturing plants is the fact that they are mostly composed of standard “off-the-shelf” components (machine tools, robots, etc.) that are composed in a modular way. Thanks to this and with a good organization of modules, in fact, it is possible to speed up the simulation set up, reusing as much as possible already developed models. For this reason, usually simulation software tools adopt a mechanism based on the definition of libraries of models that can be applied to assemble a full plant layout.

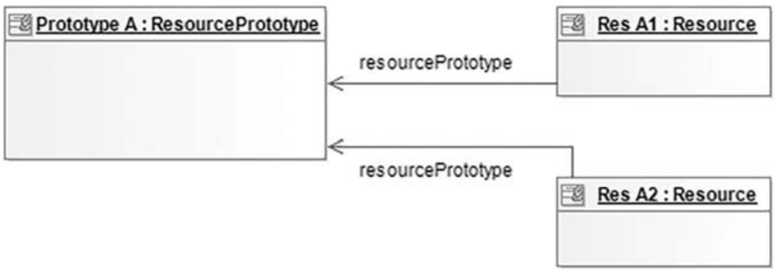

Figure 10.4 Prototype-resource object diagram.

The data model aims at natively supporting the same efficient re-use approach implementing the classes to describe "ready to use" resources, called "prototypes" and "instances" of such elements that are the actual resources composing plants. The relationship that exists between prototypes and instances is the same that in OOP exists between a class and an object (instance) of that class.

A prototype is a Resource model that is complete from a digital point of view, but it is still not applied in any plant model. It contains all the relevant information, assets and behaviours that simulation tools may want to use and, ideally, device manufacturers should directly provide Prototypes of their products ready to be assembled into production line models.

As shown in Figure 10.4, a Resource instance is a ResourcePrototype that has become a well-identified, specific resource of the manufacturing plant. Each instance shares with its originating Prototype the initial definition, but during life cycle, its model can diverge from the initial one because properties and even models change. Therefore, a single ResourcePrototype can be used to instantiate many specific resources that share the same original model.

10.3.3.2 Prototypes and instances aggregation patterns

An important aspect that *Meta Data Model* defines is the one related to the composition of resources into higher-level resources. This concept is at the basis of the creation of a hierarchy of resources within a plant and it is an intrinsic way of organizing the description of a manufacturing system. Nevertheless, depending on each specific discipline, there are many ways resource instances (and therefore CPSs) can be grouped in a hierarchical structure. For example, spatial relationships define the topological hierarchy of a system, but from a safety grouping or electrical perspective, the same resources should be organized into different hierarchies (e.g. in the

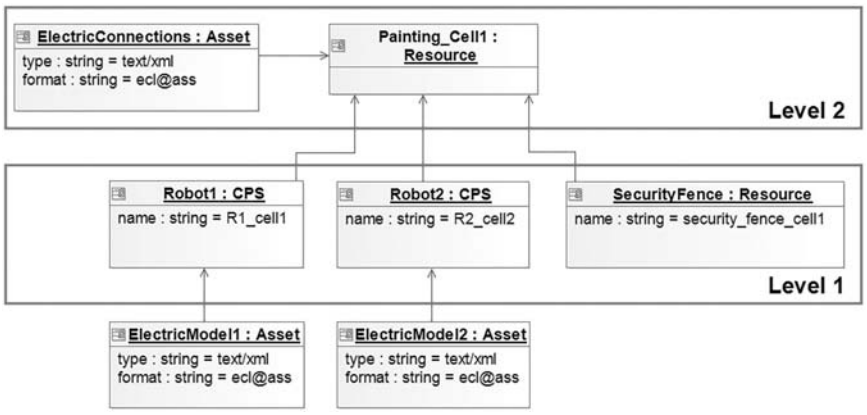

Figure 10.5 Example of usage of main and secondary hierarchies.

automotive, a cell safety group contains the robot and the surrounding fences, but from an electrical point of view, fences are not represented at all).

For this reason, *Meta Data Model* provides an aggregation system that is based on two levels:

- a first main hierarchy structure that is implemented in the two base classes for prototypes and instances, AbstractResourcePrototype and AbstractResource (Figure 10.6);
- a second level, discipline-dependent, that is defined in parallel to the main one and that should be contained inside domain-specific Assets.

The former hierarchy level is meant to provide a reference organization of the plant that enables both simulation tools and the CSI to access resources in a uniform way. In fact, the main hierarchy has the fundamental role of controlling the "visibility level" of resources, setting the lower access boundaries that constrain the resources to which the secondary ("parallel") hierarchies should be associated.

Figure 10.5 shows an example of application of the main resources hierarchy and the secondary, domain-specific one. The main hierarchy organizes the two robots and the surrounding security fence with a natural logical grouping since Robot1, Robot2 and SecurityFence belong physically to the same production cell, Painting_Cell1. Even if this arrangement of the instances is functional from a management point of view, it is not directly corresponding to the relationships defined in the electrical schema of the plant, for which the only meaningful resources are the two robots. Imagining that an electric connection exists between the two robots, a secondary, domain-specific schema (in this case, the domain is the electric design) needs to be defined separately. The Painting_Cell1 resource acts as the aggregator of the two robot CPS; therefore, it has the "visibility" on the two resources of the lower level (Level 1), meaning that they exist and it knows how to reference them. For this reason, the electrical schema that connects Robot1 and Robot2 is defined at Level 2 as the "ElectricConnections" Asset associated to the Painting_Cell1. This asset, if needed, is allowed to make references to each electric schema of the lower-level resources.

10.3.3.3 AbstractResourcePrototype

AbstractResourcePrototype is abstract and extends IdentifiedElement (see Figure 10.6). It represents the base class containing attributes and relationships that are common both to prototypes of intelligent devices and to prototypes of simple passive resources or aggregation of prototypes. The main difference between prototype and instance classes is that the former does not have any reference to a Plant model, because they represent "not-applied" elements.

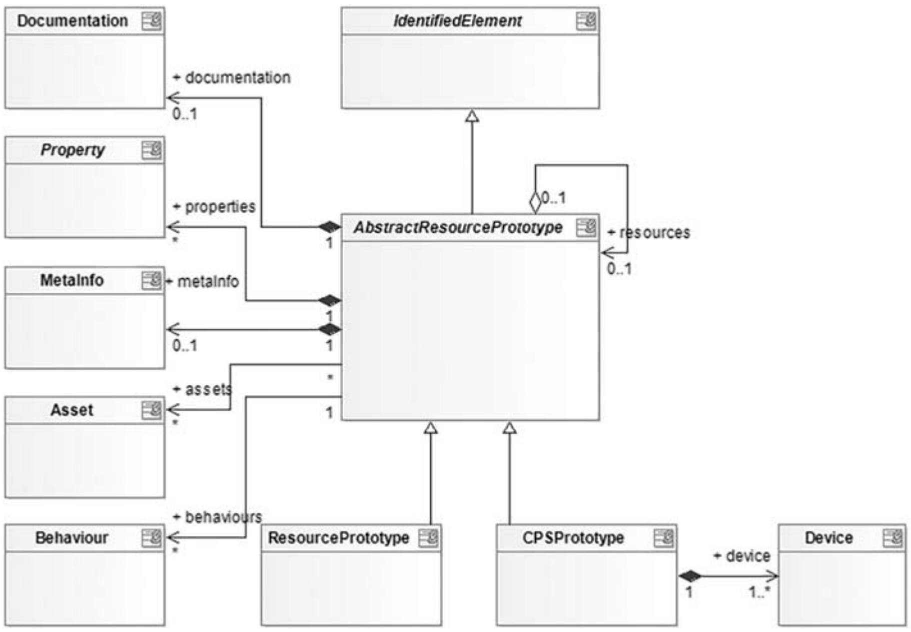

Figure 10.6 Prototype Model class diagram.

Each AbstractResourcePrototype can aggregate other AbstractResource-Prototype (i.e. CPSPrototype and ResourcePrototype instances), and it can use its Assets and Behaviours to create higher-level complex models and functionalities starting from the lower-level ones.

10.3.3.4 ResourcePrototype

ResourcePrototype extends AbstractResourcePrototype. This class represents the prototype of a generic passive resource of the plant that does not have any electronic equipment capable of sending/receiving data to/from its digital counterpart, or an aggregation of multiple resource prototypes. Examples of simple resources are cell protection fences, part positioning fixtures, etc.

Resource class is the direct instance class of a ResourcePrototype.

Since a ResourcePrototype must be identifiable within the libraries of prototypes, its ID attribute should be set to a valid UUID that should be unique within an overall framework deployment.

10.3.3.5 CPSPrototype

CPSPrototype extends AbstractResourcePrototype. This class represents a prototype of an "intelligent" resource that is a resource equipped with an electronic device, capable of sending/receiving data to/from its digital counterpart. A CPSPrototype defines the way its derived instances should connect to the physical devices to maintain synchronization between shop floor and simulation models. CPS class is the direct instance class of a CPSPrototype. Since a CPSPrototype must be identifiable within the libraries of prototypes, its ID attribute should be set to a valid UUID that should be unique within an overall framework deployment.

10.3.4 Resources Model

From *Meta Data Model* perspective, each simulated plant can be represented as a bunch of resources (machine tools, robots, handling systems, passive elements, etc.). Each resource can have a real physics counterpart to which it can be connected or defined from a product life cycle management point of view. This section of the model is meant to document the classes that support the description of resource instances (see §**10.3.4.1 Prototypes and instances** for the definition of the instance concept).

10.3.4.1 AbstractResource

AbstractResource is abstract and extends IdentifiedElement (Figure 10.7). This class represents the generalization of the concept of plant resource. As cited at the beginning of the section, a plant is a composition of intelligent devices (e.g. machines controlled by PLC, IoT ready sensors, etc.) or passive elements (fences, fixtures, etc.). Even if such resources are semantically different, from a simulation point of view, they have a certain number of common properties. This fact justifies, from a class hierarchy perspective, the definition of a base class that CPS and Resource classes extend.

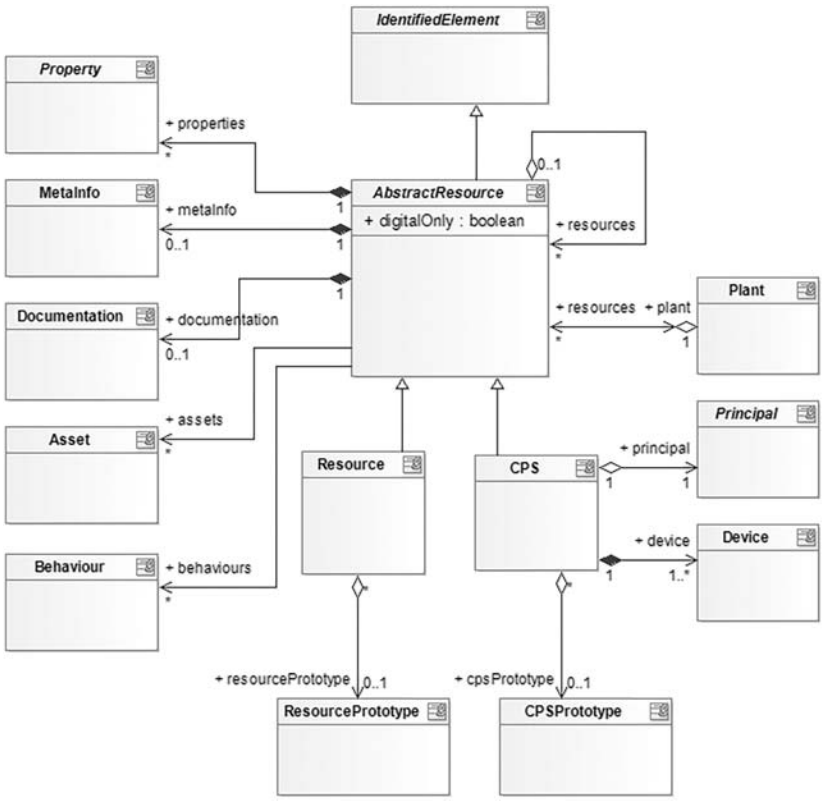

Figure 10.7 Class diagram of resources section.

An AbstractResource is identified by its ID, which must be unique within the same plant.

Field	Type	Description
digitalOnly	Boolean	This flag indicates whether this resource (be it a CPS or a simple resource) has a physical counterpart somewhere in the real plant or if it is purely a virtual element. In design phase of a plant that goes on green field, resources will all have digitalOnly = true, while during the reconfiguration of a plant, there will be a mixed condition with some resources having the flag set to true (the ones existing in the running production lines) and some others set to false (the ones that are going to be evaluated with simulation).
properties	Property[]	Runtime properties of the resource. Each property of the resource represents a relevant piece of information that can be shared (accessed and modified) by the simulation tools and by the functional and behavioural models. The length of the array can be 0 to n.
resources	AbstractResource[]	List of the resources that this instance aggregates. This field implements the hierarchy relationships among resources inside a plant. See §**Prototypes and instances aggregation patterns**. The length of the array can be 0 to n.

10.3.4.2 CPS

CPS extends AbstractResource. This class represents each "intelligent" device belonging to the plant equipped with an electronic device capable of sending/receiving data to/from its digital counterpart. A CPS can be connected with the physical device to maintain synchronization between shopfloor and simulation models. A CPS can be an aggregation of other CPSs and simple Resources, using its Assets and Behaviours to aggregate lower-level models and functionalities.

Each CPS must be identified by a string ID that must be unique within the plant.

Field	Type	Description
cps-Prototype	CPS-Prototype	Each CPS can be an instantiation of a prototype CPS that has been defined in a library of models (usually stored in the CSI) that simulation tools can access and use. See §**10.3.4.1 Prototypes and instances**. This field can be null if the CPS does not derive from the instantiation of a prototype.
device	Device	Represents the description of the device that ensures the data connection between the physical and digital contexts. This object characterizes all the I/Os that can be received and sent from and to the real equipment. This field cannot be null, while it is possible that the device, even if fully defined, is not connected to real electronic equipment.
principal	Principal	Each CPS has a related access level that is defined in compliance with the security data model described in section "Security Data Model" and implemented by the CSI.

10.3.5 Device Model

This section contains the documentation of the classes needed to model the electronic equipment of the intelligent resources. This equipment is described in terms of the interfaces that can be used by the digital tools to open data streams with the real devices (Figure 10.8).

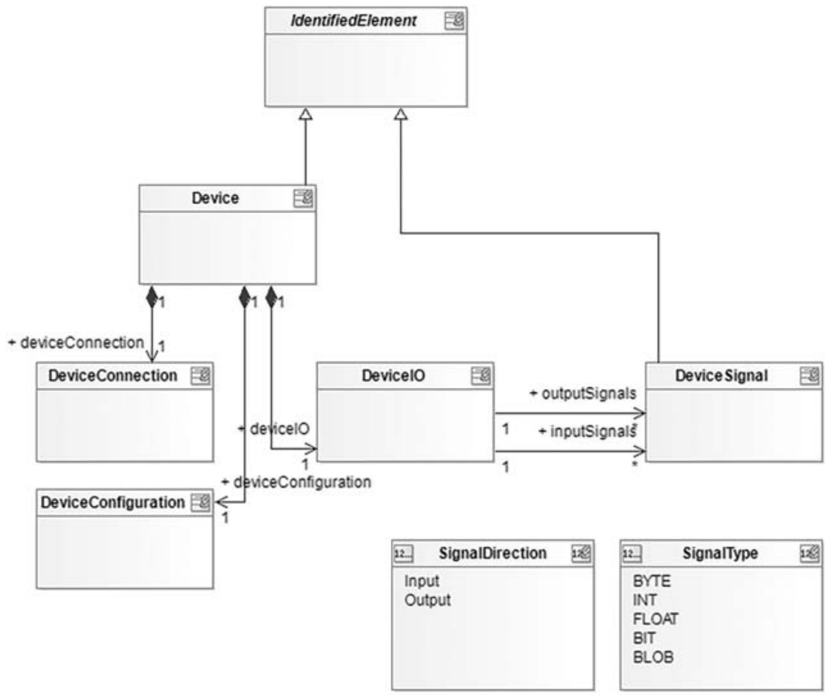

Figure 10.8 Class diagram of devices section.

10.3.5.1 Device

Device is an IdentifiedElement and represents an electronic equipment of physical layer that can be connected to the digital counterpart to send/receive data.

Field	Type	Description
device-Connection	Device-Connection	It contains all the details to open data streams with the physical device. E.g. for Ethernet-based connections, it contains IP address as well as information on ports, protocols and possibly the security parameters to apply to receive access rights to the specific resource. The field can be null.
device-Configuration	Device-Configuration	It contains details on the device hardware and software configuration (e.g. version of the running PLC code). This object can be updated dynamically based on data read from the physical

Field	Type	Description
		device to reflect the actual working condition of the device. The field can be null.
deviceIO	DeviceIO	It contains the map of Input/Output data signals that can be exchanged with the physical device. The field cannot be null. If no signal can be exchanged with the device, the DeviceIO map is present but empty. Normally, this should not happen (except during the drafting phase) because if a device does not allow any data exchange with its digital counterpart, then it should be treated as a passive resource.

10.3.5.2 DeviceIO

DeviceIO represents a map of input and output signals that can be exchanged with a specific device. Moreover, the DeviceIO represents the communication between CPS on IO-Level.

Field	Type	Description
input-Signals	Device-Signal[]	Array of DeviceSignal describing input signals. Signal direction is seen by the device; therefore, this is the list of data that can be sent TO the device. The field cannot be null. Length of the array can be 0 to n All DeviceSignal instances belonging to this collection must have *direction* attribute set to *SignalDirection.Input.*
output-Signals	Device-Signal[]	Array of DeviceSignal describing output signals. Signal direction is seen by the device; therefore, this is the list of data that can be received FROM the device. The field cannot be null. Length of the array can be 0 to n All DeviceSignal instances belonging to this collection must have *direction* attribute set to *SignalDirection.Output.*

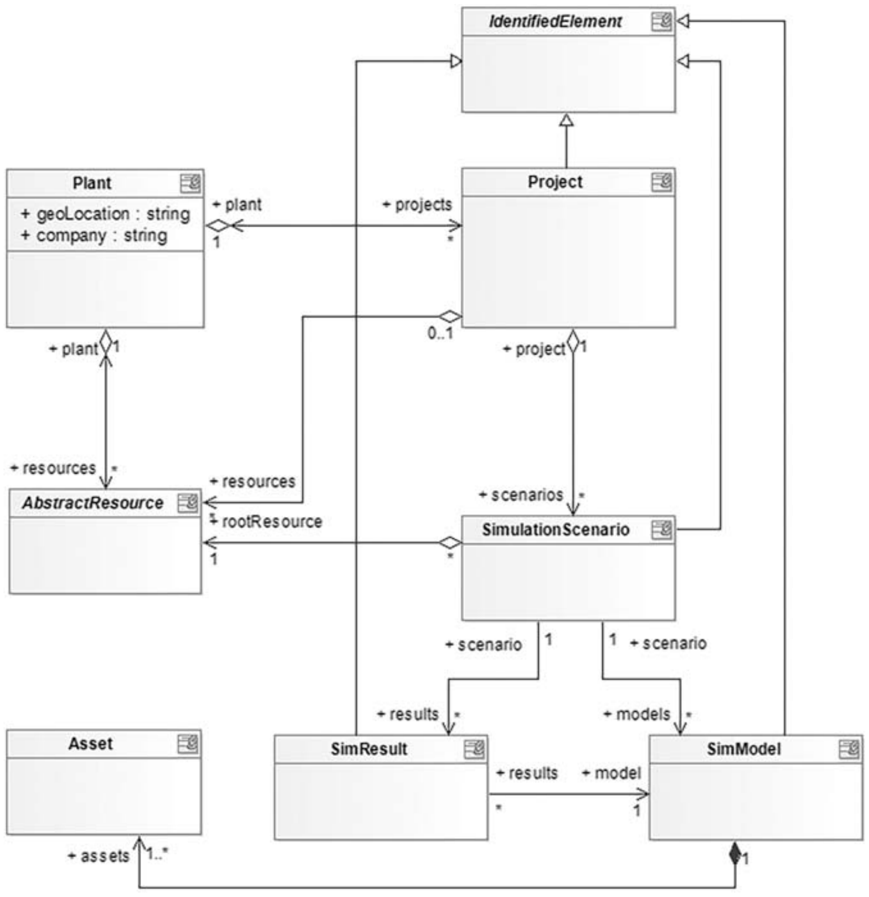

Figure 10.9 Class diagram of the Project Model section.

10.3.6 Project Model

This section describes the classes related to the management of projects, scenarios and results of simulations for a certain plant that are produced and consumed by simulation tools (Figure 10.9).

10.3.6.1 Project

A project is an IdentifiedElement. It can be considered mainly as a utility container of different simulation scenarios that have been grouped together because they are related to the same part of the plant (e.g. different scenarios for the same painting cell of the production line).

A project could identify a design or a reconfiguration of a part of the plant for which each SimulationScenario represents a hypothesis of layout of different resources.

10.3.6.2 Plant

Plant is an extension of IdentifiedElement and represents an aggregation of projects and resources. A plant instance could be considered as an entry point for simulation tools that want to access models stored on the CSI. It contains references to all the resource instances that are subject of SimulationScenarios. In this way, it is possible to have different simulation scenarios, even with simulation of different types, bound to a single resource instance.

Note: the fact that different simulations of different nature can be set up for the same resource (be it a cell, a line, etc.) is not related to the concept of multi-disciplinary simulation that is, instead, implemented by the Simulation Framework and refers to the possibility of running concurrent, interdependent simulations of different types.

The ID of the Plant must be unique within the overall framework deployment.

10.3.6.3 SimulationScenario

SimulationScenario is an extension of IdentifiedElement and represents the run of a SimModel producing some SimResults. A simulation scenario refers to a root resource that is not necessarily the root resource instance of the whole plant, because a simulation scenario can be bound to just a small part of the full plant. A simulation scenario can set up a multi-disciplinary simulation, defining different simulation models for the same resource instance to be run concurrently by the Simulation Framework.

10.3.6.4 SimModel

SimModel is an IdentifiedElement and represents a simulation model within a particular SimulationScenario. Each model can assemble different behavioural models of the root resource into a specific simulation model, creating scenario-specific relationships that are stored inside simulation assets that can be expressed both in an interoperable format (e.g. AutomationML) when there is need for data exchange among different tools and in proprietary formats.

The ID of a SimModel instance must be unique within a Simulation Scenario.

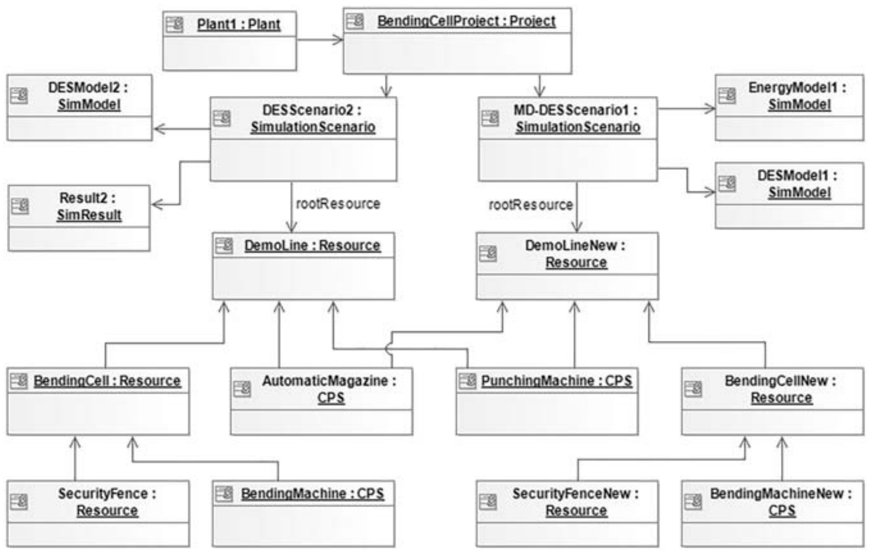

Figure 10.10 Object diagram of the Project Model.

The object diagram shown below (Figure 10.10) shows a possible application of the Project Model: a set of simple resources and CPS is organized into two hierarchies: one representing the actual demo line and a second hierarchy modelling a hypothesis of redesign of the demo plant. All the Resource and CPS instances belong to the plant model Plant1 (relationships in this case have not been reported to keep the diagram tidy). The user wants to perform two different simulations, one for each root resource. For this reason, he/she sets up two SimulationScenario instances: MD-DESScenario1 and DESScenario2. Each one refers to a different root resource. The former is a multi-disciplinary scenario of the DemoPlantNew that will use a combination of a DES model and an Energy Consumption model, while the latter represents a simple DES-only scenario of the original DemoPlant. These scenarios are aggregated in a Project instance (BendingCellProject) that belongs to the Plant1 project and that is meant to compare the performance of the plant using two different setups of the bending cell. For DESScenario2, there are already simulation results Result2.

10.3.7 Product Routing Model

In this paragraph, a description of the product routing section of the meta data model is given. Structural choices as well as requirements consideration are reported, with a particular focus on the validation points that have been reviewed by experts. In order to describe this part of the model, each class is treated separately and clusters of functional areas have been created for simplicity. All attributes, cardinality indications and relationships are described with respect to the single entity and in the general data model perspective.

10.3.7.1 Relationship between product routing model and ISO 14649-10 standard

The product routing section of the data model has been developed according to the ISO 14649-10 standard, "Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 10: General process data", which was deeply analysed and chosen as best-fitting standard for the product feature – operation coupling part. Its characteristics and focus areas are suitable from the functional point of view, as it tackles some aspects that the model needs to cover in exactly the same application environment. In fact, it supports the communication between CAD and CNC. ISO 14649-10 specifies the process data that is generally needed for NC programming in any of the possible machining technologies. These data elements describe the interface between a computerized numerical controller and the programming system (i.e. CAM system or shopfloor programming system). On the programming system, the programme for the numerical controller is created. This programme includes geometric and technological information. It can be described using this part of ISO 14649 together with the technology-specific parts (ISO 14649-11, etc.). This part of ISO 14649 provides the control structures for the sequence of programme execution, mainly the sequence of working steps and associated machine functions.[1] The standard ISO 14649-10 gives a set of terms and a certain hierarchy among them, though without specifying the type of relations. Being focused on process data for CNC (Computerized Numerical Control), the terminology is deeply technical in describing all different types of manufacturing features, mechanical parameters and measures. The relationship between workpiece features, operations and sequencing is of relevance for the purpose of this work, so a number of entities have been selected. Only after that, the distinction between classes and attributes was made, together with the definition of the types of relationships and references among the classes.

[1]ISO 14649. http://www.iso.org/iso/catalogue_detail?csnumber=34743

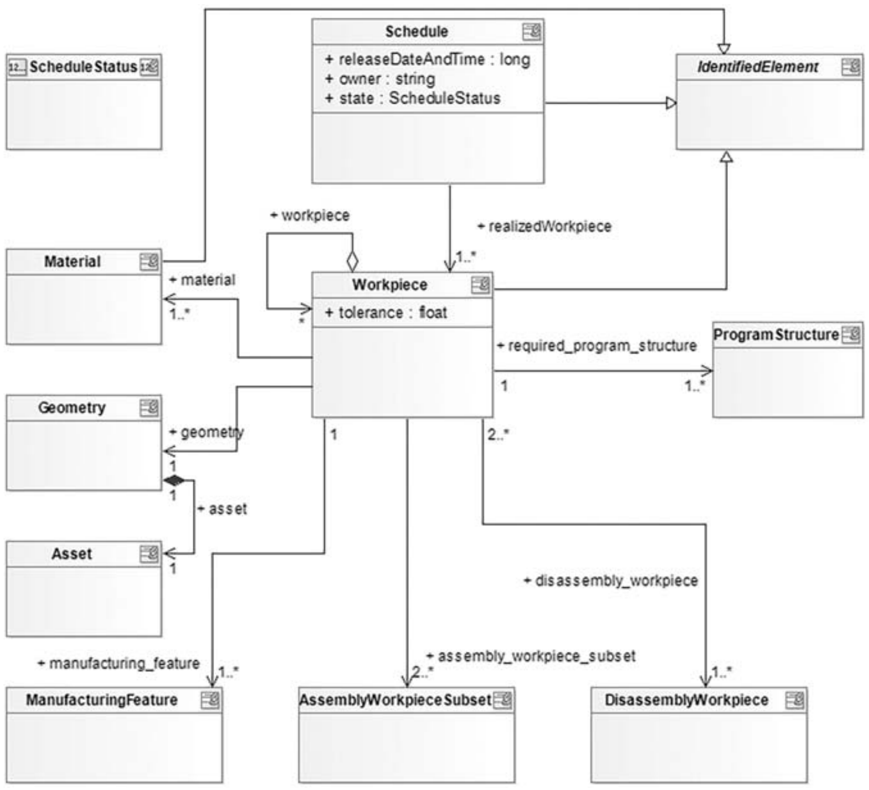

Figure 10.11 Schedule and workpiece representation.

10.3.7.2 Workpiece

Workpiece class (Figure 10.11) represents the part or product that needs to be machined, assembled or disassembled. Each schedule realizes at least one workpiece, but it may also realize different product variants, with various features. Each product variant is a different instantiation of the class "Workpiece" and extends the IdentifiedElement class. Being a central entity for the data model, the workpiece has a further development side that concerns the production scheduling and product routing. Manufacturing methods and instructions are not contained in the workpiece information but are determined by the operations themselves.

10.3.7.3 ProgramStructure

ProgramStructure determines how the different operations are executed for a specific work piece, i.e. in series or parallel (see also Figure 10.12).

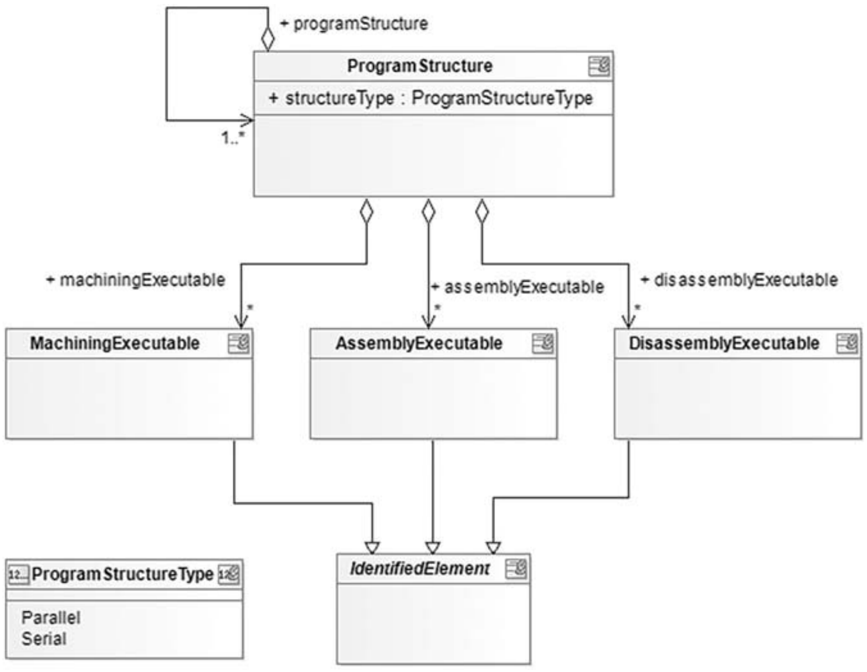

Figure 10.12 Program structure representation.

A program structure, at low level, is composed of single, ordered steps, called "Executables". Depending on the type of program structure, the executables are realized in series or parallel. The program structure thus defines how the different steps are executed and at the same time gives some flexibility in the choice, by taking into account data from the system.

10.3.7.4 ProgramStructureType

Enumeration representing the allowed types of a ProgramStructure instance (Figure 10.12).

10.3.7.5 MachiningExecutable

Machining executables initiate actions on a machine and need to be arranged in a defined order. They define all those tasks that cause a physical transformation of the workpiece. MachiningExecutable class extends the IdentifiedElements class and is a generalization of machining working steps and machining NC functions, since both of these are special types of machining executables. Hierarchically, it is also a sub-class of program structures, being their basic units, as it constitutes the steps needed for the execution of the program structure. Starting from the machining executable, the connected classes are represented in Figure 10.12.

10.3.7.6 AssemblyExecutable

AssemblyExecutable also extends IdentifiedElement class. AssemblyExecutable are a specialization of program structures and generalizations of working steps or NC functions. As in the case of machining executables, they initiate actions on a machine and need to be arranged in a defined order: assembly executables include all those operations that allow creating a single product from two or more work pieces. Starting from the assembly executable, the connected classes are represented in Figure 10.12.

10.3.7.7 DisassemblyExecutable

DisassemblyExecutable is derived from IdentifiedElement. DisassemblyExecutables are generalizations of working steps or NC functions. As in the case of machining and assembly executables, they are also a specialization of program structures, being their basic units, as these three classes constitute the steps needed for the execution of the program structure. Thus, it can be imagined that one or more machining executables, one or more assembly executables and one or more disassembly executable compose program structure. Disassembly executables also initiate actions on a machine and need to be arranged in a defined order: disassembly executables perform an opposite activity with respect to assembly, which means that from a single part it extrapolates more than one part. Starting from the disassembly executable, the connected classes are represented in Figure 10.12.

10.3.7.8 MachiningNcFunction

MachiningNcFunction is an IdentifiedElement and a specialization of MachiningExecutable (Figure 10.13) that differentiates from the machining working step for the fact that it is a technology-independent action, such as a handling or picking operation or rapid movements. It has a specific purpose and given parameters. If needed, other parameters regarding speed or other technological requirements can be added as attributes.

10.3.7.9 MachiningWorkingStep

MachiningWorkingStep is an IdentifiedElement that is also a specialization of MachiningExecutable, the most important one for the purpose of this work. It is the machining process for a certain area of the workpiece, and as such,

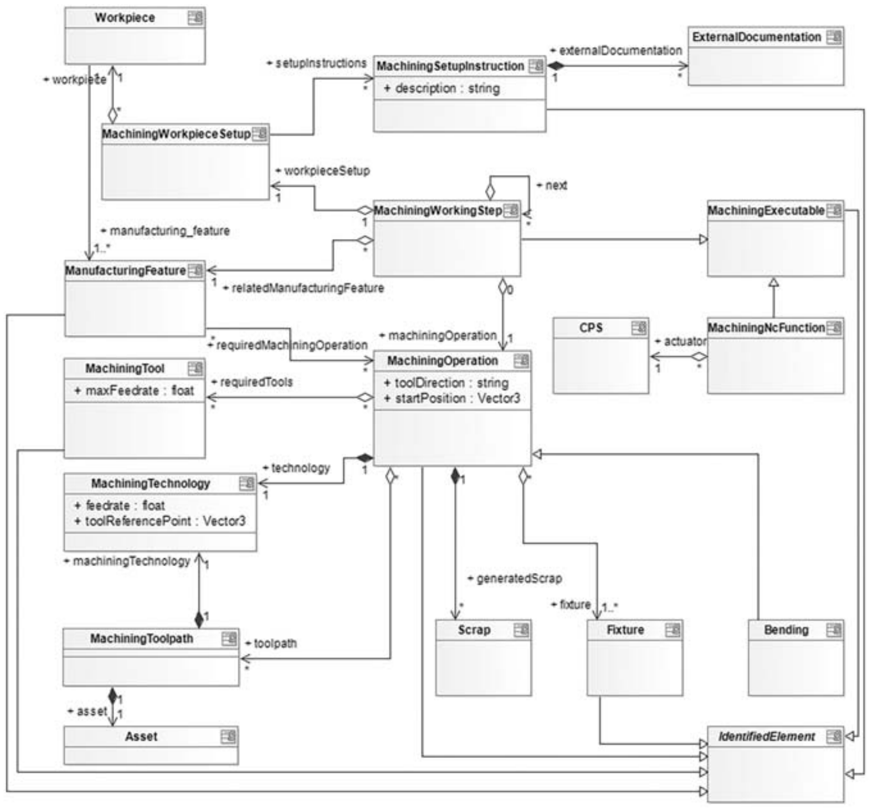

Figure 10.13 Machining executable representation.

it is related to a technology like milling, drilling or bending. It cannot exist independent of a feature, but rather specifies the association between a distinct feature and an operation to be performed on the feature. It creates an unambiguous specification, which can be executed by the machine. An operation can be replicated for different features, while a working step is unique in each part program as it spans for a defined period of time and relates to a specific workpiece and a specific manufacturing feature. Each working step thus defines the conditions under which the relative operation has to be performed. This means also that the operation related to the machining working step must be in the list of possible operations related to a certain manufacturing feature (Figure 10.13).

10.3.7.10 MachiningWorkpieceSetup

MachiningWorkpieceSetup has a direct reference to the workpiece and is defined for each machining working step, since it defines its position for machining. In fact, it may change according to the position of the single machining feature on the workpiece. In fact, also the reference to the manufacturing feature for which it is defined is unique: a single workpiece setup, in fact, refers to only one machining working step that is meant to realize a defined feature.

10.3.7.11 MachiningSetupInstructions

For each single operation in time and space, precise setup instructions may be specified, connected to the workpiece setup, such as operator instructions and external material in the forms of tables, documents and guidelines. MachiningSetupInstructions class extends the IdentifiedElement class.

10.3.7.12 ManufacturingFeature

ManufacturingFeature is an IdentifiedElement that is a characteristic of the workpiece, which requires specific operations. For 3D simulation and Computer Aided Design, it is fundamental to have the physical characteristics specifications: as shown in Figure 10.13, the workpiece manufacturing features are a relevant piece of information for modelling and simulation, as they determine the required operations.

10.3.7.13 MachiningOperation

MachiningOperation is an IdentifiedElement that specifies the contents of a machining working step and is connected to the tool to be used and a set of technological parameters for the operation. The tool choice depends on the specific working step conditions (Figure 10.13). The more information is specified for tool and fixture, the more limited the list of possible matches is. Therefore, only the relevant, necessary values should be specified.

10.3.7.14 MachiningTechnology

MachiningTechnology collects a set of parameters, such as feed rate or tool reference point. The addition of new attributes would expand the possibilities of technological specifications.

10.3.7.15 FixtureFixture

Fixture class is an IdentifiedElement that represents the fixtures required by machining operations, if any. Given that the same operation may be performed under different conditions, the choice of a fitting fixture is done for the single working step.

10.3.7.16 Assembly and disassembly

In Figures 10.14 and 10.15, assembly-executable and disassembly-executable branches are examined, even though their development is very similar to the machining executable branch. In fact, they differ only for a low number of details and specifications. These differences are presented in the following subsections.

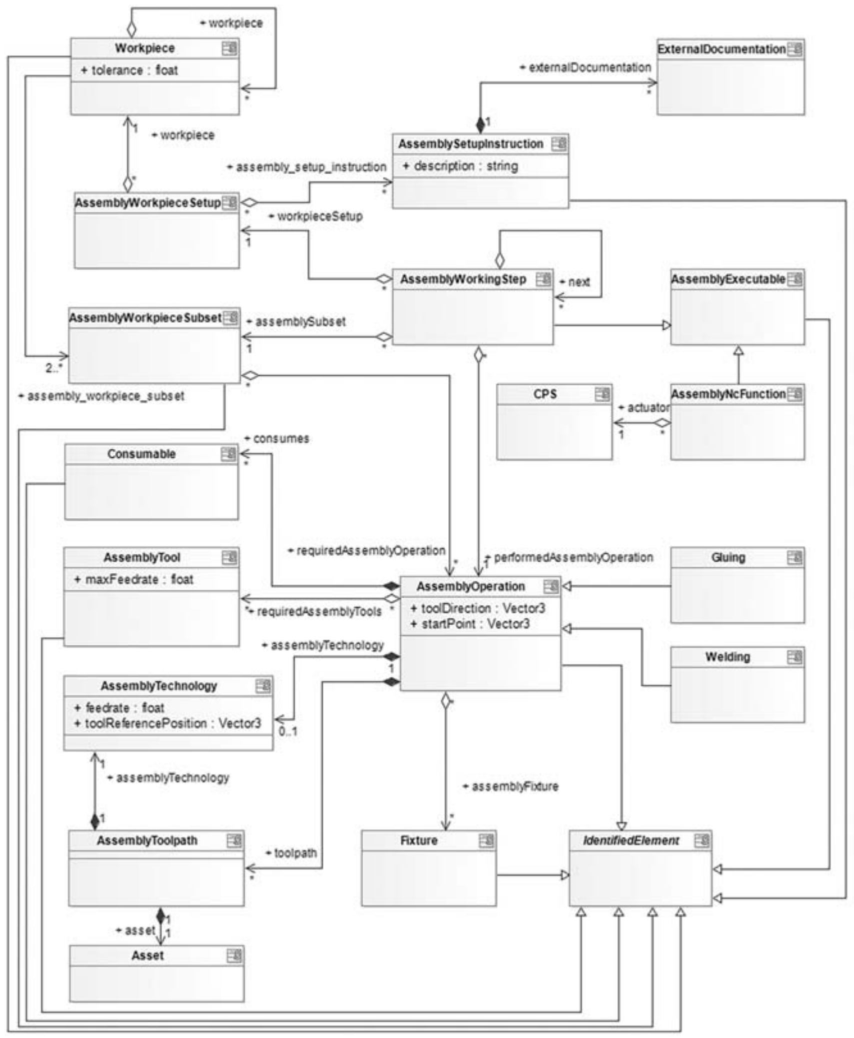

Figure 10.14 Assembly-Executable representation.

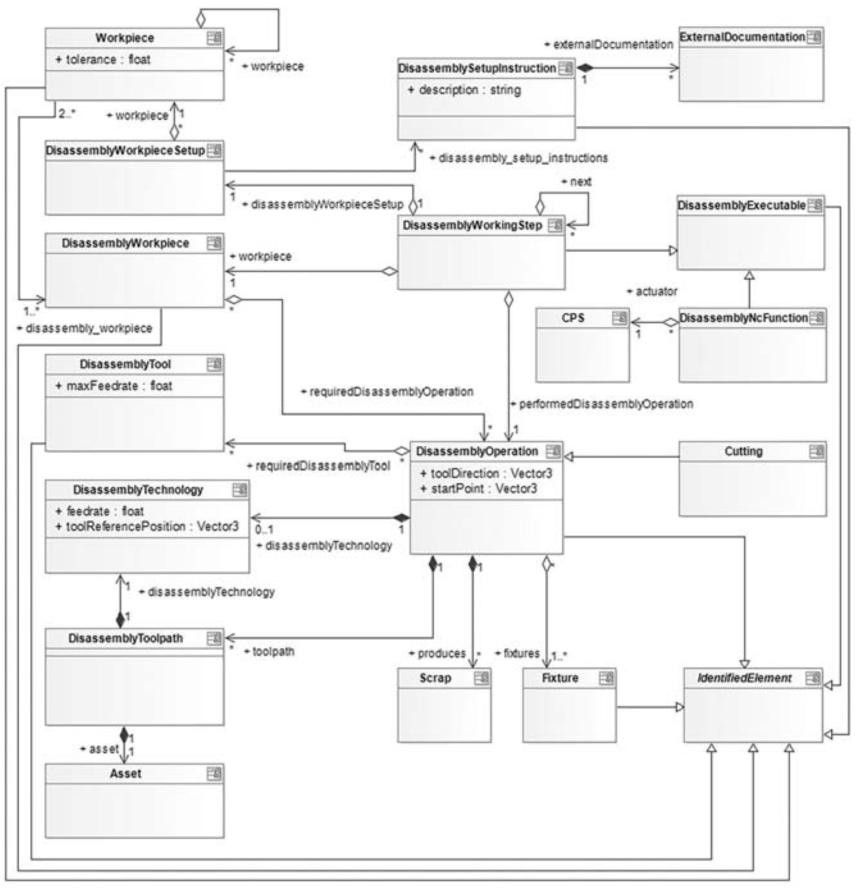

Figure 10.15 Disassembly representation.

10.3.8 Security Model

The phases of requirement gathering and analysis highlighted that security and privacy are two of the principal issues that must be properly addressed in a simulation platform.

Here, security and privacy will be enforced focusing mainly on the following aspects:

- The implementation of suitable **authentication/authorization** mechanisms
- **Securing communication and data storage** via encryption

These aspects fall under the so-called Privacy-Enhancing Technologies (PETs).

More in detail, authentication is the process of confirming the identity of an external actor in order to avoid possible malicious accesses to the system resources and services. Authentication, however, is only one side of the coin, it is in fact tightly coupled with the concept of authorization, which can be defined as the set of actions a software system has to implement in order to grant (authenticated) users the permission to execute an operation on one or more resources. Authentication and authorization are concepts related to both security (unwanted possible catastrophic access to inner resources) and privacy and data protection issues (malicious access to other users' data).

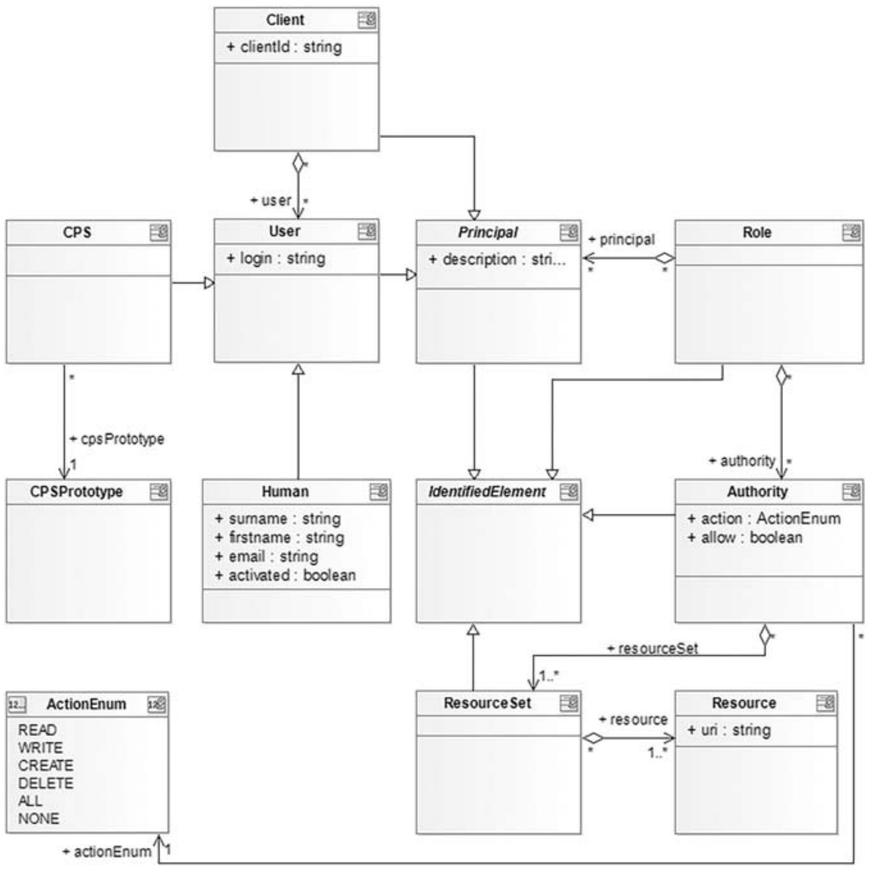

Figure 10.16 Class diagram for the security section of the *Meta Data Model*.

Securing communication is the third piece of this security and privacy puzzle, and it is as necessary as authentication and authorization. As a matter of fact, most physical devices (e.g. wireless networks) show very few privacy guaranties, and in many cases, it is practically impossible to secure wide networks against eavesdroppers. Nonetheless, confidentiality and privacy are fundamental rights (acknowledged by the European Convention on Human Rights) and must be enforced over often unsecure (communication and storage) infrastructures. For this reason, the simulation platform is committed to employ state-of-the-art encryption mechanisms (e.g. SSL and TLS) on both data storage and transport.

In the following sections of the document, the part of *Meta Data Model* devoted to security/access control management is reported and discussed. The elements of the meta model that play a role in security-related scenarios are depicted in Figure 10.16.

10.4 Conclusions

Multidisciplinary simulation is increasingly important with regard to the design, deployment and management of CPS-based factories. There are many challenges arising when exploiting the full potential of simulation technologies within Smart Factories, where a consistent technological barrier is the lack of digital continuity. Indeed, this chapter targets the fundamental issue of the lack of common modelling languages and rigorous semantics for describing interactions – physical and digital – across heterogeneous tools and systems towards effective simulation applicable along the whole factory life cycle.

The data model described in this chapter is the result of the joint effort of different actors from the European academia and industry. From the reference specifications presented in this chapter, which should be considered as a first release of a broader collaboration, a model has indeed been developed and has subsequently been validated within both an automotive industry use case and a steel carpentry scenario.

Acknowledgements

This work was achieved within the EU-H2020 project MAYA, which received funding from the European Union's Horizon 2020 research and innovation programme, under grant agreement No. 678556.

References

[1] www.automationml.org, accessed on March 24, 2017.

[2] Weyer, Stephan, et al.: Towards Industry 4.0-Standardization as the crucial challenge for highly modular, multi-vendor production systems. IFAC-PapersOnLine, 48. Jg., Nr. 3, S. 579–584, 2015.

[3] Baudisch, Thomas and Brandstetter, Veronika and Wehrstedt, Jan Christoph and Wei{\ss}, Mario and Meyer, Torben: Ein zentrales, multiperspektivisches Datenmodell fur die automatische Generierung von Simulationsmodellen fur die Virtuelle Inbetriebnahme. Tagungsband Automation 2017.