Disassembly Process Information Model for Remanufacturing

Shaw C. Feng[1], Hanmin Lee[2], Thomas Kramer[1], Che B. Joung[3], Parisa Ghodous[4], and Ram D. Sriram[1]

[1]National Institute of Standards and Technology
[2]Korea Institute of Machinery and Materials
[3]Korea Institute of Industrial Technology
[4]University of Claude Bernard Lyon I

Abstract
Disassembly is essential to dismantle a product for remanufacturing during maintenance or at the end of service life. The National Institute of Standards and Technology has developed an information model for describing disassembly processes. A disassembly process includes many subprocesses, such as separation, cleaning, repair, replacement, and inspection. This paper describes a disassembly process information model with the following key components: workpiece, material content, equipment, and workflow. The workflow aspect supports the modeling of operations, operation sequences, branching an operation into multiple ones, and joining multiple operations into one. The model provides a foundation for computer-aided disassembly software systems development.

Key words:
disassembly, disassembly modeling, disassembly process, information modeling, and remanufacturing.

1. Introduction

Manufacturing is one of the fundamental aspects of an industrialized society. Today many manufacturers are facing challenges of natural resource shortages [1, 2], industrial waste accumulations [3], and economic stagnation. Manufacturing companies are under pressure to cope with these problems and maintain competitiveness. The need for sustainability in manufacturing has been prominent as problems of resource depletion are worsening [1]. One solution is to close the loop on material flows from manufacturers to users and back to manufacturers. This solution reduces extraction of resources from the earth and energy use when producing stock materials [4]. The reuse and recycle of end-of-life products is key to reducing landfill, energy use, and greenhouse gas emission [5]. Furthermore, precious materials can be recovered from recycling end-of-service-life products. In order to restrict solid and toxic wastes released to the environment, regulations on waste management of electronic products have been promulgated by many countries. Specifically, regulations have been in place to curb the amount of toxic materials used in electronic products [6, 7]. These regulations have already had positive impact on the sustainability performance of the electronic industry. The European Commission mandates that 95 % in weight of end-of-use vehicles will have to be recycled or recovered by 2015 [8]. Hence, disassembly for reuse, recycle, and remanufacturing becomes important.

Efficient disassembly reduces remanufacturing costs and decreases the rate of resource depletion [9, 10]. Also, customers demand sustainability; therefore, industries are adopting means to

manufacture products in a cleaner and more socially responsible way [11]. Disassembly is key to achieving closed-loop material flow. Disassembly separates out-of-service products into reusable, recyclable, remanufacturable, and disposable parts. Currently, some manufacturing companies lack capability to design for disassembly and remanufacturing because a comprehensive description of disassembly for remanufacturing does not exist [12].

The degree of difficulty in dismantling a product at the end of its service life, i.e., disassemblability, is determined in the product design phase. Disassembly cost is also directly related to disassemblability. Design engineers, thus, need to describe disassembly and estimate the cost in product design. To help design engineers in determining disassemblability and estimate costs, information models must be available to describe disassembly processes. The information model is to represent the disassembly process plan, which includes the information of disassembly features, disassembly sequence, dismantling tasks, equipment requirements, and resource requirements. Figure 1 shows that design for disassembly data is shared among eco-Computer-Aided Design (ecoCAD) systems and disassembly process planning systems. Also, disassembly plans need to be exchanged among disassembly process planning systems. Hence, a common information model is also necessary to share and exchange disassembly information among lifecycle applications, such as design and disassembly planning.

(Figure 1 goes here.)

This paper describes a disassembly model, using the Unified Modeling Language (UML) [13]. Section 2 reviews disassembly-related literature. Section 3 describes all the classes and their relationships in the developed information model. Models of disassembly features and tolerance information can be found in an earlier National Institute of Standards and Technology (NIST) report [14]. Section 4 provides two case studies of a car suspension module and a cellular phone disassemblies. Section 5 concludes the paper with possible future directions.

2. Review of Disassembly-related Literature

Publications are available on disassembly representation in design, disassembly process planning, and cost estimation. This section provides a literature review in three parts: design for disassembly, modeling strategies, and existing models. Gaps in an integrated disassembly information model are identified at the end of the section.

2.1 Design for disassembly

Research results exist for product designers to increase disassemblability in products. Tang et al. [15] developed disassemblability analysis of a product and modeled disassembly sequence. Lee et al. [16] demonstrated the search of disassembly paths of an assembly. Desai et al. [17, 18] analyzed disassemblability during product design so that the product could be quickly and cost effectively disassembled at the end of service life. Algorithms have been developed to search for the optimal number of components of a product that should be disassembled to minimize the cost and maximize benefits. For example, Ijomah et al. [19, 20] extended design analysis for disassemblability and remanufacturability, reusability, and recyclability.

2.2 Modeling strategies for disassembly process planning

In a product disassembly system, choosing the representation of disassembly sequences is an important decision not only in creating a disassembly sequence planner but also in designing an intelligent controller for a disassembly process. Over the past decade, many modeling strategies have been proposed, i.e. AND/OR graph, disassembly Petri net (DPN), and Component-Fastener Graph. The AND/OR graph is the most popular and forms the basis for much of the later work. The nodes and the hyperarcs in these AND/OR graphs, respectively, correspond to subassemblies and disassembly tasks in which a more complex subassembly is separated into two or more subassemblies. Homem et al. [21] presented an AND/OR graph approach to model all the possible disassembly sequences of a system. The postulated condition is that it is possible to derive the assembly sequence by reversing the disassembly sequence obtained. For each possible assembly sequence generated, a certain set of assembly weights is assigned so that the optimal sequence could be obtained. Lambert et al. [22, 23, 24] used a disassembly graph, which was based on the AND/OR graph and liaison analysis, to derive optimal disassembly sequences.

Petri nets are useful to represent assembly sequences. Moore et al. [25] applied the disassembly Petri net approach for including complex AND/OR relationships in disassembly diagrams. Petri nets are frequently used in adaptive disassembly planners. The generation of an optimal disassembly sequence for devices with a probabilistic condition and adaptation based on additional observations has been a principal challenge for many years. Zussman et al. [26] developed adaptive planners and some associated experimental results. Tang et al. [27] presented an adaptive planner, based on product Petri nets and workstation Petri nets, which modifies the disassembly sequence according to the condition of the items in a batch.

Graph theory is suitable to represent part relationships in an assembly. Kuo et al. [28] proposed a non-directed graph-based heuristic approach for the generation of the disassembly sequence for recycling. A product is modeled by a component-fastener graph. By identifying the “cut-vertices,” the search splits the graph into subgraphs until a disassembly tree is formed. Based on the disassembly tree, disassembly sequences can then be generated. Li et al. [29, 30] proposed a Disassembly Constraint Graph (DCG), where all the possible disassembly operations needed for the maintenance of certain components or subassemblies can be deduced.

2.3 Disassembly-related Information Models

The literature lacks a standard information model of disassembly. This section reviews available assembly information models for the purpose of creating a disassembly information model. Some standard-based approaches and frameworks for assembly are also reviewed.

ISO 10303-44 concerning product structure configuration [31] provides limited assembly design representation that captures the assembly structure and kinematic joints during the design process. The assembly model establishes a neutral representation of assemblies of products. In this model, complete products are called “assemblies,” and the components of the lowest levels in the assemblies are called “parts.” The model focuses on the hierarchy of the product and on the position and orientation between parts.

The ISO working group TC 184/SC4/WG12 has proposed several enhancements to ISO 10303-44 assembly representation [32]. The proposal introduces the detailed geometric information not only for hierarchical relationships but also for peer-to-peer relationships among component parts via an assembly feature. Geometric constraints among component parts at the detailed geometric element level are also enabled. Furthermore, the proposal introduces more information on component association and includes detailed information about appropriate assembly features involved in component association.

The Open Assembly Model (OAM) [33] provides a standard representation and exchange protocol for assembly and system-level tolerance. The OAM is extensible. It currently provides tolerance representation and propagation, representation of kinematics, and engineering analysis at the system level. The assembly information model emphasizes the nature and information requirements for part features and assembly relationships. The model includes both assembly as a concept and assembly as a data structure. For the latter it uses the data structures of ISO 10303.

2.4 High-level requirements for an information model

Based on the literature review, the following gaps are identified for modeling the information of disassembly process plans:

- A common disassembly task sequence representation [34].
- Disassembly equipment and methods that are associated with the serial and parallel tasks.
- Task decomposition in disassembly processes.
- Cleaning and inspection processes following the separation of out-of-service products.

3. Disassembly Information Model

This section describes all the major classes and their relationships in the disassembly information model. The model has three major packages (i.e., modules) in UML. These packages support processes and operations of separation, cleaning, and inspection in a disassembly process. They are the Workpiece package, the Equipment package, and the Workflow package.

3.1 Workpiece Package

A workpiece is the whole or a part of an out-of-service product that is being disassembled. An out-of-service product to be remanufactured is also referred to as a "core." Class Workpiece is used in a disassembly project. Class Project will be described in Section 3.3 – Workflow. This subsection describes all the classes relevant to workpieces to be disassembled in the WorkpiecePack package. Figure 2 is a diagram of classes in the WorkpiecePack package.

(Figure 2 goes here.)

Class *PartCharacteristic*[1] represents a characteristic of a workpiece. It is an abstract data type. The class has no attributes.

Enumeration type SurfaceRoughnessType is used to list types of surface roughness measurements. All the definitions on surface finish roughness can be found in ISO 1302 [35]. This enumeration type includes R_a (the arithmetic average of a set of absolute measured values) and R_{rms} (the root mean square of a set of measured values). The list can be extended when it is necessary.

Class SurfaceRoughness represents the surface roughness of the feature of a workpiece that is being processed. It is a subtype of *PartCharacteristic*. The class has two attributes. Attribute *roughnessType*[2] represents the type of the surface roughness measurements. The attribute's data type is SurfaceRoughnessType. Attribute *value* represents the value of the surface roughness, and the attribute's data type is MeasureWithUnit[3].

Class Reflectivity represents the reflectivity of the surface of a workpiece that is being processed. It is a subtype of *PartCharacteristic*. The class has one attribute. Attribute *value* represents the value of the surface reflectivity, and the attribute's data type is MeasureWithUnit.

Class PropertyParameter represents the parameter of a workpiece material property. The class has three attributes. Attribute *description* represents the description of the material property parameter, and the attribute's data type is String. Attribute *name* represents the name of the material property parameter, and the attribute's data type is String. Attribute *parameter* represents the parameter of the material property, and the attribute's data type is MeasureWithUnit.

Class Material represents the material of a workpiece. The class has three attributes. Attribute *materialID* represents the identification of the material, and the attribute's data type is Identification [14]. Attribute *properties* represents the material properties, and the attribute's data type is a set of PropertyParameter. Attribute *standardMaterialID* represents the standard material identification, and the attribute's data type is Identification[4].

Class Workpiece represents a workpiece. The class has six attributes. Attribute *characteristics* represents characteristics of the workpiece, and the attribute's data type is a set of *PartCharacteristic*. Attribute *clampingPositions* represents a set of clamping positions of the workpiece on a machine tool, and the attribute's data type is a set of Coordinates3D [14]. The clamping positions are in the part coordinate system. Attribute *globalTolerance* represents the default tolerance that is used in dimensions of the workpiece, and the attribute's data type is RangeOfDeviation [14]. Attribute *id* represents the identification of the workpiece, and the attribute's data type is Identification. Attribute *workpieceMaterial* represents the material of the workpiece, and the attribute's data type is Material. Attribute

[1] Note that the font style of an abstract class name is italic and that the first letter of the abstract class name is capitalized.
[2] Note that the font style of an attribute name is italic and that the first letter of the attribute name is in lower case.
[3] MeasureWithUnit is defined in NIST Interagency Report (NISTIR) 7772 [14].
[4] Identification is defined in NISTIR 7772 [14].

features represents a set of disassembly features in the workpiece, and the attribute's data type is a set of DisassemblyFeature [14].

3.2 Equipment Package

The EquipmentPack package contains all the classes relevant to describing equipment that is used to perform disassembly tasks. The package has one class and three subpackages. Figure 3 shows the diagram of classes in the EquipmentPack package.

(Figure 3 goes here.)

Class Equipment represents a piece of equipment that is used in disassembly. The class has three attributes. Attribute *id* represents the identification of a piece of equipment, and its data type is Identification. Attribute *location* represents the location of the piece of equipment in a factory, and the attribute's data type is String. Attribute *responsiblePersonnel* represents the name of the person who is responsible for the equipment, and the attribute's data type is String.

Three subpackages are SeparationEquipmentPack, CleaningPack, and DMEPack. They are described in the following subsections.

3.2.1 Separation Equipment Package

The SeparationEquipmentPack package contains all the classes relevant to separation equipment. The package includes classes that represent a piece of equipment used in dismantling an assembly into subassemblies or individual parts. Figure 4 shows the diagram of classes in the package.

(Figure 4 goes here.)

Class SeparationEquipment represents a piece of equipment used in separating an assembly into subassemblies or individual parts for reuse, recycle, or remanufacturing. The class is a subtype of Equipment and has no additional attribute.

Enumeration type SeparationToolType is used to list types of tools used in separation. This enumeration type includes *ScrewDriver*, *HandDrill*, and *Knife*. The list can be extended when it is necessary.

Class SeparationTool represents a tool used in separating an assembly for reuse, recycle, or remanufacturing. The class is a subtype of SeparationEquipment and has one attribute. Attribute *toolType* represents the type of a separation tool, and the attribute's data type is SeparationToolType.

Enumeration type SeparationWorkstationType is used to list types of workstation used in separation. This enumeration type includes *DestructiveDisassemblyWorkstation*,

NondestructiveDisassemblyWorkstation, *ToolChangingSystem*, *MaterialHandlingSystem*, and *CryotechnicalWorkstation*. The list can be extended when it is necessary.

Class SeparationWorkstation represents the workstation used in separating an assembly into subassemblies or individual parts for reuse, recycle, or remanufacturing. The class is a subtype of SeparationEquipment and has one attribute. Attribute *workstationType* represents the type of a separation workstation, and the attribute's data type is SeparationWorkstationType.

3.2.2 Cleaning Equipment Package

The CleaningPack package contains all the classes relevant to cleaning equipment. It includes classes that represent a piece of equipment used in cleaning a subassembly or an individual part. Figure 5 is a diagram of classes in the package.

(Figure 5 goes here.)

Class CleaningEquipment represents a piece of equipment used in cleaning separated parts for reuse, recycle, or remanufacturing. The class is a subtype of Equipment and has no additional attribute.

Enumeration type CleaningWorkstationType is used to list types of workstation used in a cleaning process. This enumeration type includes *CompressedAirWorkstation*, *DryIceCleaningWorkstation*, *CO2-SnowWorkstation*, and *LaserCleaningWorkstation*. The list can be extended when it is necessary.

Enumeration type CleaningToolType is used to list types of tools used in cleaning. This enumeration type includes *CompressedAirSupply*. The list can be extended when it is necessary.

Class CleaningWorkstation represents a workstation used in cleaning separated parts. The class is a subtype of CleaningEquipment and has one attribute. Attribute *type* represents the type of a separation workstation, and the attribute's data type is CleaningWorkstationType.

Class CleaningTool represents a tool used in cleaning separated parts. The class is a subtype of CleaningEquipment and has one attribute. Attribute *type* represents the type of the cleaning tool, and the attribute's data type is CleaningToolType.

3.2.3 Dimensional Measurement Equipment Package (DMEPack)

The DMEPack contains all the classes relevant to dimensional measurement equipment. It includes classes that represent a piece of equipment used in measuring a disassembled part. Figure 6 is a diagram of classes in the package.

(Figure 6 goes here.)

Class DimensionalMeasurementEquipment represents a piece of equipment used in dimensional measurement of separated parts. The class is a subtype of Equipment and has no additional attributes.

Class *Sensor* represents a sensor used in dimensional inspection of separated parts. It is an abstract class. The class has one attribute. Attribute *extensionDescription* represents the description of the extension with which the sensor is mounted on a coordinate measurement machine, and the attribute's data type is String. If there is no extension, the description would be stated as "none."

Class LaserScanningSensor represents a laser sensor used in scanning separated parts. It is a subtype of *Sensor*. The class has one attribute. Attribute *incidentAngle* represents the incident angle of the laser beam relative to the workpiece coordinate system, and the attribute's data type is AngularMeasure.

Class TouchTriggeredProbe represents a touch-triggered probe used in probing separated parts. It is a subtype of *Sensor*. The class has seven attributes. Attribute *MPE-MF* represents the maximum permissible form error in the fixed multiple-stylus probing system, as defined in ISO 10360-1 [36], and the attribute's data type is optional LengthMeasure[5]. Attribute *MPE-MS* represents the maximum permissible size error in the fixed multiple-stylus probing system, as defined in ISO 10360-1, and the attribute's data type is optional LengthMeasure. Attribute *MPE-P* represents the maximum permissible probing error as defined in ISO 10360-1, and the attribute's data type is LengthMeasure. Attribute *numberOfStylus* represents the number of styli on the probe, and the attribute's data type is PositiveInteger[6]. Attribute *stylusLengths* represents the lengths of the styli, and the attribute's data type is an ordered set of LengthMeasure. Attribute *stylusOrientations* represents the orientations of the styli, and the attribute's data type is an ordered set of UnitVector3D[7]. Attribute *tipDiameters* represents the diameters of the styli, and the attribute's data type is an ordered set of LengthMeasure.

Class ContactScanningProbe represents a contact scanning probe used in scanning parts. It is a subtype of *Sensor*. The class has five attributes. Attribute *definedPath* represents whether the scanning path is predefined. The attribute's data type is ***boolean***[8]. Attribute *highPointDensity* represents whether the point density is high or low relative to a predefined density, and the attribute's data type is ***boolean***. Attribute *MPE-Tij* represents the maximum permissible scanning probe error as defined in ISO 10360-1, and the attribute's data type is optional LengthMeasure. Attribute *stylusLength* represents the length of the stylus of the scanning probe, and the attribute's data type is LengthMeasure. Attribute *tipDiameter* represents the diameter of the stylus, and the attribute's data type is LengthMeasure.

Class CoordinateMeasuringMachine represents a Coordinate Measuring Machine (CMM) used in measuring separated parts. It is a subtype of *DimensionalMeasurementEquipment*. The class has four attributes. Attribute *configuration* represents a machine configuration as defined in

[5] LengthMeasure is defined in NISTIR 7772 [13].
[6] PositiveInteger is defined in NISTIR 7772 [13].
[7] UnitVector3D is defined in NISTIR 7772 [13].
[8] A type in bold italic font denotes a UML defined type.

ANSI/ASME B89.1.12M [37]. The attribute's data type is optional String. Attribute *MPE-EL* represents the maximum permissible error of length measurement as defined in ISO 10360-2 [38], and the attribute's data type is optional LengthMeasure. Attribute *MPE-R0* represents the repeatability range of the maximum permissible error of length measurement as defined in ISO 10360-2, and the attribute's data type is optional LengthMeasure. Attribute *sensors* represents sensors loaded on a CMM, and the attribute's data type is a set of *Sensor*.

Class ArticulatingArmMeasuringMachine represents an articulating arm measuring machine. It is a subtype of CoordinateMeasuringMachine. The class has no additional attributes.

Class CoordinateMeasuringMachineWithRotaryTable represents a coordinate measuring machine with a rotary table. It is a subtype of CoordinateMeasuringMachine. The class has four additional attributes. Attribute *MPE-FA* represents the maximum permissible axial error of the rotary table as defined in ISO 10360-1, and the attribute's data type is optional AngularMeasure. Attribute *MPE-FR* represents the maximum permissible radial error of the rotary table as defined in ISO 10360-1, and the attribute's data type is optional LengthMeasure. Attribute *MPE-FT* represents the maximum permissible tangential error of the rotary table as defined in ISO 10360-1, and the attribute's data type is optional LengthMeasure. Attribute *rotaryTable* represents a description of the rotary table, and the attribute's data type is String.

Class *WorkpieceSensingSystem* represents a measuring system using sensing techniques. It is an abstract class and a subtype of *DimensionalMeasurementEquipment*. The class has no additional attributes.

Class OpticalGage represents an optical gage. It is a subtype of *WorkpieceSensingSystem*. The class has no additional attributes.

Class LaserTracker represents a laser tracker. It is a subtype of *WorkpieceSensingSystem*. The class has no additional attributes.

Class Theodolite represents a Theodolite machine. It is a subtype of *WorkpieceSensingSystem*. The class has no additional attributes.

Class VisionCheckingSystem represents a vision checking system. It is a subtype of *WorkpieceSensingSystem*. The class has no additional attributes.

Class PhotogrammetricalInstrument represents a photogrammetric instrument. It is a subtype of *WorkpieceSensingSystem*. The class has no additional attributes.

Class *MeasurementSoftware* represents a measurement software system. It is an abstract class and a subtype of *DimensionalMeasurementEquipment*. The class has no additional attributes.

Class MotionControl represents a motion control software system. It is a subtype of *MeasurementSoftware*. The class has no additional attributes.

Class Calibration represents a software system for calibration. It is a subtype of *MeasurementSoftware*. The class has no additional attributes.

Class Fitting represents a fitting software system. Examples of fitting include least-square fitting and minimax fitting. It is a subtype of *MeasurementSoftware*. The class has no additional attributes.

3.3 Workflow Package

The WorkflowPack package contains all the classes relevant to workflow. It includes classes and two subpackages that represent workflow. Figure 7 is a diagram of WorkflowPack.

(Figure 7 goes here.)

Class PostCondition represents the post condition of the completion of an operation. The post condition class describes when the succeeding operation should be started. The class has one attribute. Attribute *processElementID* represents the identification of the succeeding process element, and the attribute's data type is Identification.

Class ProcessElement represents an element in a process, including an operation, a decision node, a joint node, a while loop, or an end node. The class has three attributes. Attribute *elementID* represents the identification of the process element, and the attribute's data type is Identification. Attribute *subElements* represents child process elements, and the attribute's data type is an optional set of ProcessElement. Attribute *successor* represents the successor of the process element, and the attribute's data type is PostCondition.

Class WhenStarted represents a process element that occurs immediately when the process is started. (The start of this process can trigger the start of another process or other processes.) The class is a subtype of PostCondition and has one attribute. Attribute *successor* represents the succeeding process element, and the attribute's data type is ProcessElement.

Class WhenCompletion represents a process element that occurs when the completion of an operation. The class is a subtype of PostCondition and has one attribute. Attribute *successor* represents the succeeding process element, and the attribute's data type is ProcessElement.

Class DecisionNode represents a decision node in a process. The successor(s) will be determined, based on the result of a Boolean expression in the decision node. The class is a subtype of ProcessElement and has two attributes. Attribute *expression* represents the Boolean expression, and the attribute's data type is an ordered list of *BooleanExpression*. Classes of Boolean expressions will be described in the subsection below. Attribute *successors* represents the succeeding process elements, and the attribute's data type is an ordered list of ProcessElement. The successors have to correspond to the expressions in the first attribute. The result of an expression will start the corresponding successor in the second attribute.

Class JointNode represents a joint node in a process. The class is a subtype of ProcessElement and has two attributes. Attribute *predecessors* represents the preceding process elements, and the attribute's data type is a set of ProcessElement. Attribute *successor* represents the succeeding process element, and the attribute's data type is ProcessElement.

Class WhileLoop represents a while loop of process elements. The class is a subtype of ProcessElement and has two attributes. Attribute *expression* represents the Boolean expression of the termination of the while loop, and the attribute's data type is BooleanExpression. Attribute *operations* represents the operations within the while loop, and the attribute's data type is a list of Operation.

Class BranchNode represents a selection of operation to be performed at the end of the current operation, based on a predefined rule. The class is a subtype of ProcessElement and has two attributes. Attribute *expressions* represents a set of rules defined by multi-ary Boolean expressions, and the attribute's data type is a set of Multi-aryBooleanExpression. Attribute *selection* represents the selected operation that satisfied the rules, and the attribute's data type is Operation.

Class StartNode represents the start node of a process, such as a disassembly process or a subprocess. The class has three attributes. Attribute *actualStartDateTime* represents the actual date and time of the start of a process, and the attribute's data type is optional DateTime. Attribute *name* represents the name of the process, and the attribute's data type is String. Attribute *successor* represents the successor of the start node, and the attribute's data type is ProcessElement.

Class EndNode represents the end node of a process, such as a disassembly process or a subprocess. The class has one attributes. Attribute *actualStartDateTime* represents the actual date and time of the start of a process, and the attribute's data type is optional DateTime.

Class Process represents a process, consisting of a start node and process elements, including an end node. The class has six attributes. Attribute *name* represents the name of the process, and the attribute's data type is String. Attribute *performer* represents the performer of the process, and the attribute's data type is optional ContactInformation. Attribute *processID* represents the identification of the process, and the attribute's data type is Identification. Attribute *processStartingPoint* represents the start of the process, and the attribute's data type is optional StartNode. Attribute *subProcesses* represents child processes, and the attribute's data type is an optional list of ProcessElement. Attribute *alternatives* represents possible alternative processes, and the attribute's data type is an optional list of Process.

Class Setup represents a setup process. The class is a subtype of Process and has five attributes. Attribute *equipment* represents the equipment used in the setup, and the attribute's data type is Equipment. Attribute *instructions* represents instructions of setting up, and the attribute's data type is an optional list of String. Attribute *orientation* represents the orientation of the workpiece in the machine coordinate system, and the attribute's data type is UnitVector3D. Attribute *origin* represents the origin of the workpiece, and the attribute's data type is Coordinates3D. Attribute

securingPlane represents a plane on the workpiece that is used to secure or clamp down the workpiece on a separation, cleaning, or inspection machine, and the attribute's data type is Plane.

Class Project represents a project. The class has nine attributes. Attribute *dueDateTime* represents the project due date and time, and the attribute's data type is optional DateTime. Attribute *iD* represents the identification of the project, and the attribute's data type is Identification. Attribute *mainPlan* represents the main process plan of the project. The main process plan consists of one or more processes for disassembly. The attribute's data type is a list of Process. Attribute *name* represents the name of the project, and the attribute's data type is String. Attribute *plannedStartTime* represents the project starting date and time, and the attribute's data type is optional DateTime. Attribute *projectLead* represents the lead of the project, and the attribute's data type is ContactInformation. Attribute *status* represents the approval status of the project, and the attribute's data type is Approval.

The WorkflowPack includes two subpackages: BooleanPack and OperationPack. They are described in the following subsections.

3.3.1 Boolean Expression Package

The BooleanPack package contains all the classes relevant to Boolean expressions. The package includes classes that represent Boolean constants, Boolean variables, binary Boolean expressions, multi-ary Boolean expressions, and mathematical expressions. They are used in determining the workflow based on conditions and predefined rules. Figure 8 is a diagram of the BooleanPack subpackage.

(Figure 8 goes here.)

Class *BooleanExpression* represents a Boolean expression. The class is abstract and has no attributes.

Class *BooleanConstant* represents a constant used in a Boolean expression. The class is a subtype of *BooleanExpression* and has no attributes.

Class TrueConstant represents a Boolean constant of true. The class is a subtype of *BooleanConstant* and has no attributes.

Class FalseConstant represents a Boolean constant of false. The class is a subtype of *BooleanConstant* and has no attributes.

Class BooleanVariable represents a Boolean variable used in a Boolean expression. The class is a subtype of *BooleanExpression* and has two attributes. Attribute *name* represents the name of the variable, and the attribute's data type is String. Attribute *value* represents the value of the variable, and the attribute's data type is ***boolean***.

Class NotBooleanExpression represents a negation unary Boolean expression. The class is a subtype of *BooleanExpression* and has one attribute. Attribute *operand* represents the Boolean operand, and the attribute's data type is *BooleanExpression*.

Class *BinaryBooleanExpression* represents a binary Boolean expression. The class is an abstract class and a subtype of *BooleanExpression*. The class has two attributes. Attribute *operand1* represents a Boolean operand, and the attribute's data type is *BooleanExpression*. Attribute *operand2* represents the other Boolean operand, and the attribute's data type is *BooleanExpression*.

Class EqualBooleanComparisonExpression represents an evaluation of whether two operands are equal. The class is a subtype of *BinaryBooleanExpression*. The class has no additional attributes.

Class NotEqualBooleanComparisonExpression represents an evaluation whether two operands are not equal. The class is a subtype of *BinaryBooleanExpression*. The class has no additional attributes.

Class *Multi-aryBooleanExpression* represents a multi-ary Boolean expression. The class is an abstract class and a subtype of *BooleanExpression*. The class has one attribute. Attribute *operands* represents a set of two or more Boolean expressions, and the attribute's data type is a set of two or more *BooleanExpression*.

Class AndBooleanExpression represents a Boolean expression that evaluates to be true if all of its operands evaluate to true and evaluates to false otherwise. The class is a subtype of *Multi-aryBooleanExpression*. The class has no additional attributes.

Class OrBooleanExpression represents a Boolean expression that evaluates to false if all of its operands evaluate to false and evaluates to true otherwise. The class is a subtype of *Multi-aryBooleanExpression*. The class has no additional attributes.

Class XorBooleanExpression represents a Boolean expression that evaluates to true if exactly one of its operands evaluates to true and evaluates to false otherwise. The class is a subtype of *Multi-aryBooleanExpression*. The class has no additional attributes.

All the mathematical expressions are in the MathematicExpressionPack subpackage, which is the subpackage of BooleanPack. Figure 9 is a diagram of classes in MathematicExpressionPack.

(Figure 9 goes here.)

Class *MathematicalExpression* represents a mathematical expression. It is an abstract class.

Class MathematicalConstant represents a constant used in a mathematical expression. The class is a subtype of *MathematicalExpression*. The class has one attribute. Attribute *value* represents the value of the constant, and the attribute's data type is double.

Class MathematicalVariable represents a variable used in a mathematical expression. It is a subtype of *MathematicalExpression* and has two attributes. Attribute *name* represents the name of the variable, and the attribute's data type is String. Attribute *value* represents the value of the variable, and the attribute's data type is MeasureWithUnit.

Class UnaryFunction represents a mathematical function with only one operand. The class is a subtype of *MathematicalExpression*, and has one attribute. Attribute *operand* represents the operand used in the mathematical function, and the attribute's data type is *MathematicalExpression*.

Class Sine represents a sine function. The class is a subtype of UnaryFunction. The class has no additional attributes.

Class Cosine represents a cosine function. The class is a subtype of UnaryFunction. The class has no additional attributes.

Class Tangent represents a tangent function. The class is a subtype of UnaryFunction. The class has no additional attributes.

Class Arctangent represents an arctangent function. The class is a subtype of UnaryFunction. The class has no additional attributes.

Class Arcsine represents an arcsine function. The class is a subtype of UnaryFunction. The class has no additional attributes.

Class Arccosine represents an arccosine function. The class is a subtype of UnaryFunction. The class has no additional attributes.

Class Secant represents a secant function. The class is a subtype of UnaryFunction. The class has no additional attributes.

Class Cosecant represents a cosecant function. The class is a subtype of UnaryFunction. The class has no additional attributes.

Class Cotangent represents a cotangent function. The class is a subtype of UnaryFunction. The class has no additional attributes.

Class Logarithm represents a base 10 logarithm function. The class is a subtype of UnaryFunction. The class has no additional attributes.

Class NaturalLogarithm represents a natural logarithm function. The class is a subtype of UnaryFunction. The class has no additional attributes.

Class Absolute represents an absolute value function. The class is a subtype of UnaryFunction. The class has no additional attributes.

Class *BinaryMathematicalExpression* represents a binary mathematical expression with two operands. The class is a subtype of *MathematicalExpression* and an abstract class. The class has two attributes. Attribute *operand1* represents an operand used in the mathematical expression, and the attribute's data type is *MathematicalExpression*. Attribute *operand2* represents the other operand used in the mathematical expression, and the attribute's data type is *MathematicalExpression*.

Class Addition represents an addition operation. The class is a subtype of *BinaryMathematicalExpression*. The class has no additional attributes.

Class Subtraction represents a subtraction operation. The class is a subtype of *BinaryMathematicalExpression*. The attribute *operand2* is subtracted from the *operand1*. The class has no additional attributes.

Class Multiplication represents a multiplication operation. The class is a subtype of *BinaryMathematicalExpression*. The class has no additional attributes.

Class Division represents a division operation. The class is a subtype of *BinaryMathematicalExpression*. The attribute *operand1* is divided by the *operand2*. The class has no additional attributes.

Class Nth-Root represents an nth-root operation. The class is a subtype of *BinaryMathematicalExpression*. The attribute *operand1* is the base. The attribute *operand2* is the factor to perform the base in the nth-root operation. The class has no additional attributes.

Class Exponential represents an exponential operation. The class is a subtype of *BinaryMathematicalExpression*. The attribute *operand1* is the base. The attribute *operand2* is the exponent. The class has no additional attributes.

Class Modulus represents a modulus operation. The class is a subtype of *BinaryMathematicalExpression*. This operator returns the remainder when *operand1* is divided by *operand2*. The class has no additional attributes.

Class *MathematicalComparisonBooleanExpression* represents a mathematical comparison expression of the two operands. The class is a subtype of *BooleanExpression* and an abstract class. The class has two attributes. Attribute *operand1* represents an operand used in the mathematical comparison, and the attribute's data type is *MathematicalExpression*. Attribute *operand2* represents the other operand used in the mathematical comparison, and the attribute's data type is *MathematicalExpression*.

Class EqualTo represents an evaluation of whether *operand1* is equal to *operand2*. The class is a subtype of *MathematicalComparisonBooleanExpression*. The class has no additional attributes.

Class NotEqualTo represents an evaluation of whether *operand1* is not equal to *operand2*. The class is a subtype of *MathematicalComparisonBooleanExpression*. The class has no additional attributes.

Class LessThan represents an evaluation of whether *operand1* is less than *operand2*. The class is a subtype of *MathematicalComparisonBooleanExpression* . The class has no additional attributes.

Class LessThanOrEqualTo represents an evaluation of whether *operand1* is less than or equal to *operand2*. The class is a subtype of *MathematicalComparisonBooleanExpression*. The class has no additional attributes.

Class GreaterThan represents an evaluation of whether *operand1* is greater than *operand2*. The class is a subtype of *MathematicalComparisonBooleanExpression*. The class has no additional attributes.

Class GreaterThanOrEqualTo represents an evaluation of whether *operand1* is greater than or equal to *operand2*. The class is a subtype of *MathematicalComparisonBooleanExpression*. The class has no additional attributes.

3.3.2 Operation Package

The OperationPack package includes classes that represent operations in disassembly, such as separation, cleaning, and dimensional inspection. Figure 10 is a diagram of OperationPack, including a subpackage on the cleaning operation.

(Figure 10 goes here.)

Enumeration type OperationStateType represents the state of an operation. The states in the list are *active, suspended, stopped, and resumed*.

Class Operation represents an operation in a process. The class is a subtype of ProcessElement and has four attributes. Attribute *alternatives* represents possible alternative operations, and the attribute's data type is an optional list of Operation. Attribute *onFeature* represents the feature on which the operation is performed, and the attribute's data type is DisassemblyFeature. Attribute *performer* represents the performer of the operation, and the attribute's data type is ContactInformation. Attribute *postCon* represents the post condition of the operation, and the attribute's data type is PostCondition. Attribute *state* represents the state of the operation, and the attribute's data type is OperationStateTypes.

Class MeasurementOperation represents a dimensional measurement operation. The class is a subtype of Operation and has five attributes. Attribute *consumerRisks* represents a description of the consumer risks that are associated with the measurement results, and the attribute's data type is String[9]. Attribute *reportRequirements* represents the requirements of reporting measurement results, and the attribute's data type is a list of ReportingRequirement[10]. Attribute *samplingStrategy* represents any specified sampling strategy associated with the measurement operation, and the attribute's data type is optional SamplingStrategy[11]. Attribute *sensors*

[9] String is defined in NISTIR 7772 [14].
[10] ReportingRequirement is defined in NISTIR 7772 [14].
[11] SamplingStrategy is defined in NISTIR 7772 [14].

represents sensors used in dimensional measurement of a feature, and the attribute's data type is a set of *Sensor*. Attribute *toleranceToBeVerified* represents the tolerance to be verified, and the attribute's data type is Tolerance[12].

Class Separation represents a separation operation. The class is a subtype of Operation and has six attributes. Attribute *appliedTo* represents the assembly to which the separation operation is applied, and the attribute's data type is OAM:Assembly. OAM:Assembly is the assembly class of the Open Assembly Model [32]. Attribute *cost* represents the cost that is associated with the separation, and the attribute's data type is MeasureWithUnit. Attribute *name* represents the name of the operation, and the attribute's data type is String. Attribute *subassemblies* represents the separated subassemblies resulting from the separation, and the attribute's data type is a set of OAM:Assembly. Attribute *tool* represents the tool used in the separation operation, and the attribute's data type is SeparationTool. Attribute *workstation* represents the workstation used in the separation operation, and the attribute's data type is SeparationWorkstation.

Enumeration type MethodType represents the type of a separation operation. This enumeration type includes *Drilling*, *Unscrewing*, *Knife-cutting*, *Strike-cutting*, *Splitting*, *Shearing*, *Abrasive-cutting*, *WaterJetCutting*, *CryotechnicalSeparation*, *SnapOff*, *Shredding*, *Sorting*, *Unhooking*, *Sliding-off*, *Moving-apart*, and *Pressing-out*. The list can be extended when it is necessary.

Class Method represents the method used in a separation operation. The class has one attribute. Attribute *type* represents the type of method, and the attribute's data type is MethodType.

Class DestructiveSeparation represents a destructive separation operation. The class is a subtype of Separation and has one attribute. Attribute *destructiveMethods* represents methods used in a destructive separation, and the attribute's data type is a set of Method.

Class NonDestructiveSeparation represents a nondestructive separation operation. The class is a subtype of Separation and has one attribute. Attribute *nonDestructiveMethods* represents methods used in a nondestructive separation, and the attribute's data type is a set of Method.

3.3.3 Cleaning Operation Package

The CleaningOperationPack package includes classes that represent cleaning operations in disassembly. Figure 11 is a diagram of classes in the package.

(Figure 11 goes here.)

Enumeration type CleaningMethodType represents the type of a cleaning operation. This enumeration type includes *CompressedAirBlasting*, *DryIceBlasting*, *CO2-SnowBlasting*, and *Laser-PulseCleaning*. The list can be extended when it is necessary.

Class CleaningMethod represents the method used in a cleaning operation. The class has one attribute. Attribute *type* represents the type of method, and the attribute's data type is CleaningMethodType.

[12] Tolerance is defined in NISTIR 7772 [14].

Class CleaningOperation represents a cleaning operation. The class is a subtype of Operation and has four attributes. Attribute *method* represents a chosen cleaning method used in a cleaning operation, and the attribute's data type is a set of CleaningMethod. Attribute *part* represents the part or subassembly to which the cleaning operation is applied, and the attribute's data type is OAM:Assembly. Attribute *tool* represents the tool used in the cleaning operation, and the attribute's data type is CleaningTool. Attribute *workstation* represents the workstation used in the cleaning operation, and the attribute's data type is CleaningWorkstation.

4. Case studies

This section provides two case studies: a car suspension module and a flip-top cell phone. They illustrate the use of the disassembly information model.

4.1 Car Suspension module

The car suspension module can be composed of four parts (1-4) and two sub-assemblies (A and B), as shown in Figure 12. The sub-assembly A contains two parts (5 and 6), and the sub-assembly B contains five parts (7-11) and a sub-assembly (C), which further decomposes into five parts (12-16).

(Figure 12 goes here.)

Figure 13 has a diagram of the connection graph for the car suspension module. The nodes indicate the parts, and the edges indicate the connection relationships between two parts. The dotted rectangles indicate the ranges of sub-assemblies. A connection graph can be used as an input for a disassembly sequence planning system.

(Figure 13 goes here.)

Figure 14 shows the instance diagram of the disassembly information model, which contains the assembly hierarchy of the car suspension module. The instance diagram has a hierarchical relationship between assemblies and parts as well as connection relationships between parts.

(Figure 14 goes here.)

Figure 15 has an example of a more detailed connection relation between two parts. Parts 10 and 11 have two connections between them, namely one pin-hole connection and four bolt-nut connections. It is noted that the pin-hole connection is established just by two assembly features, while the bolt-nut connections are established by a connector applied on two assembly features. In this case, the pin-hole connections are established by putting the pin feature of part 10 into the hole feature of part 11, but the bolt-nut connection is established by applying the bolt-nut connector to the hole features of parts 10 and 11.

(Figure 15 goes here.)

A disassembly process planning system can use the connection graph as an input to generate a disassembly sequence, as shown in Figure 16. The car suspension module can be disassembled through three levels of disassembly sequences.

(Figure 16 goes here.)

According to the disassembly information model, a project is composed of several processes, each process has several sub-processes. For disassembly, a sub-process can be a non-destructive or destructive separation. A separation class has an assembly as input and two sub-assemblies as output, and it needs several operations to separate the sub-assemblies from the input assembly. Each operation class has a corresponding connection class of two sub-assemblies. Figure 17 shows an example of a disassembly project, which contains a disassembly process. The disassembly process is a sequence of four disassembly operations.

(Figure 17 goes here.)

Figure 18 shows relation between the instance diagrams of the disassembly information model representing disassembly sequences and connection graph. NDSeparation means non-destructive separation. It is an operation, and its class NonDestructiveSeparation is described in Section 3.3.2. The NDSeparation is an operation that separates the assembly into the subassembly consisting of parts 1, 2, 3, and 4 and the subassembly consisting of assembly A and B. There are three pin-hole connections between two subassemblies, so this NDSeparation has three pull tasks to disconnect the pin-hole connections.

(Figure 18 goes here.)

4.2 Flip-top cell phone

The flip-top cell phone is composed of thirteen parts, as shown in Figure 19. Figure 20 shows the connecting graph for the cell phone. The nodes indicate the parts and the edges indicate the connecting relationships between two parts. A connecting graph can be used as an input to a disassembly sequence planning system.

(Figure 19 goes here.)
(Figure 20 goes here.)

Figure 21 shows the instance diagram of the disassembly information model, which contains the assembly hierarchy of the cell phone. The instance diagram has connecting relationships among parts. For example, there is a screw connection between Parts 2 and 6.

(Figure 21 goes here.)

A disassembly process planning system can use the connecting graph as input to generate a disassembly sequence, as shown in Figure 22.

(Figure 22 goes here.)

Figure 23 shows the representation of a disassembly sequence using the disassembly information model. To separate the assembly into the subassembly (1,2,3,4,5,6) and the subassembly (7,8,9,10,11,12,13), the pin-hole connection between the subassemblies should be eliminated. The NDSeparation has a pull task to do this.

(Figure 23 goes here.)

5. Conclusion and Future Work

Manufacturing industries are facing the challenge of reusing and recycling products at the end of their service lives. Our literature review shows that the number of companies that embrace disassembly for remanufacturing is rapidly increasing. Reuse and recycling are critical activities to alleviate natural resource depletion and save energy to achieve the goal of sustainable development. Disassembly of end-of-service-life products is a key operation to separate the product into reusable and recyclable parts. Information on design for disassembly and the disassembly process is critical for decision making in design and manufacturing. An information model for disassembly processes is, hence, developed, using the Unified Modeling Language (UML). The model includes all the classes and their relationships on destructive disassembly, nondestructive disassembly, disassembly methods, equipment, and disassembly workflow. The model forms a basis for software development of design for disassembly and disassembly process planning systems.

Potential future work includes the following four areas:
(1) Comprehensive tests on the model with more complicated designs.
(2) Prototype disassembly databases, cost of disassembly analysis software, and disassembly process planning systems to be developed using the information model.
(3) A standard data exchange format of the UML model for design for disassembly and disassembly process plans.
(4) Cost estimation based on the model.

Acknowledgement

The authors thank Mr. Gwangsub Chang of Ajou Universiy for collaborating with us on the case studies.

References

1 Jovane, F., Yoshikawa, H., Alting, L., Boer, C., Westkamper, E., Williams, D., Tseng, M., Seliger, G., and Paci, A., “The incoming global technical and industrial revolution towards competitive sustainable manufacturing,” CIRP Annals – Manufacturing Technology, Vol. 57, 2008, pp. 641 - 659.
2 Brundtland, G. (1987), *Our Common Future*, World Commission on Environment and Development, Oxford University Press, United Kingdom.

3 Subramoniam, R., Huisingh, D., Chinnam, R., "Aftermarket remanufacturing strategic planning decision-making framework: theory & practice", Journal of Cleaner Production, Vol. 18, 2010, pp. 1575 – 1586.
4 Nasr, N. and Thurston, M., "Remanufacturing: A Key Enabler to Sustainable Product Systems," Proceedings of the 13th CIRP International Conference on Life Cycle Engineering, Leuven, Belgium, pp. 15 - 18, 2006.
5 Sriram, R., Navinchandra, D., and Allen, R., *Environmental Issues in Collaborative Design*, Mechanical Life Cycle Handbook: Good Environmental Design and Manufacturing, Hundal, M. (editor), Mercel Dekker , Inc, 2000.
6 Rifer, W., Brody-Heine, P., Peters, A., and Linnell, J., "Closing the Loop Electronics Design to Enhance Reuse/Recycling Value," the Green Electronics Council, Portland, Oregon, January 2009.
7 Jofre, S. and Morioka, T., "Waste Management of Electric and Electronic Equipment: Comparative Analysis of End-of-Life Strategies," Journal of Material Cycles and Waste Management, Volume 7, pp. 24 - 32, 2005.
8 Kumar, V. and Sutherland, J., "Sustainability of the Automotive Recycling Infrastructure: Review of Current Research and Identification of Future Challenges," International Journal of Sustainable Manufacturing, Volume 1, Nos. 1/2, pp. 145 - 167, 2008.
9 Bogue, R., "Design for Disassembly: A Critical Twenty-first Century Discipline," Assembly Automation, Volume 27, No. 4, pp. 285 - 289, 2007.
10 M'Saoubi, R., Outeiro, J., Chandrasenkaran, H., Dillon O., and Jawahir, I., "A review of surface integrity in machining and its impact on functional performance and life of machined products," International Journal of Sustainable Manufacturing, Volume 1, Nos. 1/2, pp. 203 - 236, 2008.
11 MIT Sloan Management Review and Boston Consulting Group (2011), "Sustainability: The 'Embracers' Seize Advantage," MIT Sloan Management Review Research Report, winter 2011.
12 Ilgin, M. and Gupta, S., "Environmentally conscious manufacturing and product recovery (ECMPRO): A review of the state of the art," Journal of Environmental Management, Vol. 91, 2010, pp. 563–591.
13 Rumbaugh, J., Jacobson, I., and Booch, G., The Unified Modeling Language Reference Manual, Addison Wesley, 1999.
14 Feng, S., Lee, H., Joung, C., Kramer, T., Ghodous, P., and Sriram, R., "Information Model for Disassembly for Reuse, Recycling, and Remanufacturing," NISTIR 7772, National Institute of Standards and Technology, Gaithersburg, Maryland, February 2011.
15 Tang, Y., Zhou, M. and Caudill, R., "An integrated approach to disassembly planning and demanufacturing operation," Proceedings of 2000 IEEE International Symposium on Electronics and the Environment, pp. 354 - 359, 2000.
16 Lee, K. and Gadh, R., "Computer Aided Design for Disassembly: A Destructive Approach," Proceedings of the 1996 IEEE International Symposium on Electronics and the Environment, pp. 173 - 178, May 1996.
17 Desai, A. and Mital, A., "Evaluation of Disassemblability to Enable Design for Disassembly in Mass Production," International Journal of Industrial Ergonomics, Vol. 32, pp. 265 – 281, 2003.

18 Desai, A. and Mital, A., “Incorporating Work Factors in Design for Disassembly in Product Design,” Journal of Manufacturing Technology Management, Vol. 16, No. 7, pp. 712 - 732, 2005.
19 Ijomah, W., McMahon, C., Hammond, G., and Newman, S., “Development of Design for Remanufacturing Guidelines to Support Sustainable Manufacturing,” Journal of Robotics and Computer-Integrated Manufacturing, Vol. 23, pp. 712 – 719, 2007.
20 Seliger, G., “Sustainability in manufacturing: recovery of resources in product and material cycles,” Springer, 2007.
21 Homem de Mello, L.S. and Sanderson, A. C., “AND/OR graph representation of assembly plans,” IEEE Transactions on Robotics and Automation, Vol.6, No.2, pp.188 - 100, 1990.
22 Lambert, A., “Optimal disassembly of complex products,” International Journal of Production Research, Vol.35, No.9, pp. 2509 - 2523, 1997.
23 Lambert, A., “Disassembly sequencing: a survey,” International Journal of Production Research, Vol.41, No.16, pp. 3721 - 3759, 2003.
24 Lambert, A. and Gupta, S., “Disassembly Modeling for Assembly, Maintenance, Reuse, and Recycling,” CRC Press, 2005.
25 Moore, K., Gungor, R. and Gupta, S., “Disassembly process planning using Petri nets,” Proceedings of 1998 IEEE Conference on Electronics and the Environment, pp. 88 - 93, 1998.
26 Zussman, E. and Zhou, M., “Design and implementation of an adaptive process planner for disassembly processes,” IEEE Transactions on Robotics and Automation, Vol.16, No.2, pp. 171 - 179, 2000.
27 Tang, Y., Zhou, M., Zussman, E., and Caudill, R., “Disassembly Modeling, Planning, and Application: A Review,” Proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Francisco, CA, April 2000, pp. 2197 – 2202.
28 Kuo, T., Zhang, H., and Huang, S., “Disassembly analysis for electromechanical products: a graph-based heuristic approach,” International Journal of production research, Vol.38, No.5, pp. 993 - 1007, 2000.
29 Li, J., Khoo, L.. and Tor, S., “A novel representation scheme for disassembly sequence planning,” International Journal of Manufacturing Technology, Vol.20, pp. 621- 630, 2002.
30 Li, J.R. Khoo, L,P. and Tor, S.B., “An object-oriented intelligent disassembly sequence planner for maintenance,” Computers in Industry, Vol.56, pp. 699 - 718, 2005.
31 ISO 10303-44, 1994, Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 44: Integrated Generic Resources: Product Structure.
32 Sugimura, N. and Ohtaka, A., "ISO TC 184/SC4/WG12 N597, JNC Proposal of STEP Assembly Model for Products (June 2000)," ISO, 2000.
33 Sudarsan, R., Han, Y., Feng, S., Roy, U., Wang, F., Sriram, R., and Lyons, K., “Object Oriented Representation of Electro-Mechanical Assemblies Using UML,” National Institute of Standards and Technology, NISTIR 7057, October 2003.
34 Vinodh, S., Nachiappan, N., and Kumar, R., "Sustainability Through Disassembly Modeling, Planning, and Leveling: a case study," Journal of Clean Technology and Environmental Policy, Published online by Springer-Verlag, 4 March 2011.
35 ISO 1302, 2002, Geometrical Product Specifications (GPS) - Indication of surface texture in technical product documentation.
36 ISO 10360-1, 2000, Geometrical Product Specifications – Acceptance and reverification tests for coordinate measuring machines, Part 1: Vocabulary.

37 ANSI/ASME B89.1.12M, Methods for Performance Evaluation of Coordinate Measuring Machines, The American Society of Mechanical Engineers, New York City, New York, 1985.
38 ISO 10360-2, 2009, Geometrical product specifications - Acceptance and reverification tests for coordinate measuring machines (CMM) - Part 2: CMMs used for measuring linear dimensions.

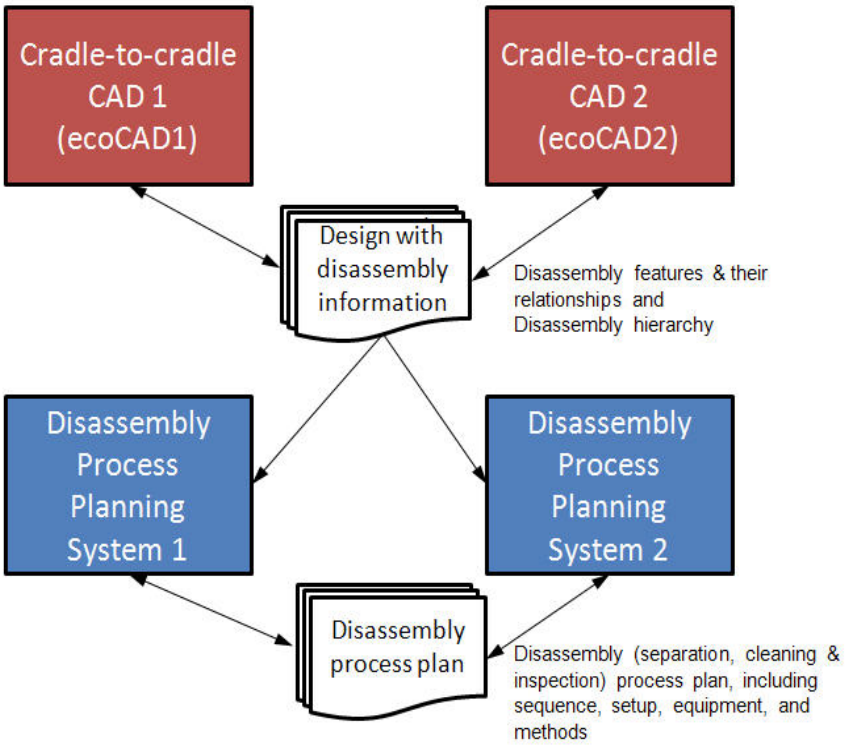

Figure 1. Disassembly information sharing

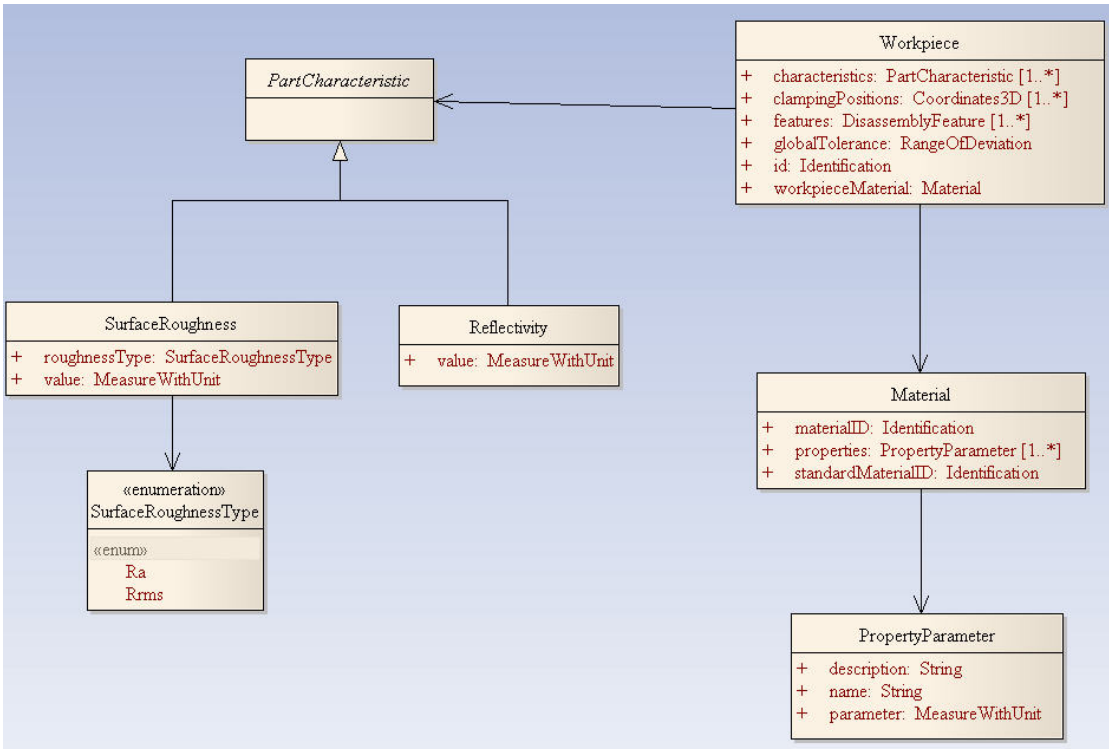

Figure 2. Class diagram of WorkpiecePack

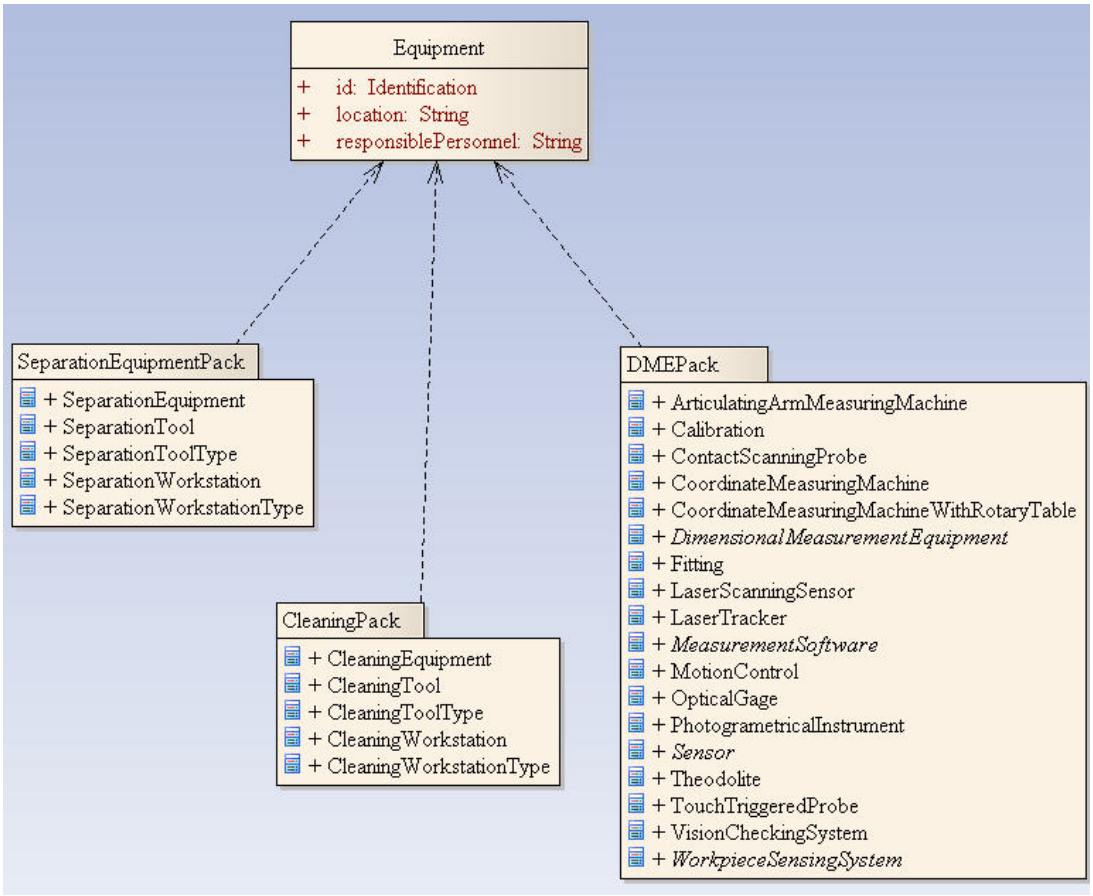

Figure 3. Class diagram of EquipmentPack

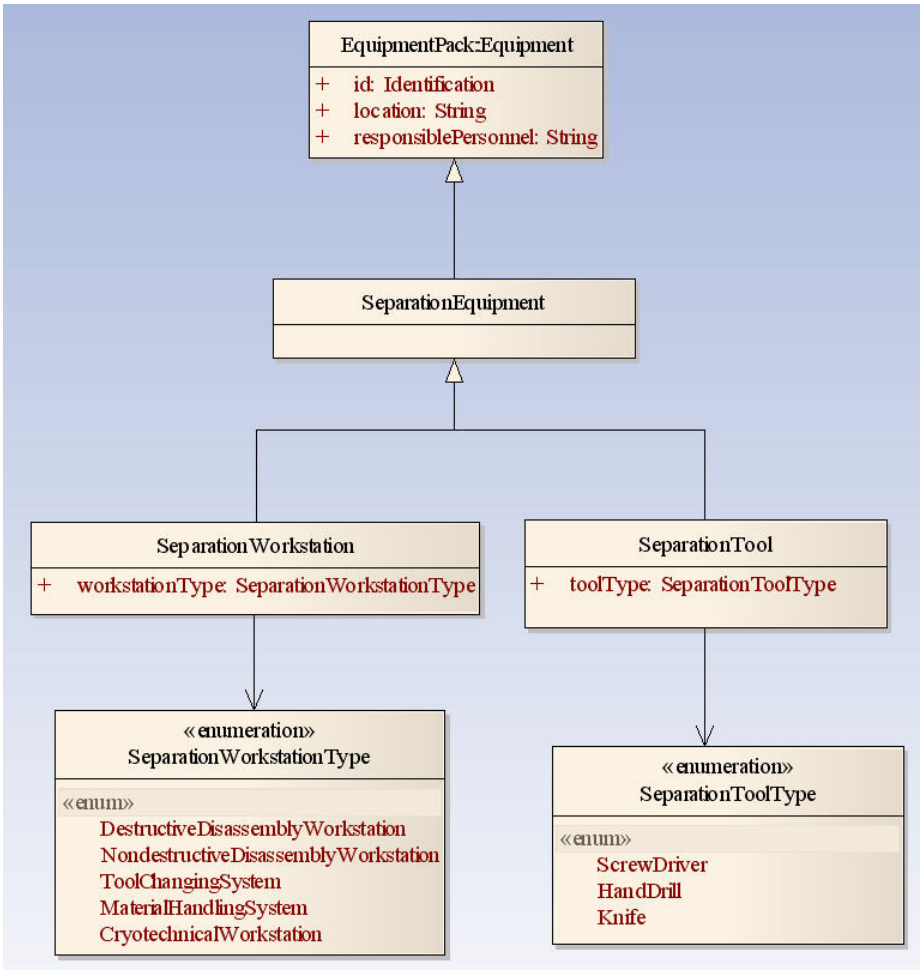

Figure 4. Class diagram of SeparationEquipmentPack

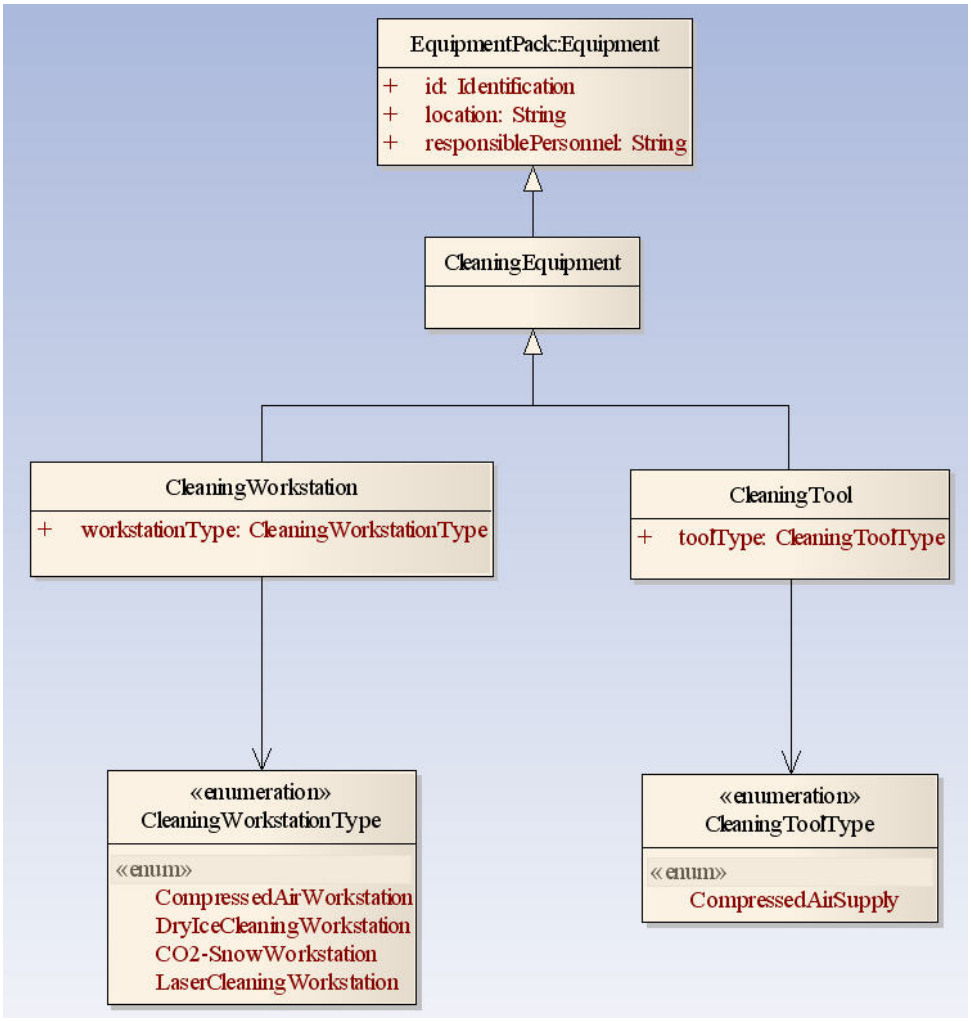

Figure 5. Class diagram of CleaningEquipmentPack

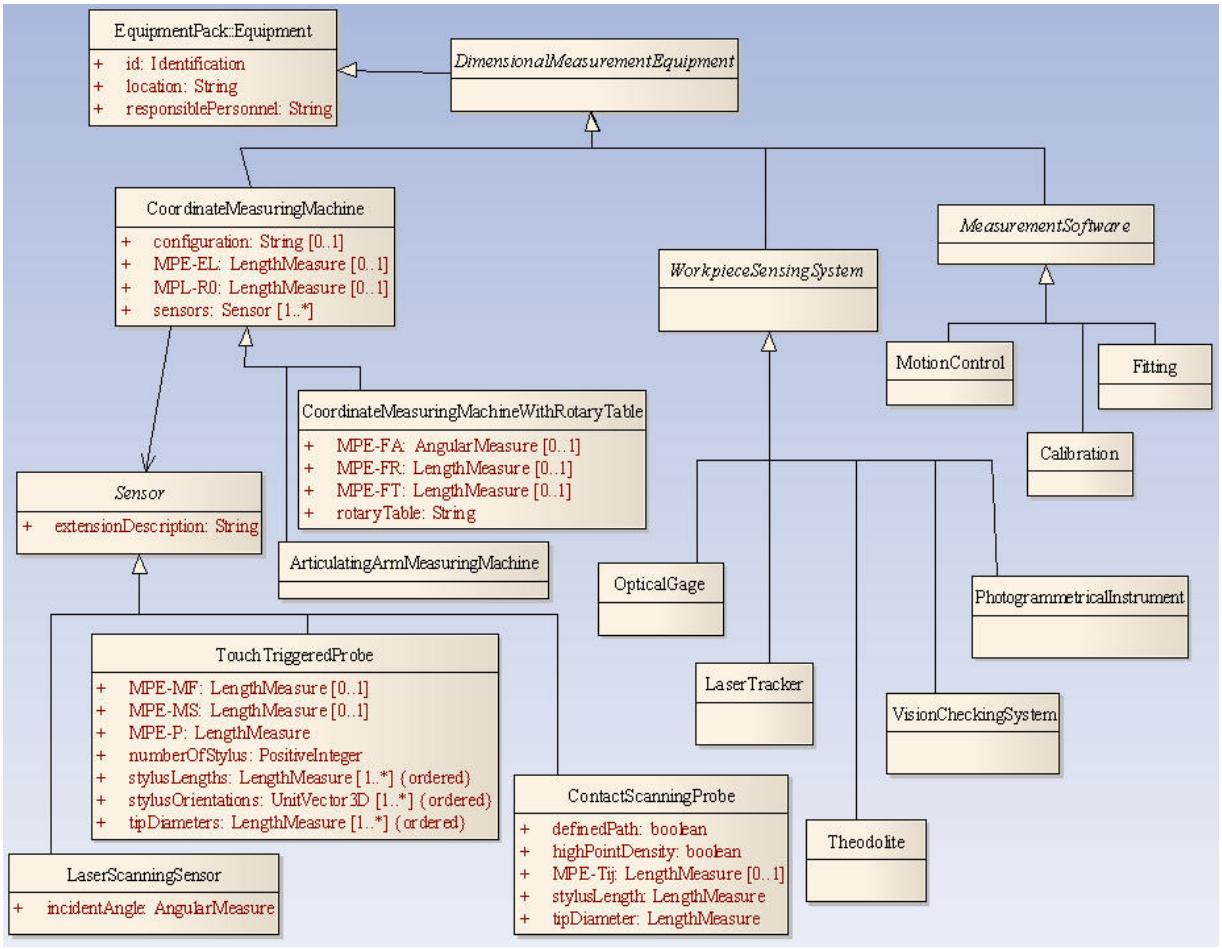

Figure 6. Class diagram of DMEPack

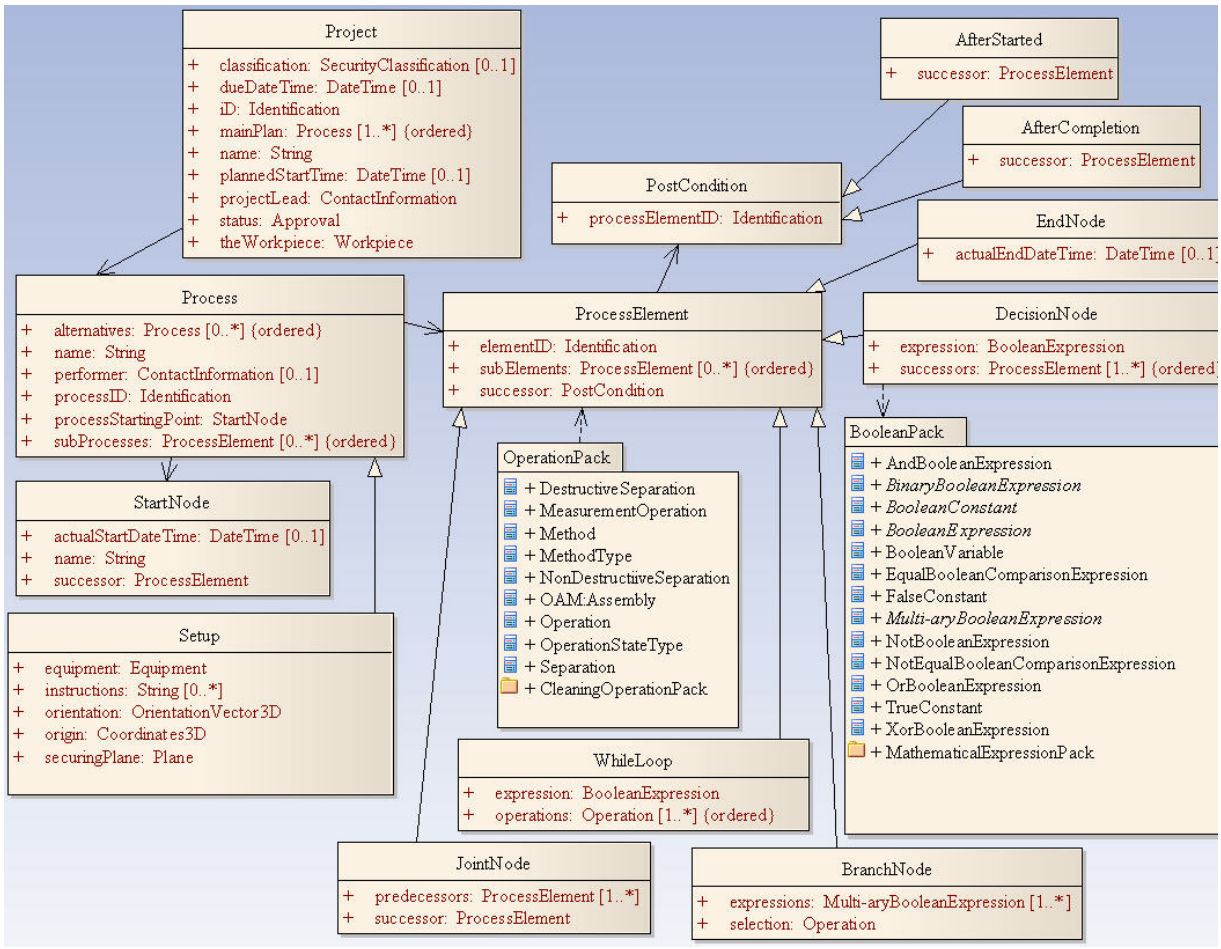

Figure 7. Class diagram of WorkflowPack

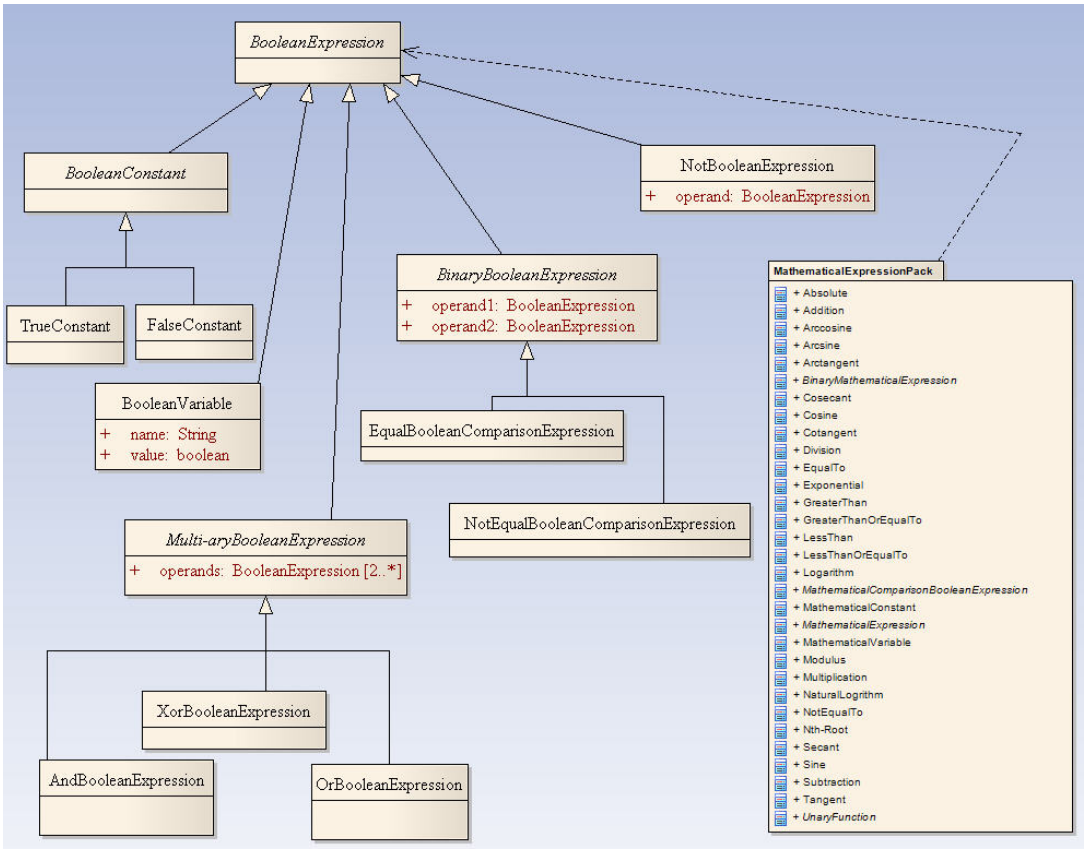

Figure 8. Class diagram of BooleanPack

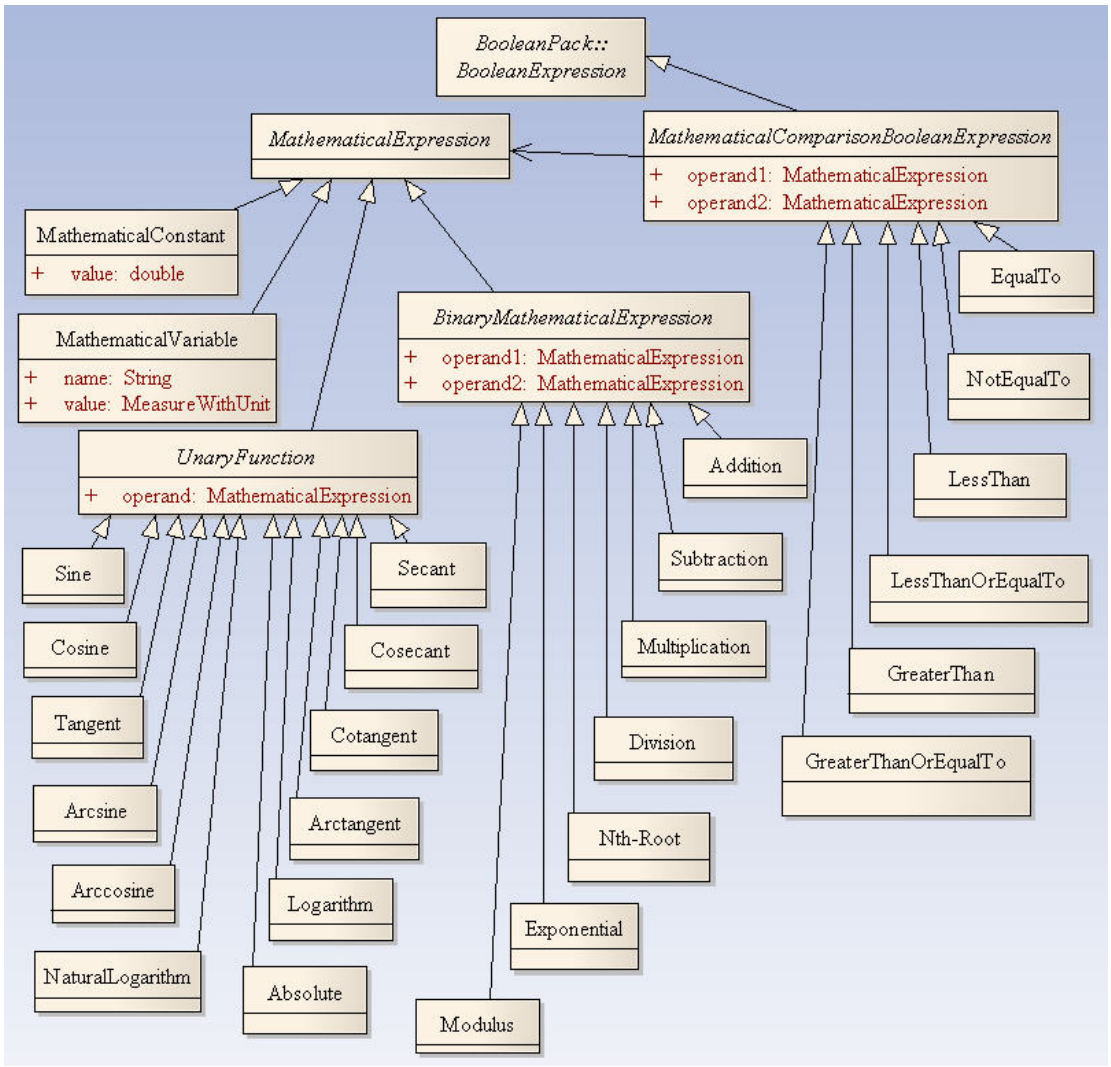

Figure 9. Class diagram of mathematical Comparison

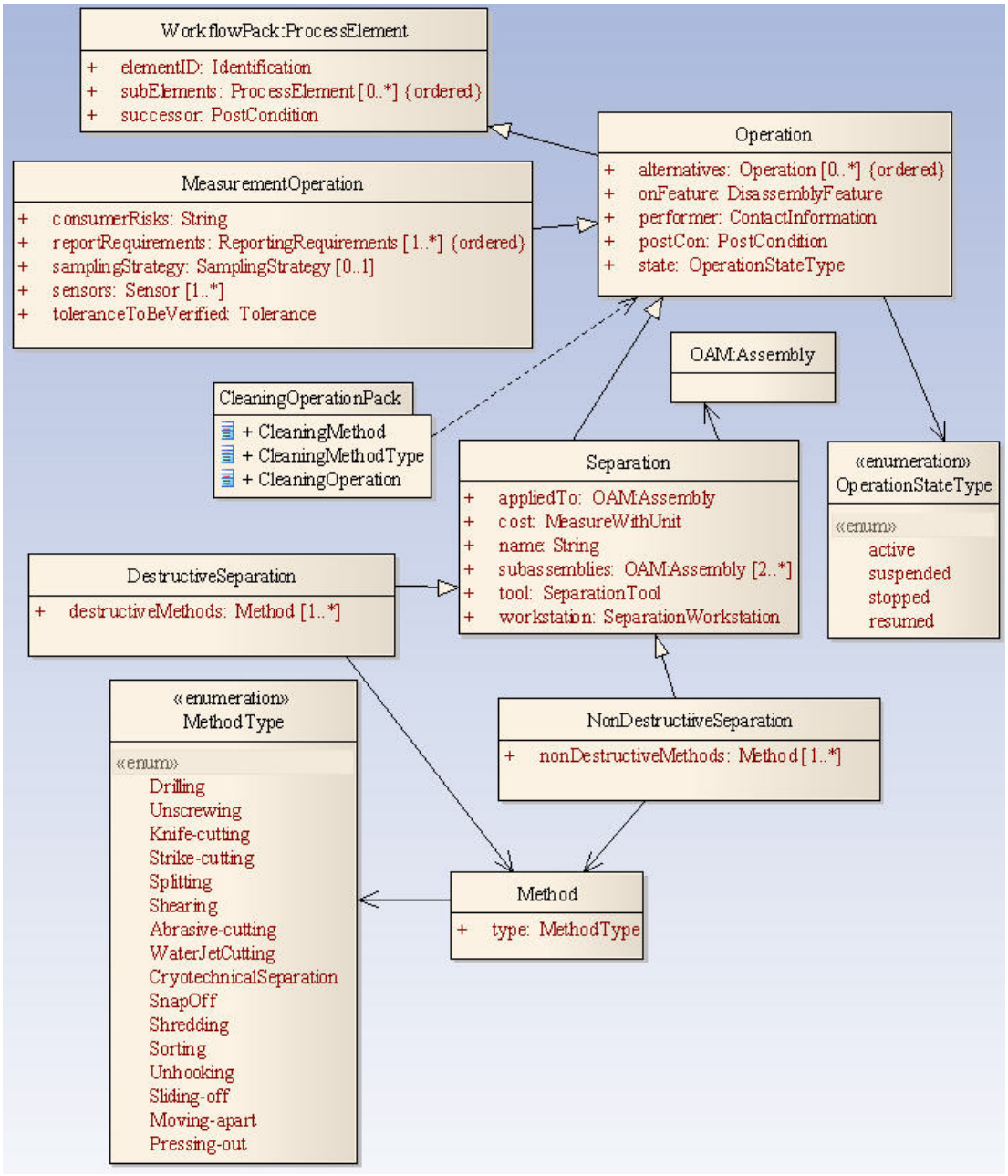

Figure 10. Class diagram of OperationPack

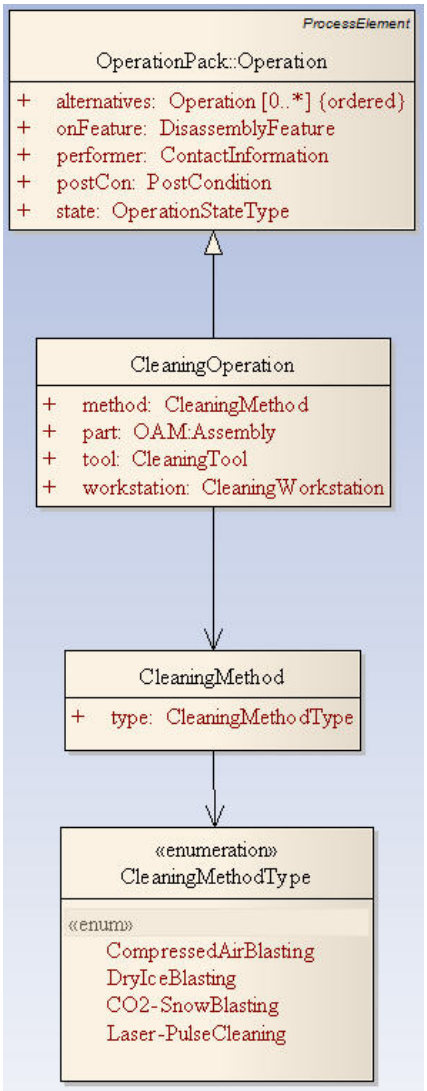

Figure 11. Class diagram of CleaningOperationPack

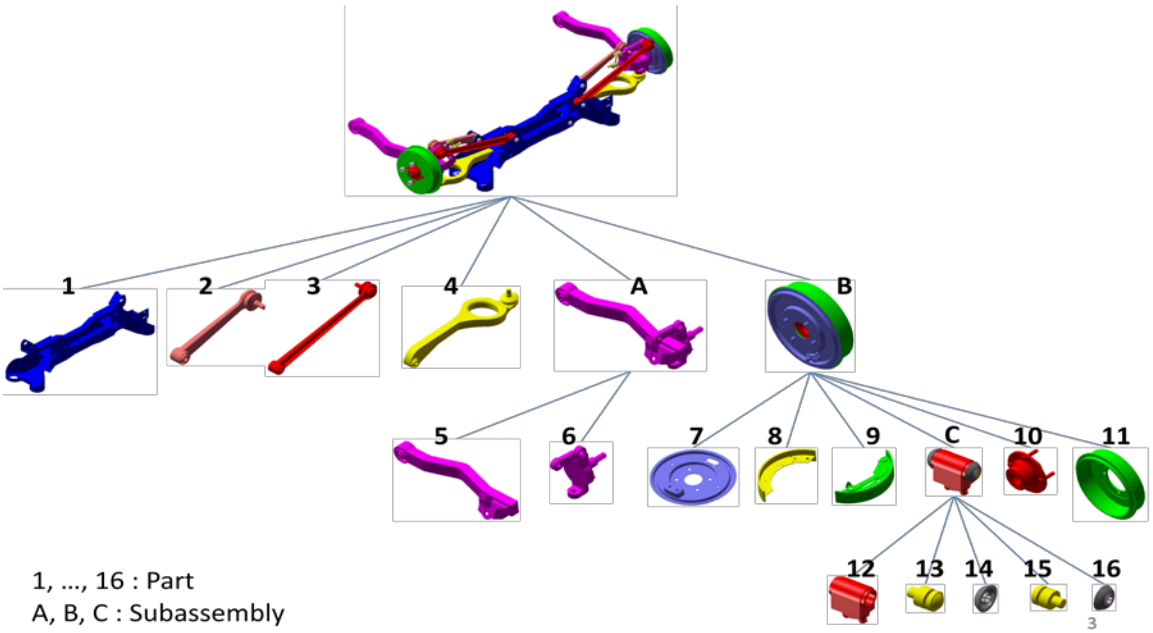

Figure 12. Assembly hierarchy of car suspension module

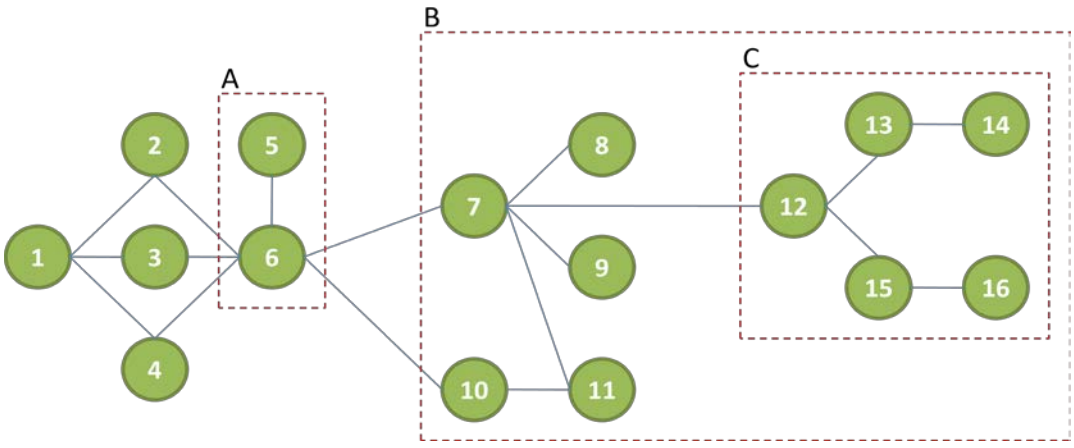

Figure 13. Connection graph

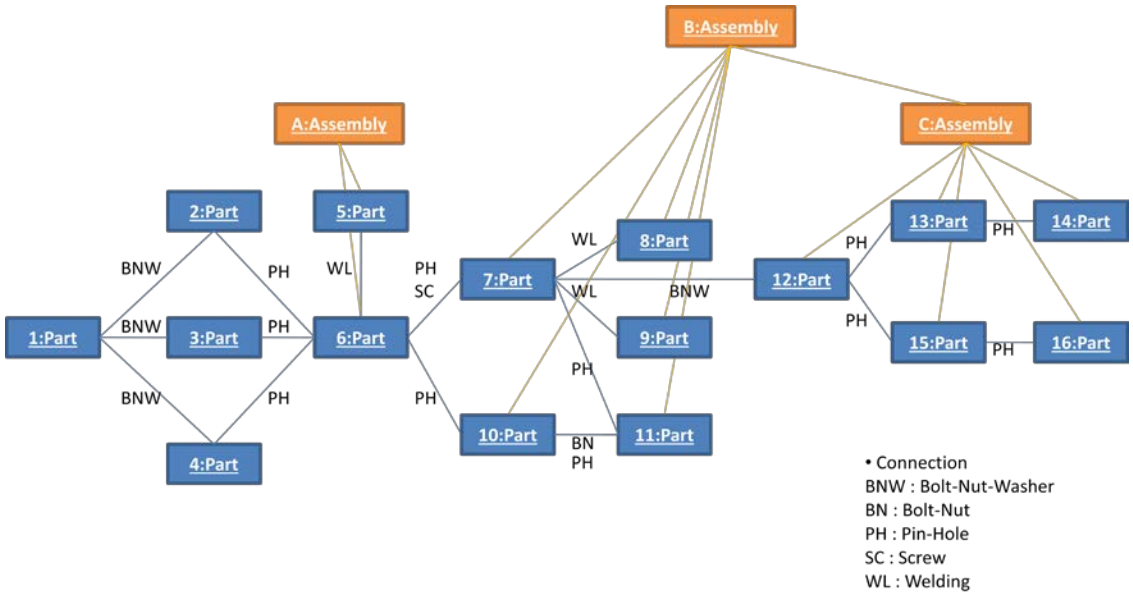

Figure 14. Instance diagram of the car suspension module assembly

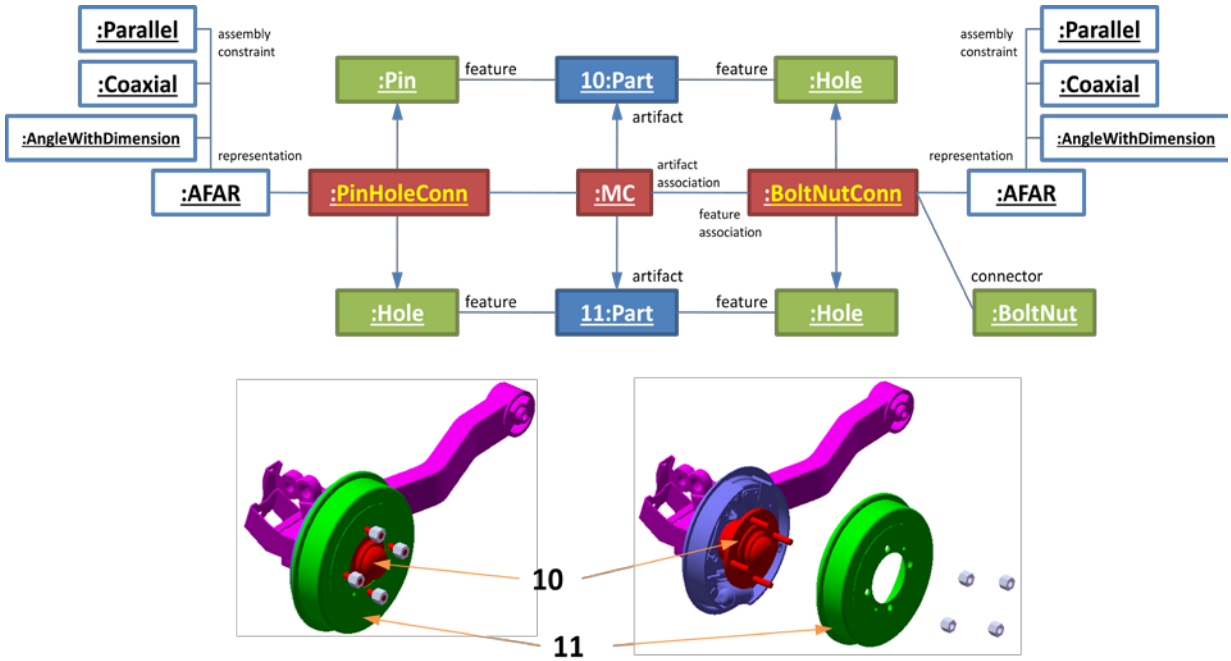

Figure 15. Instance diagram of a sub-assembly (detailed)

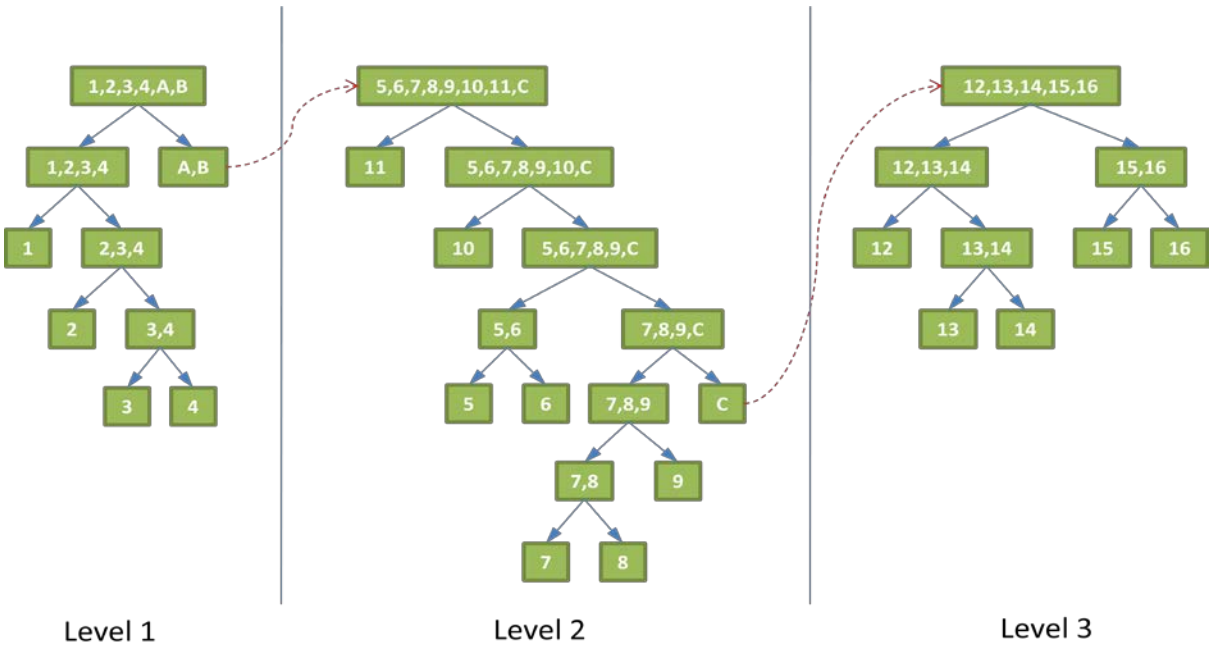

Figure 16. Disassembly sequence of the car suspension module

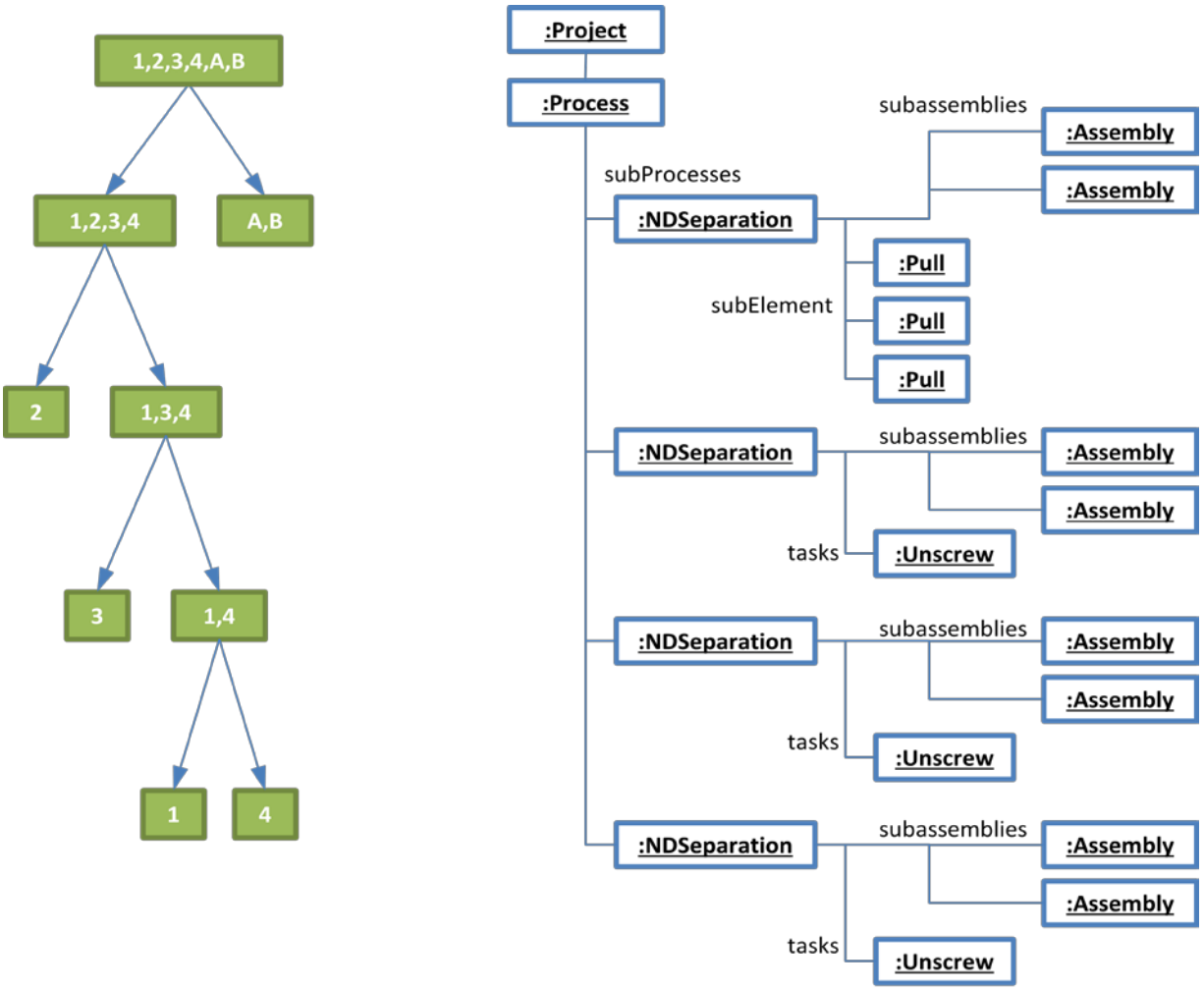

Figure 17. Relation between disassembly sequence and connection graph

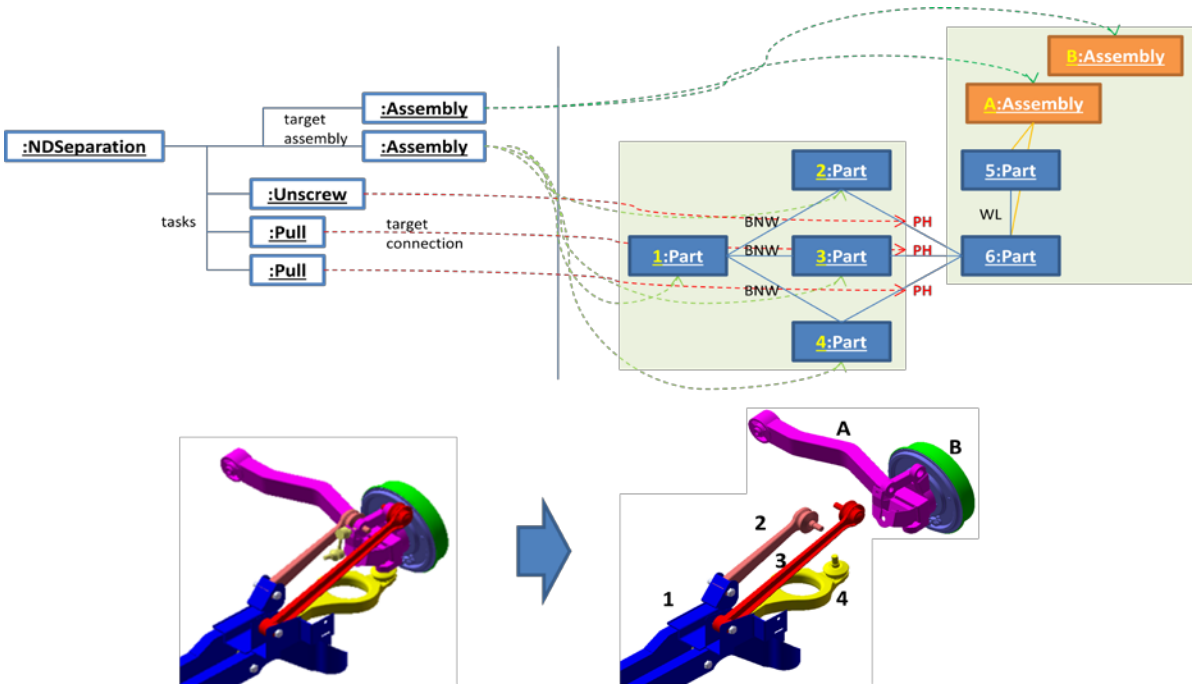

Figure 18. Relation between disassembly sequence and connection graph

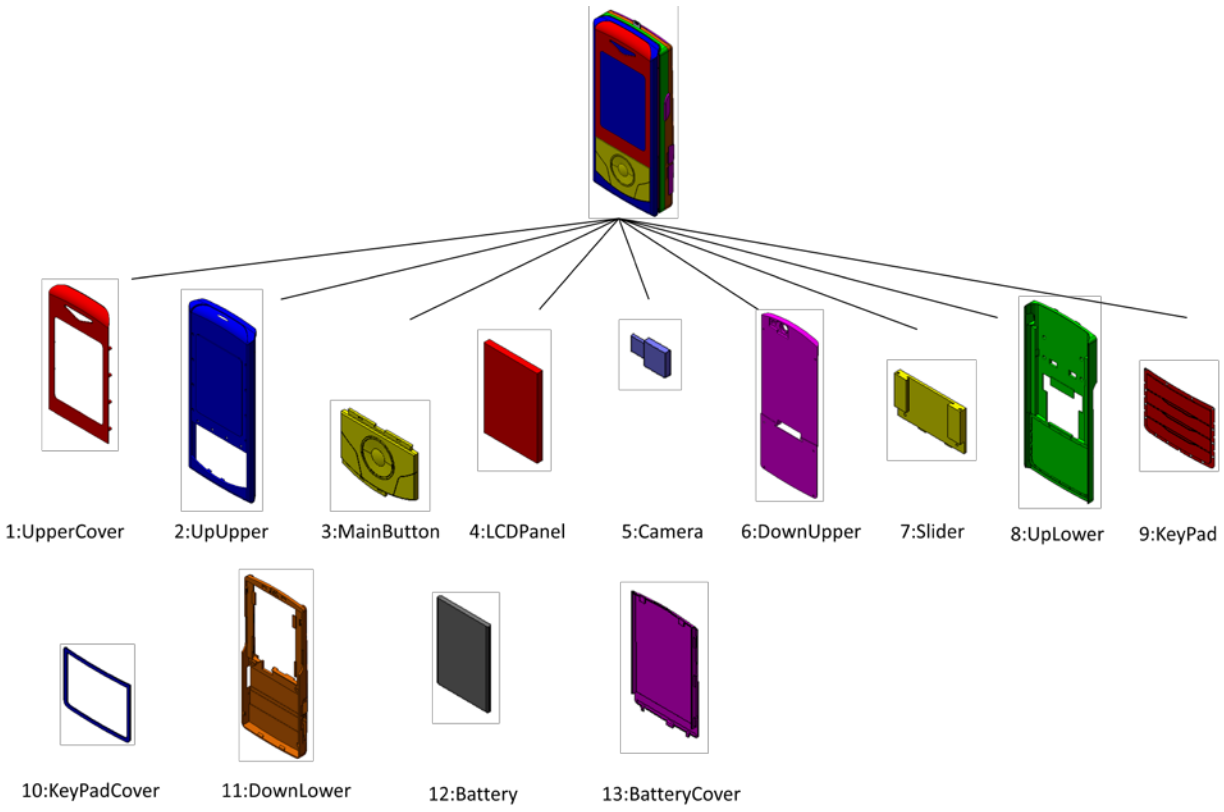

Figure 19. Assembly sequence of flip-top cell phone in a tree structure

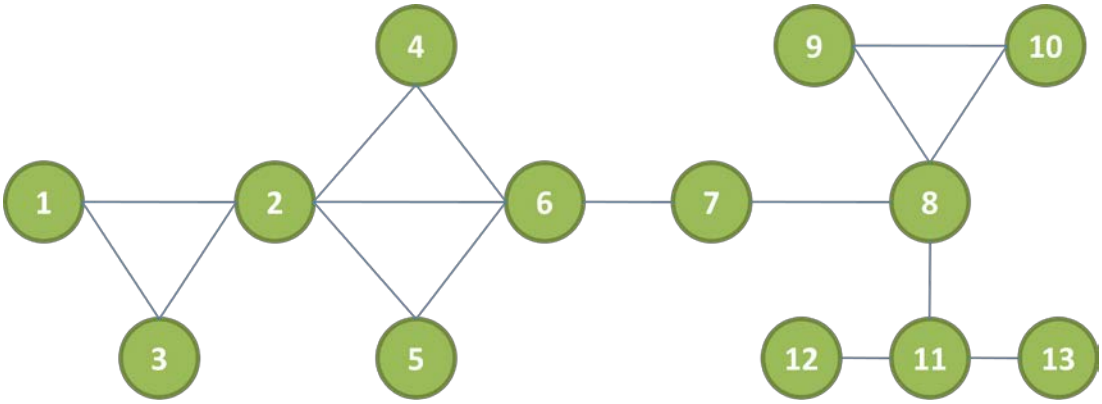

Figure 20. Connection graph

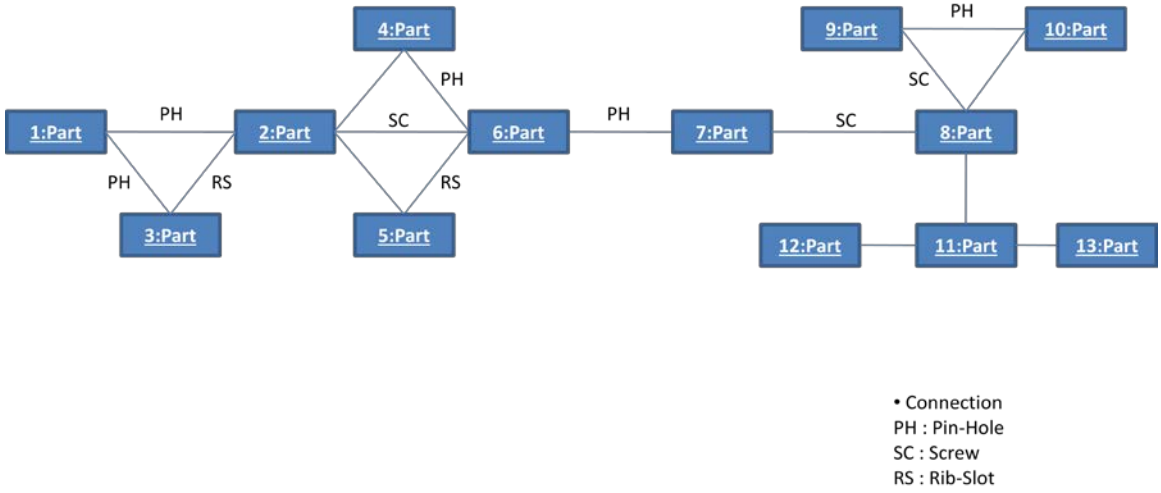

Figure 21. Instance diagram of a cell phone

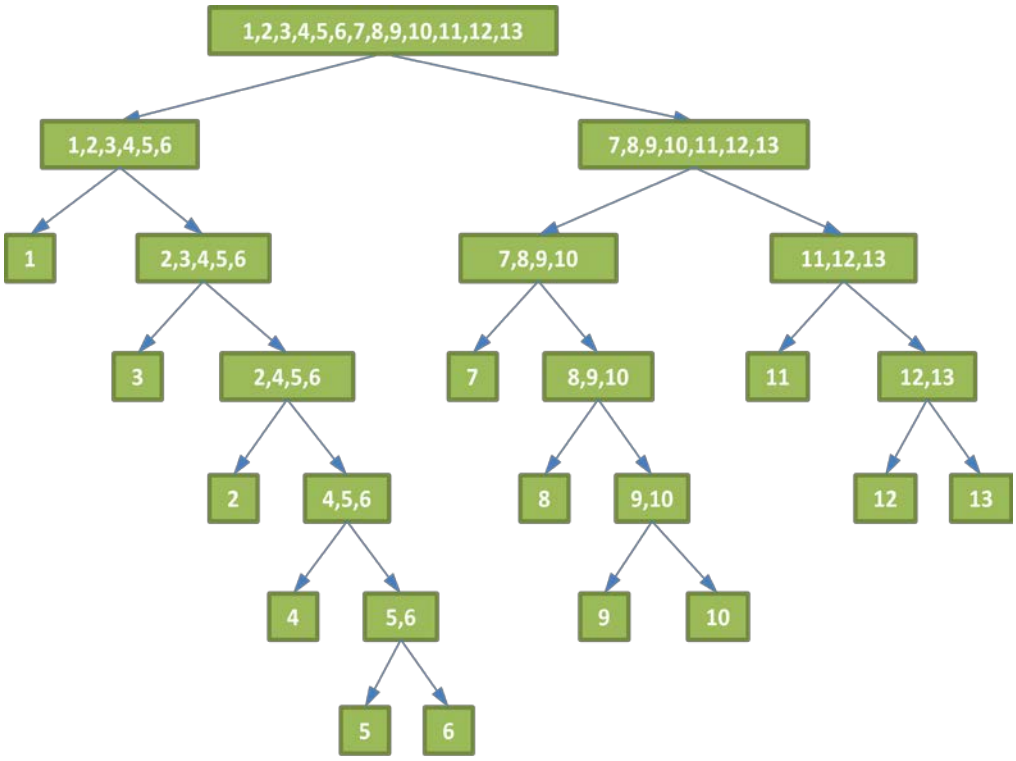

Figure 22. Disassembly sequence of a cell phone

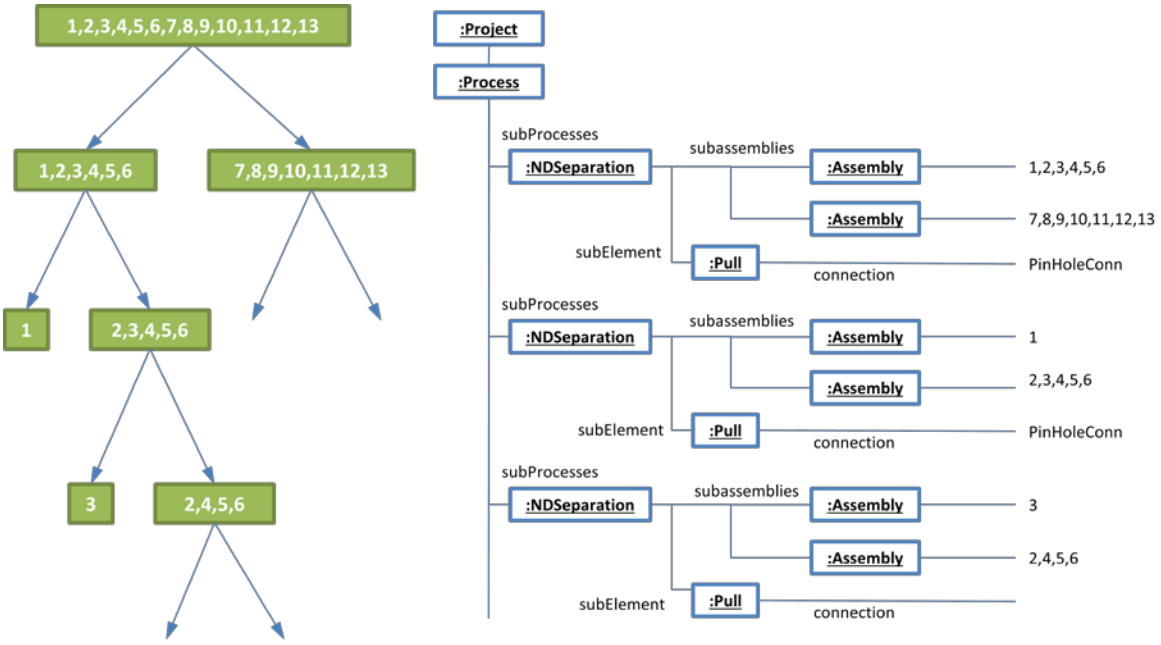

Figure 23. Instance diagram of the disassembly sequence of a cell phone