NISTIR 8262

Theory of Discrete Event Logistics Systems (DELS) Specification

Timothy Sprock
George Thiers
Leon F. McGinnis
Conrad Bock

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8262

National Institute of
Standards and Technology
U.S. Department of Commerce

NISTIR 8262

Theory of Discrete Event Logistics Systems (DELS) Specification

Timothy Sprock
Conrad Bock
Systems Integration Division
Engineering Laboratory

George Thiers
MBSE Tools, Inc.
Alpharetta, GA 30009

Leon F. McGinnis
Georgia Institute of Technology
Atlanta, GA 30332

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8262

June 2020

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

National Institute of Standards and Technology
Interagency or Internal Report 8262
Natl. Inst. Stand. Technol. Interag. Intern. Rep. 8262, 65 pages (June 2020)

Abstract

System models and model-based engineering methods have the promise of transforming the way that industrial engineers interact with production and logistics systems. Model-based methods play a role in improving communication between stakeholders, interoperability between systems, automated access to consistent analysis models, and multi-disciplinary design methods for complex systems. However, there remains a need for a foundation for modeling these kinds of systems – a foundation that tailors methods and tools developed in other engineering domains to the unique concepts and semantics of production and logistics. This foundation is the topic of this report.

This report documents a framework and model libraries for modeling discrete event logistics systems (DELS), an abstraction that covers manufacturing plants, material handling and transportation systems, warehouses, supply chains, etc. The DELS abstraction was created by identifying and modeling commonalities across the kinds of systems that industrial engineers typically encounter, and analysis models they use to analyze those system. It extends well-known product, process, and resource (PPR) ontologies to incorporate a library of operational control model components, and is connected to Commodity Flow Network (CFN), modeling networks, flow networks, and process networks. The relationship between DELS and CFN formally links system models to abstractions used to create analysis models, such as discrete event simulation.

This report is the first public release of models and documentation capturing many years of refinement and application by the authors. As a first release, the goal is to solicit additional use cases and feedback from the community to improve the models and make them the foundation for the model-based industrial and systems engineering community.

Key words

Smart Manufacturing; System Modeling.

Table of Contents

List of Figures

1. INTRODUCTION

A discrete event logistics systems (DELS) is:

- a network of resources, arranged in a facility; each resource has one or more processing capabilities, with a capacity for each capability;
- products flow through this network of resources, transformed by processes executed by the resources; a process might require capabilities of more than one resource; processes can change location, age, or condition of products.

The term "discrete" refers to the things flowing and process steps (transformations). The things flowing are separate from each other, e.g., individual product units, components of product units, or batches of product units. Process steps on the same product are taken separately. They have well-defined start and end events, e.g., the start of a machining or heat-treating process, even though our knowledge of the event time may be uncertain. Transformations (mostly) require resources that are separate from each other, e.g. subcomponents, or equipment, tools, fixtures, and input raw materials (discretized by units).

Factories are obviously a kind of DELS, but there are others. A warehouse is a DELS with much simpler resources and processes than factories. A supply chain is a DELS where the facility, rather than being a building, is the geographical organization of factories, warehouses, and transportation resources. A hospital also is a DELS, where the products are patients flowing through the hospital, the resources are staff and machines in the hospital, the processes are diagnostic, prescriptive, and general care activities performed on/for the people flowing through the hospital.

The term DELS is used in this paper as an abstraction of the many kinds of related systems that are extensively studied in Industrial Engineering, Operations Research, and Management Science (IE/OR/MS). These systems share some common characteristics, as do analysis methodologies and tools used to study them. These similarities can be captured in a framework (conceptual organization, abstraction), common language (syntax and semantics), and model libraries that simplify construction of DELS models.

The abstractions and model libraries in this document are designed for operations management decisions and analysis models supporting them, as required by IE/OR/MS stakeholders. Operations management is the layer between process/equipment and enterprise concerns [1], and the abstractions are intended as an intermediary, or bridge, between concrete, technological, embodiment models and analysis abstractions. Other concerns, such as part/process design and quality, equipment-level motion control and kinematics, enterprise level strategy (except resource investment, but not business operations concerns), etc. are outside of scope of this report.

This paper seeks to document the DELS model libraries (archived at [2, 3]), incorporating recent simplifications and extensions to [4, 5]. It focuses on DELS systems modeling 'infrastructure', analysis abstractions, and logical abstractions for defining and analyzing DELS. This report uses the Systems Modeling Language (SysML) [6] to present abstractions and model libraries. While it briefly describes the aspects of SysML needed, a reader not familiar with SysML can also refer to [6, 7].

Section 1.1 motivates the application of system models to DELS and the formalization of DELS abstractions to support development of those models. Then section 1.2 describes the modeling framework for the abstractions and provides an overview of the model library (summarized in figure 2). The remaining sections discuss network abstractions (section 2), DELS plant behavior (section 3), and finally DELS operational control (section 3.8).

1.1 Motivation

System models and model-based engineering methods have the potential to transform the way that stakeholders interact with their systems. This section describes some benefits and potential opportunities of model-based engineering ecosystems. At the base level, developing and integrating models including system models, abstractions of those models, and related analysis models; foster better communication between stakeholders, i.e., "are we all talking about the same artifact in the same way?" Streamlined communication and shared conceptualization between stakeholders can be translated into improved system interoperability and methods for operating and analyzing the systems (tool interoperability). Model-based methods and greater system and data interoperability directly support system (re-)design efforts. These projects can include small modifications, such as changing control algorithms; larger resource investment or shop-floor reconfiguration efforts; and can even be deployed to support greenfield design and commissioning of new systems. This section motivates the role of model-based methods in improving communication, interoperability, analysis accessibility, and design methods.

Communication Constructing system models turns tacit knowledge into explicit information, building a conceptualization of a system shared between stakeholders that have different viewpoints and concerns. Not only do these stakeholders have different viewpoints, but there are often terminological gaps between experts in different, often adjacent, domains. One gap that is of particular interest is the gap between industrial engineering practitioners and analysis experts, such as those constructing models for costing, scheduling, simulation, etc.

System models, as compared to analytic and geometric models, describe logical relationships between different aspects of the system and its environment. System models bridge human-interpretable descriptive models with machine-readable representations. These kinds of representations enable models to be constructed using defined (standard) syntax and semantics, to be stored in structured computer format (machine-readable, repository-based), and to be stored along with supporting metadata about the models [8]. Dedicated modeling languages such as SysML [6] are more expressive than analysis languages, enabling the development of precise analysis-independent system models that are not constrained by any target analysis language. In fact, what is created is platform-independent, agnostic of any implementation language, analysis or otherwise.

Interoperability Enterprise interoperability has traditionally focused on data exchange standards, including standard formats and controlled vocabulary / terminology. One way to improve the system (and ecosystem) functionality is to identify opportunities to improve the level of interoperability between data, functions, and systems [9, 10]; for example, expanding standardization efforts to include the content of exchanged information, including standard reference models and common workflow models.

The Object Management Group (OMG)'s Architectural Context document describes the purpose of Model Driven Architecture (MDA) as enabling "different applications to be integrated by explicitly relating their models, facilitating integration and interoperability. The three primary goals of MDA are portability, interoperability, and reusability." [11, 12]. Model-based methods may offer some support in developing contextual interoperability between enterprise applications, such as those supporting the manufacturing operations management ecosystem, and to analysis applications, such as simulation and optimization [13]. Increasing the quality of communication and interoperability between applications, people, and systems supports improved analysis, design, and operational environments.

Analysis Model-driven system-analysis integration methods enable analysis methods to interact by exchanging formal system models. Exchanging system models requires tools to interact with each other using standard data formats (syntax) that are interpreted in standard ways (semantics). For example, DELS simulation and optimization models would benefit from standard formats and interpretations for items flowing through a system (types and quantity), how they are flowing (path and resource), and control of that flow. System models, as compared to analytic and geometric models, describe logical relationships between different aspects of the system and its environment. Dedicated modeling languages such as SysML are more expressive than analysis languages, enabling precise analysis-independent system models that are not constrained by any target analysis language. Standard syntax and semantics to express the structure, behavior, and control of the system independent of analysis enables one system model to create many kinds of analysis models, including purpose-specific simulation and optimization models. For example, exchanging system models between simulation and optimization tools enables analysis models to be generated, or updated, when necessary to reflect a required view, new solution, etc. [14, 15].

However, developing and deploying appropriate model-driven system-analysis integration methods remains a challenge, especially when every analysis model is formulated from a unique abstraction of the system. For many practitioners, it is difficult to decide which analysis model/tool to use in a particular situation/context to answer a particular question. Often this challenge is compounded by the fact that multiple analyses may available to answer the same question, perhaps just at a different level of fidelity, robustness, quickness, etc. Can multiple, coherent analysis models be extracted, or built, from a single system model or multiple views of the same system model?

One research goal of this report is to formalize multiple abstractions used to create different analysis models, relate those abstractions to each other (“unify them”), and then connect them to system models.

Design Model-based systems engineering (MBSE) and design methodologies, though a common theme of our work, is not the focus of this report. Conceptual models based on agreed-upon terminology and semantics support the development of integrated and interoperable enterprise data, functions, and systems [13]. Design methodologies can leverage model libraries and reference architectures that capture reusable artifacts and best practices for assembling them into system models (see, e.g., [16]). Shared abstractions and reusable reference architectures are becoming essential for designing complex, interoperable systems. For example, designing self-similar system architectures that integrate make, move, and store functional capabilities requires a unified model of decision-making and abstractions that link decision-support (abstract resources) with execution (specific resources)

[17]. Finally, optimization and simulation are common methods supporting system design (trade-space exploration and high-fidelity validation); but can only be useful if they can be accessed efficiently and inexpensively [18].

1.2 Modeling Framework

Reference models created to support model-based methods can be reused and extended (specialized) when specifying new systems. These models identify commonalities across a family of system models, providing a language, model libraries, and patterns (best practices) for constructing new system models [19]. Reference models can be elaborated and extended as necessary. This method encourages discovery of common concepts and terms, an emerging ontology for system specification. For DELS, reference models should provide basic DELS concepts, support high-level subsystem decomposition (logical architectures or conceptual models), and provide templates for assembling subsystem components.

Here we follow the OMG's MDA framework [11] consisting of three layers: M2 is the language layer (UML/SysML), M1 contains models constructed using the language, and M0 represents instances of the models, i.e., actual systems, the data representing them, or simulations of them. Previous work in this area developed the DELS Specification as a domain-specific language, an extension of SysML, using its profiling mechanism [4, 5]. In that approach, systems models are related to the DELS specification through stereotype application. This paper seeks to unify the DELS models as M1 models rather than M2 SysML extensions. For example, here the commodity flow network (CFN) is modeled as an M1 model (used to instantiate and classify (describe) instances), rather than a domain-specific language (M2 syntactical extension of SysML). See [20] for a discussion on benefits of M1 abstractions. M1 models are related to their abstractions (DELS Specification models / reference models) through generalization, either by directly extending system model concepts or mapping them afterward.

Generalization Generalization is a method to organize things into taxonomies (classifications) by their similarity, defining specialized classes to elaborate differences within broader classes while retaining a relationship to them. Taxonomies constructed using generalization explicitly model the assumptions, extensions, and simplifications made in the classifications. Things that are logically similar can be organized by generalization. For example, trucks and forklifts can be generalized to mobile resources that carry pallets, mobile resources in general, or all resources. Classes can be specialized to capture differences between specialized things. For example, machines that execute subtractive manufacturing processes can be specialized into classes of milling machines and turning machines, or further into specific brands of milling or turning machines.

In the DELS modeling framework, `Manufacturing Systems`, `Storage Systems` (such as warehouses), `Transportation Systems`, and `Supply Chains` are all kinds of Discrete Event Logistics Systems (`DELS`) (figure 1). They are related formally to DELS definition by the generalization relationship denoted in SysML using a hollow-headed arrow directed from the more specialized class to the more general class. In this document, `teletypefont` will be used to denote UML classes or SysML blocks, *italics* will be used to denote properties (or roles) in classes or blocks, and **boldface** will be used to denote associations between blocks. The SysML models use PascalCase and lowerCamelCase for naming blocks and properties, respectively. However to increase readability of the report, spaces will be added between the words while preserving the capitalization and typeface.

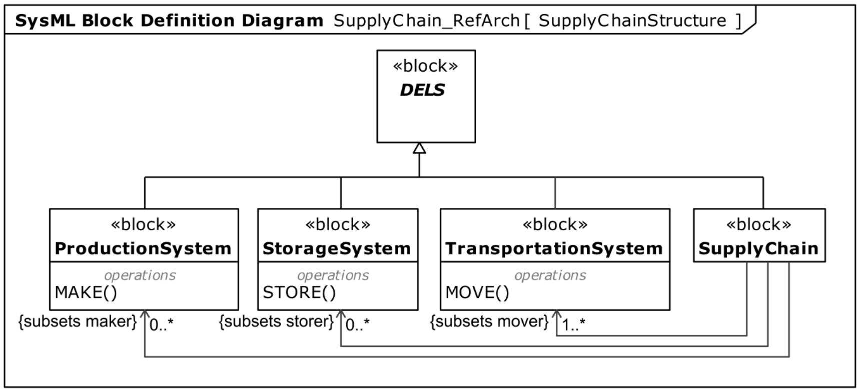

Fig. 1. Manufacturing systems, storage systems (such as warehouses), transportation systems, and supply chains are all kinds of discrete event logistics systems (DELS). This is shown by a generalization relationship from them to DELS.

The approach proposed here uses generalization to formalize the results of abstraction, rather than stereotype application. Generalization enables system models to be constructed (specialized) directly from abstractions, rather than mapped to the abstractions after the system model has been constructed, as with stereotypes. The resulting system model naturally conforms to the abstraction, because the abstraction is identified as the broader class. Abstractions can be retrieved correctly and efficiently from detailed system models. Model libraries and taxonomies constructed using generalization can be extended and specialized to incorporate new specific system behaviors and any corresponding analysis models, while retaining access to higher levels of abstraction. Generalization is supported in almost all modern programming languages, as well as UML, providing many more potential modeling platforms than stereotypes.

This report proposes a modeling framework organizing the DELS domain using a multi-layered abstraction (figure 2). Generalization is used to organize the reference system models and link them to abstractions and concrete models. The model layer (M1) is organized into roughly three layers: the *Top* contains the analysis and logical abstractions (commodity flow networks (CFN) and DELS), the *Middle* contains domain-specific reference models and architectures, and the *Bottom* contains system models built from the reference models. These layers are formally connected via generalization enabling traversing from specific system models to abstractions used for developing conceptual models and integrating supporting analysis tools. This report documents the abstract models in the *Top* layer (CFN and DELS).

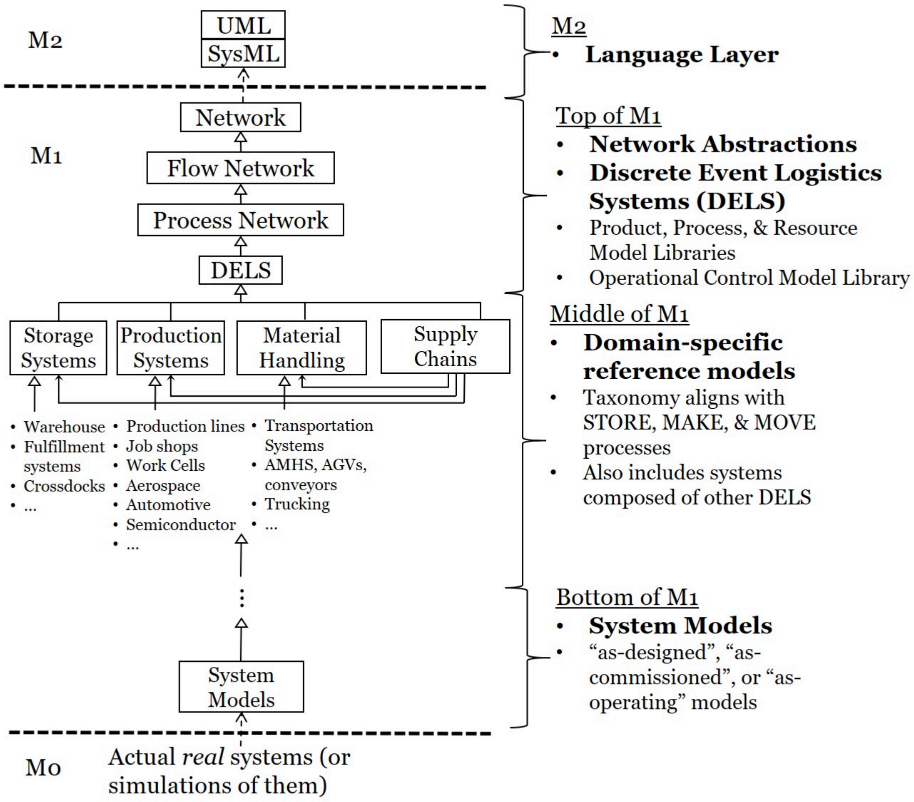

Fig. 2. DELS multi-layer architecture organizes model libraries from most general to most concrete, with generalization relationships linking the layers.

DELS Specification The *Top of M1* contains the DELS reference model which is extended from network definitions defined by the CFN. These levels capture (stable) abstractions that are useful for developing conceptual models and logical architectures [21]. These models are specific enough to understand what's flowing, how it is flowing, and the control of that flow, without specifying particular technology implementations. These models are at the same level of abstraction as many IE/OR/MS analysis models proposed to support design and operational decision-making.

The *DELS* layer contains common concepts and terminology organized around a product, process, resource (PPR) ontology, and includes facility descriptions, work (task) definition, and control of flows and transformations (operational control). The reference model includes model libraries and taxonomies supporting each concept.

Domain-specific reference models and system models can be created by specializing these abstract, conceptual models into new domain-specific concepts. Likewise, system models can be mapped, or generalized, to these abstract models to access associated analysis libraries.

Domain-specific DELS Specializations The *Middle of M1* contains reference models and architectures for systems specialized from DELS, such as production, material handling (transportation), and storage systems (see, figure 1). This specialization (generalization set) is organized by the primary system functions: *Make*, *Move*, and *Store* expressed by the *Operations* on each block. These models introduce concrete domain-specific terminology for the products, processes, and resources; e.g., trucks rather than resources.

These specializations are classified by each system's high-level core functionality, e.g. production systems *make* commodities. Most DELS, including manufacturing plants, supply chains, and warehouses; are composed of (or created by assembling) subsystems specialized from these abstract components. These systems (as specialized DELS) may be further (de-)composed into functionally specialized components; for example, a production system may be composed of material handling and storage systems as well as smaller, more specialized production systems.

System Models The *Bottom of M1* contains the most detailed system models. These models are created by extending the domain-specific reference models in the middle layer, and then adding details specific to a single system. These detailed system models may include design specification models ("as-designed") that contain sufficient detail to commission new systems. These models can also be created as documentation for existing systems (“as-commissioned”). System models created most likely will not or can not be directly reused as they represent a single system (or identical systems). However, recurring patterns for creating these detailed models can be harvested into reference models in the middle layer.

Typically, we are interested in extending the taxonomy by specialization (more refined classifications). However, developing reference architectures follows a complementary process of harvesting common patterns through abstraction (generalization) to classify and organize existing domains [22]. Each taxonomic layer contains additional specializations that refine the abstract definitions into increasingly concrete system models.

2. Network Abstractions

Network-based abstractions are common in DELS modeling because of their widely-understood mathematical interpretation, suitability to many algorithms, and applicability to a broad range of (abstract) analysis questions about DELS. These well-studied abstractions have produced many domain-specific analysis methods, such as finding shortest paths and optimal facility locations [23], determining throughput for (multi-commodity) flow networks [24], as well as service time and utilization in queueing networks [25, 26].

Formalizing network abstractions and applying them to analysis model construction was first described in [4]. It also introduced token flow networks as a unifying abstraction for DELS networks, covering basic networks, flow networks, and process (or queueing) networks — basic networks introduce structure and relationship; flow networks introduce flows; process networks introduce transformation (and duration). The network abstractions and DELS abstraction are separated, but formally linked using generalization relationships.

2.1 Basic Networks

This section formalizes characteristics common to all DELS networks. The term *network* in this report refers to all M0 (actual, digital, or simulated) networks, rather than models of these networks (e.g., graph syntax). For example, general network properties, such as “node criticality”, can describe aspects of specialized networks, e.g., the importance of a particular depot in a supply chain modeled as a specialized network.

Networks are composed of other networks and links between them playing the roles of nodes and edges, respectively. In SysML, this is expressed as a block `Network` with a part *node* typed by `Network` (kind of things playing the part of *node*). Composition is a whole-part relationship, shown in SysML by black diamond associations between blocks, with the whole on the black diamond end (*parentNetwork*) and the part on the other end (*node*). This recursive composition relationship enables network models to be decomposed or refined with additional internal details (hierarchical nested network representation). In SysML, leaf-level (atomic) networks redefine their *node* property to multiplicity [0] indicating that no further decomposition or refinement is allowed.

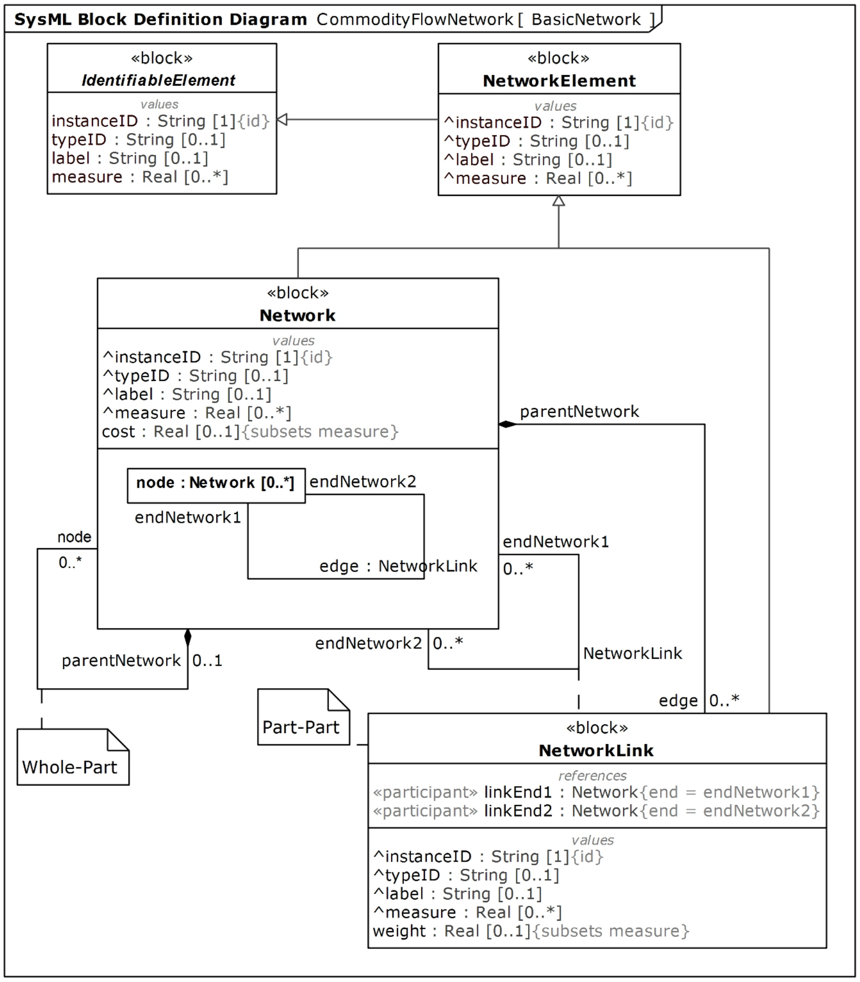

Fig. 3. Basic `Networks` are composed of *nodes* (typed by `Network`) and *edges* (typed by `NetworkLink`). Both `Network` and `NetworkLink` are specialized from a top-level `NetworkElement` block.

Part-part relationships in SysML are shown graphically in block compartments as connectors between parts (lines between rectangles). Connectors are also parts (roles), but are typed specifically by association blocks, which classify M0 links between the things playing the connected parts. Connectors between nodes in a Network are *edges*. Edges are parts typed by `Network Link` (roles played by network links), an association block for linking networks. This enables relationships between networks to be specialized as needed by applications. Networks refer to their linked networks through the ends of the `Network Link` association (*endNetwork1* and *endNetwork2*). Each network link (M0 instance of `Network Link`) identifies its two participants by *linkEnd1* and *linkEnd2*. Generally, association block (`Network Link`) references its participants by different context-specific roles (*linkEnds*) than how `Networks` reference other Networks (*endNetwork*). `NetworkLink` has a specialized *measure* called *weight* that is used to model the strength or capacity of the link between two nodes in a network.

2.1.1 Network Element

Every network and network link requires some common information, mostly to identify the object and what kind it is. The `Identifiable Element` block defines three properties for all networks and network links: *instanceID*, *typeID*, and *label*. *instanceID* gives a unique identifier for each network and link, while *label* provides a colloquial identifier, or "native" name. *typeID* tells the kind of network or link it is, such as "supply chain" or "transportation edge". Analysis languages and tools often do not support typing - systems and objects are "classified", or organized, by their *typeID* instead. This means the analysis tools can not represent taxonomies of network elements like more expressive languages, such as SysML. Typing-systems, based on formal taxonomies, are useful for checking the correctness of models and enforcing pre-defined constraints at run-time.

`Identifiable Element` defines another property *measure* for adding measurable properties as subsets (such as *cost {subsets measure}* on `Network`). Subsetting is a kind of specialization for properties, linking a specialized property to a more general (subsetted) one. It enables properties to be specialized while maintaining traceability to the more general property.

`Network Element` specializes `Identifiable Element` capturing analysis-specific commonalities between `Network` and `Network Link`. At the time of this release, no additional commonalities have been identified, but it's left for future use. Block specialization and property subsetting will be used extensively as `Network Element` and its properties are specialized in the rest of the DELS framework. The properties defined in `Network Element` are inherited by every block and association in the DELS framework, ensuring consistent identification and simplicity in implementing these models. Properties inherited from a more general block are denoted in SysML using the caret notation (^), e.g. `Network`'s *instanceID* is inherited from `Network Element` (figure 3).

2.2 Flow Networks

Flow networks are networks that commodities can flow through. Commodity is used here to describe (abstract) all generic objects that enter, exit, and flow through networks. Commodities are modeled in section 2.2.2. Commodity flow network abstractions are used in many kinds of analysis models, including discrete event simulation. This section formalizes multi-commodity flow networks described in [24] (figure 4).

`Flow Network` specializes `Network` and its properties. It has two parts: *flowNode* (typed by `Flow Network`) and *flowEdge* (typed by `Flow Network Link`), specialized from `Network`'s *node* and *edge*, respectively (figure 4). Property specialization is expressed in SysML using subsetting or redefinition. In the `Flow Network`, *flow Nodes* are a subset of all *nodes* ({subsets node}) in this kind of network, i.e. there may be a mix of nodes, some that commodities can flow through and others that do not support commodity flows. Other properties from basic networks are also specialized, such as *sourceFlowNetwork* subsetting *endNetwork1* for networks to refer to others linked to them. `FlowNetworkLink` is specialized from `NetworkLink`, and each property *subsets* its respective `NetworkLink` property, providing traceability to between special and general blocks and properties.

`Commodity` types the *inputs* and *outputs* flow properties of `Flow Network`. `Commodity` is elaborated in section 2.2.2. Flow properties are properties that specify the kinds of things that might flow between an object and its environment. They are appear with the stereotype «flow property» in property compartments or in *flow properties* compartments. Commodities that a `Flow Network` *produces* and *consumes* are a subset of all commodities it *outputs* or *inputs*, respectively (shown by *{subsets outputs}* and *{subsets inputs}*). `Flow Networks` have a property (*currentlyFlowingThrough*) that specifies the commodity objects currently flowing through (or located in) the `Flow Network`.

Commodities also flow across *flow edges* (typed by `Flow Network Link`) from source to target. This is captured as a SysML item flow across the connector, shown by a solid black triangle in the IBD compartment of `Flow Network` (figure 4).

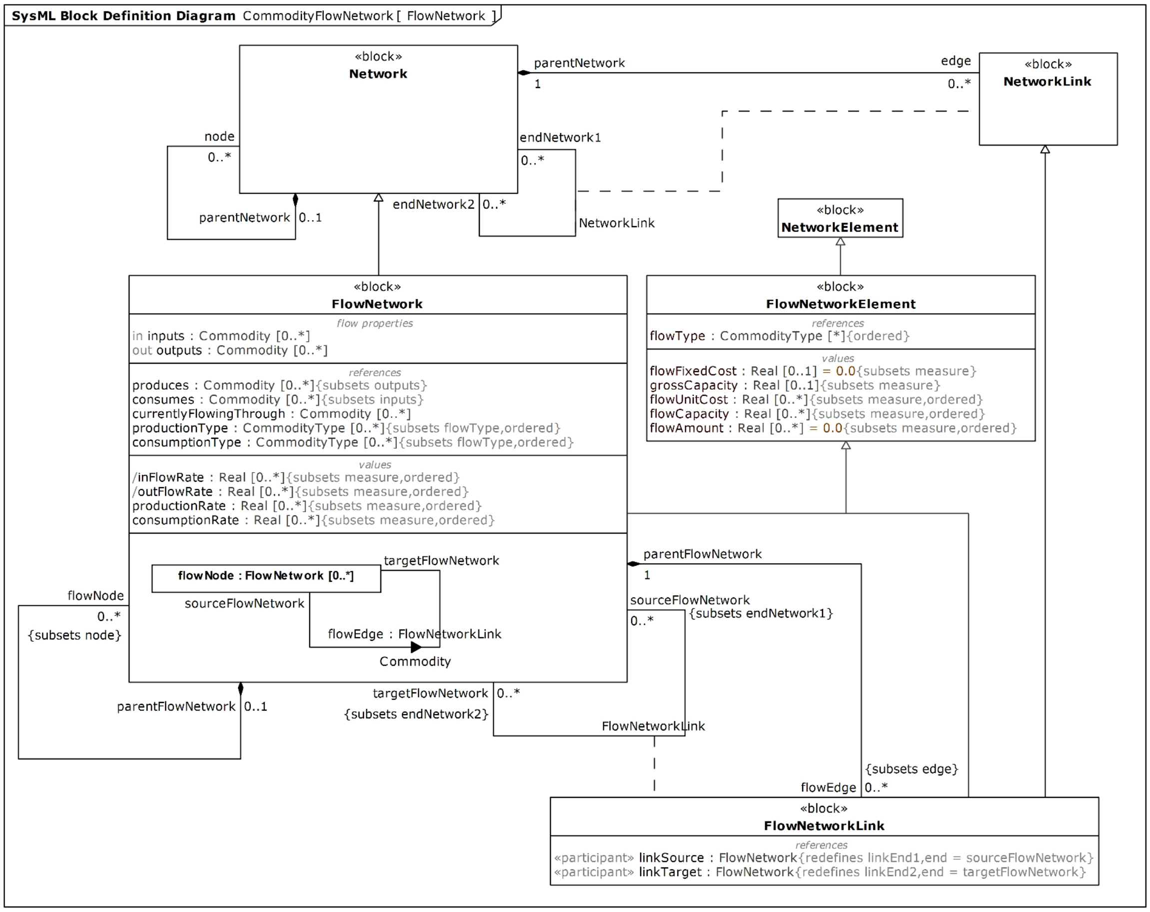

Fig. 4. Flow Networks are a foundation for many kinds of analysis models, including discrete event simulation.

2.2.1 Flow Network Elements

`Flow Network Element` (specialized from `Network Element`) captures commodity flow-related properties common to Flow Networks and Flow Network Links. *flowType* is an ordered set ({*ordered*}) of commodity types that are flowing (or are allowed to flow) through the element. Other ordered properties on the block give information about these types in the same order. For example, *flowCapacity* is the maximum flow rate of each type of commodity across the flow edge and *flowUnitCost* gives the per unit cost for each commodity type to traverse the edge. These properties must have the same number of values as *flowType* to match capacities and flow costs to commodity types. The property *flowAmount* captures the aggregate number of `Commodity` objects of each type (currently) flowing through the `Flow Network Element` (derived from the *currentlyFlowingThrough* property). Other properties are not specified by type, *grossCapacity* gives the maximum flow rate of all commodities across the flow edge and *flowFixedCost* gives the fixed cost of any flow traversing the flow network element.

Some `Flow Network Element` properties have values that give current time values and others are restrictions on current time values. For example, *flowCapacity*, *grossCapacity*, and *flowType* properties only restrict values at current time values. But *flowAmount* is a current time value, either streamed in real-time or reported ex-post as a metric. Constraints on current time values defined in OCL would useful for implementing optimization models, such as multi-commodity flow networks [24].

`Flow Networks` have additional metrics derived from other properties: *inFlowRate*, *outFlowRate*, *productionRate*, and *consumptionRate*. These properties give the amount per time period of commodities flowing in and out of the `Flow Network`. The rates are derived from *inputs*/*outputs* and *produces*/*consumes* properties, aggregated by each kind of commodity (ordered by *flowTypeAllowed*'s ordered set of commodity types). Actually, *productionRate* and *consumptionRate* are ordered by *productionType* and *consumptionType* which are subsets of *flowType*.

2.2.2 Commodity

Commodities can flow through Flow Networks, following multi-commodity flow network abstractions. A commodity is an economic good or service that has full or substantial fungibility: the market treats instances of the good or service as equivalent or nearly so, with no regard to who produced them (individual units are essentially interchangeable). Fungibility simplifies formulation of many kinds of analysis models.

`Commodity` is specialized from `Identifiable Element` (figure 5) rather than `Flow Network Element`, because commodities are not inherently parts of flow networks. The abstract `Identifiable Element` supports the commonalities of (Flow) Networks and Commodities. This covers cases where commodities exit networks and are no longer elements of them.

The `CommodityType` block (specialized from `Identifiable Element`) and its association to `Commodity` facilitates connecting these models to analysis models and information systems. For example, analysis models might specify constraints on execution by type, e.g. only this type of commodity is allowed to flow along this edge, or this node creates five of type A each period, and information systems often track items by type, e.g. stock keeping unit (SKU). The `Commodity-Commodity Type` association is an example of reflection, i.e.,

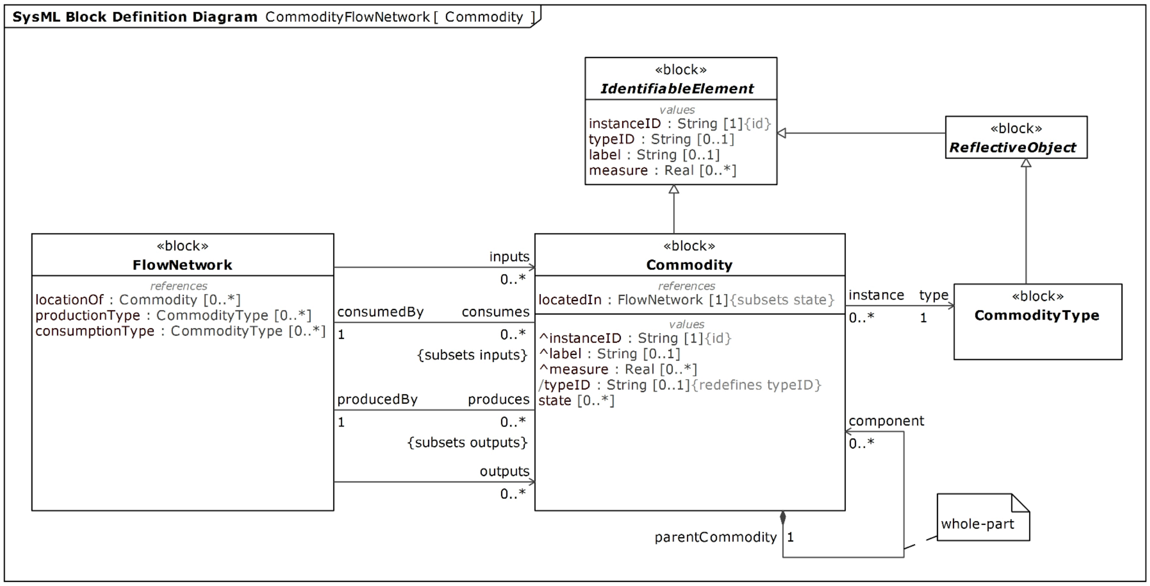

Fig. 5. Commodities can flow through Flow Networks, derived from multi-commodity flow network abstractions.

giving access to type information (M1) at run-time, indicated by specializing `Commodity Type` from `Reflective Object`, an implementation model of this capability. Instantiating a `Reflective Object` yields an object that acts like an M1 block, rather than a physical object. For example, a SKU (a distinct type of item for sale) is an instance of `Commodity Type`, while items in inventory (the things that flow) are instances of that `SKU`. Most implementation languages provide methods to convert *type*:`Commodity Type` to *typeID*:`String`.

`Commodity` types `Flow Network`'s *inputs* and *outputs* properties and their respective subsets *consumes* and *produces*. The *produces* property gives the commodities arriving at the network, which increases the total *flowAmount* of that kind of *Commodity* flowing through the system, while *consumes* gives the commodities leaving the network, which decreases the total *flowAmount* flowing through the system. `Commodity` is *flowingIn* (typed by a `Flow Network`), defined as part of (subset of) its *state*. Finally, `Commodities` can be composed of (part of) other `Commodities` playing the *component* role.

2.3 Process Networks

`Process Networks` extend `Flow Networks` (inheriting flow semantics) to add transformation of inputs to outputs and duration of transformation. DELS Processes (section 3.3) extend this generic (abstract) transformation to model, for example, transformations of parts/materials or capabilities of equipment performing the transformation. `Process Network` is a simplified (abstract) model that omits resource requirements and contention, which are added in the DELS extension. Process networks are suitable for producing low-fidelity analyses such as queueing network models [26–28]. This section treats processes as kinds of networks to maximize the applicability (reuse) of network analyses.

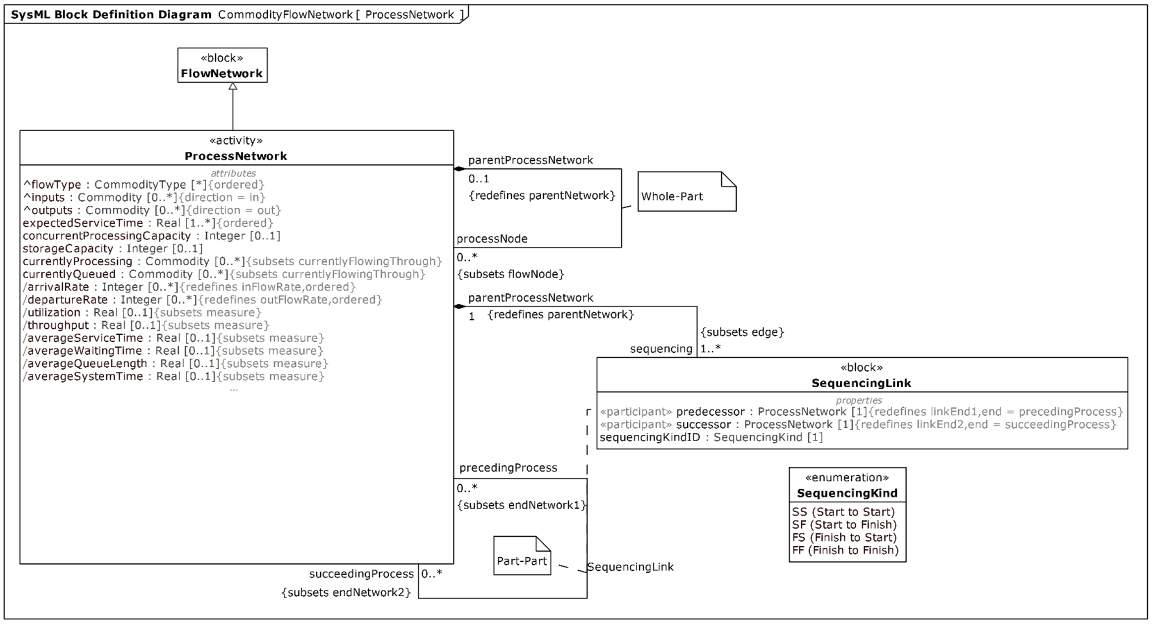

Fig. 6. Process Networks extend Flow Networks and specify transformation of flows through the network.

`Process Networks` are composed of *processNodes* typed by `Process Network` and subsetting *flowNodes* of `FlowNetworks`. `Process Networks` have two kinds of connectors (part-part relationships) between *processNodes*: *flowEdges* inherited from `Flow Network` and *sequencing* (typed by `Sequencing Link`). These enable specification of flows and time sequencing between transformations (process nodes), respectively. *sequencing* subsets *edges* from `Networks`. Process networks refer to others sequenced before and after them through ends of the `Sequencing` association (*precedingProcess* and *succeedingProcess*, subsets of *endNetwork1* and *endNetwork2*, respectively).

`Sequencing Link` has a property *sequencingKindID* (typed by enumeration `Sequencing Kind`) that gives the kind of sequencing expected between *predecessor* and *successor* processes. These include: *Start-to-Start*, where the successor process cannot start until the predecessor process does; *Start-to-Finish*, where the successor process cannot finish until the predecessor process starts; *Finish-to-Start*, where the successor process cannot start until the predecessor process finishes; and *Finish-to-Finish*, where the successor process cannot finish until the predecessor process does (the time lag on these can be nearly zero) [29]. Binary sequencing can be represented in a matrix and transformed to traditional queueing network analyses. However, more complex timing relationships might need more expressive languages, such as [30] (see section 3.3 for more discussion).

`Process Network` inherits *inputs/consumes* and *outputs/produces* properties (typed by `Commodity`) from `Flow Network`, as well as the *Rate* properties *inFlowRate/outFlowRate* and *productionRate/consumptionRate*, and *Type* properties *productionType/consumptionType*. `Process Network` redefines *inFlowRate* and *outFlowRate* to *arrivalRate* and *departureRate*, respectively, to reflect queueing network analysis terminology. The *Rate* properties are ordered in the same way as the corresponding *Type* properties, to give rates for each `Commodity Type`. To match *Rate* and *Type ordered* properties, corresponding properties (a type-rate pair) must have the same number of values. In SysML, Activities are also Blocks allowing modelers specify the structural aspects of a behavior, such as metrics, relationships and classification, while also being able to use them to construct Activity models (diagrams).

`Process Networks` have an *expectedServiceTime* (ordered by `CommodityType` specified by the *flowType* property) for the duration of their transformations. Each network has a *concurrentProcessingCapacity*, the maximum number of commodities it can transform at one time. [1] The process network also has a *storageCapacity* giving the maximum number of commodities that can be waiting for transformation. Corresponding to these `Process Network` has two roles for `Commodities` that redefine `Flow Networks`'s *currentlyFlowingThrough*: *currentlyProcessing* and *currentlyQueued*.

Specialized `Process Network` *measures* record metrics calculated by queueing network analysis models. The *measures* modeled here are taken from [31], and include: *utilization*, *throughput*, *averageWaitingTime*, *averageQueueLength*, and *averageSystemTime*.

3. Discrete Event Logistics Systems

DELS are defined by `Products` they create (or transform), `Processes` they execute, `Resources` they own (or can obtain), `Facilities` (environments) they operate in, and `Tasks` they service. Product, process, and resource (PPR) models are common abstractions for developing manufacturing system and analysis models; see, e.g., TOVE [32], MPSG [33],

---

[1] *concurrentProcessingCapacity* is an abstraction of server count concepts in queueing network analyses [26, 27]. Resources, such as servers, are introduced in the more concrete PPR ontology (section 3.1).

OZONE [34], IDEON [35], MSE ontology [36], ISO 15531 MANDATE [37, 38], CMSD [39, 40], MASON [41], and the survey of existing smart manufacturing standards incorporates a PPR organization [42].

The DELS model adds facility to PPR concepts for capturing system layout and organization, and tasks as the unit of work and authorization (PPRFT) (figure 7). It is complemented by a layer of operational control over resource assignments, task and resource flows (specialized commodities), and process executions (PPRFT+control). This is a simple top-level ontology describing DELS, abstracting and organizing the diverse terminology used across specialized domains. Figure 7 captures the general relationships between these high-level DELS concepts, which are summarized below and expanded in sections 3.1-3.2.

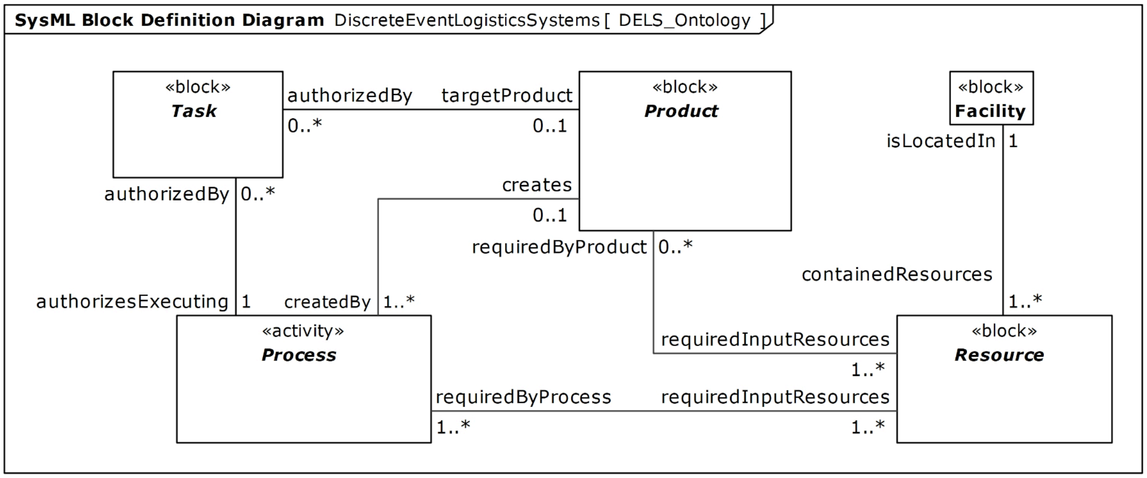

Fig. 7. The DELS ontology extends product, process, and resource (PPR) with facilities and tasks.

- `Product` is *createdBy* executing a `Process`, where there may be more than one process plan for a given part (denoted *1..**). In manufacturing models, the process relationship can be redefined as “processPlan”, but process plans do not exist in logistics systems, so *createdBy* is a more general role. Similarly, executing a `Process` can *create* a `Product` (denoted *0..1*). This covers cases where the `Process` is a service, changing the state of something but not necessarily creating anything. As with flow and process networks, we distinguish between a commodity being created by a node and one being output by a node (simply released in the same form after processing).
- `Product` and `Resource` have a **RequiredBy** association where some kinds of `Resources` are *requiredByProduct* (to distinguish from process inputs). `Product` has a inverse role for `Resources`, *requiredInputResources*.

- `Process` and `Resource` have a **requiredBy** association. `Process` defines the role *requiredInputResources* and `Resource` defines a inverse role *requiredByProcess*. This relationship is important for formulating scheduling problems.
- The DELS model refines the roles of `Resources` relative to `Product` and `Process`:
  - The **requiredByProcess** relationship is refined (subset) into **canExecute** for designating some kinds of resources, called `Active Resources`, as having some capability to execute a process, as well as being required (Section 3.1.1). For example, a machine (`Equipment`) executing a material forming process might also require auxiliary / passive resources.
  - `Product` is defined by its *billOfMaterials*, a collection (*derivedUnion*) of `Material` (specialized `Resources`, see Section 3.4).
- Each `Resource` *isLocatedIn* a `Facility`, which defines the system layout (geographic and geometric aspects) of resources and material flow (paths). For example, it might represent a concrete building for a production system, or a logical entity, such as layout of a supply chain.
- `Tasks` authorize and define units of work through references to both `Process` and `Product`.
- `Process` and `Task` have an **Authorization** association where each execution of a `Process` is *authorizedBy* any number of `Tasks`. Each `Task` *authorizesExecuting* exactly one `Process`.
- `Product` and `Task` have a **AuthorizeCreation** association where creating the *targetProduct* is *authorizedBy* a `Task`. Each `Task` might result in a Product, but also might not output anything.
- Regarding `Task` models, this modeling framework encourages specifying both the `Product` and `Process` authorized by the `Task`. Many (production) systems define the unit of work only by what it outputs; for example, a workorder authorizes the production of a part and it may even be ‘typed’ by the product. Here we have an explicit relationship to the process too; for example, a workorder authorizes the execution of a process plan that creates the same part that is authorized to be output.

Process plans often have no name, but we provide generic top-level names, for example, *MakePartX()*). Associating tasks with both parts and processes unifies cases where the process is merely a service (e.g. move, store, test) and cases where it produces a commodity as well.

Models built directly from the abstract DELS model libraries serve as conceptual models and common logical architectures for specialized DELS domains. These descriptions are a starting point for building more complex domain-specific reference models and connecting them to analysis models, without being overly prescriptive. The following sections elaborate model libraries associated with each concept to support modeling and specification of DELS models: Resource in section 3.1, Process in section 3.3, Product in section 3.4, Facility in section 3.5, Task in section 3.6, and DELS interfaces in section 3.7. An introduction and overview of the operational control layer is presented in section 3.8.

The PPR models reference at the beginning of this section are inherently product focused, a very traditional view of "what does this system need to deliver?" However, this document intentionally presents resource and process before product to focus the discussion to "what is this system capable of doing?" With this view of the system, the operational control layer focuses on managing those capabilities to satisfy product and service requirements specified by the customer.

3.1 Resource

`DELS` own `Resources` involved in `Process` execution, either as performers (such as equipment) or as consumable inputs (such as materials). Resource-related decision problems, such as investment or allocation, are among the most widely studied topics in industrial engineering, e.g., in warehousing [43], humanitarian and disaster relief [44], health care logistics [45, 46], transportation logistics [47–49], and manufacturing [50]. Consistent and precise resource behavior models remain a challenge, despite the attention devoted to studying resource problems.

Resources behavior models (models of computation) and interfaces define how DELS interact with each resource object (given its role and type). Capability modeling is one aspect ("what can it do?"), another is "how much can it do?" or "how can its capacity be allocated to do work?" In addition to defining interaction patterns, behavioral models are essential for scheduling (optimization) and simulation modeling, see, for example, OZONE [34] and DRiP [51].

Part of the challenge in creating standard behavior models is the existing literature gives different names to functionally similar resource types. For example, resources which can only perform one operation at a time might be called disjunctive resources [52], dedicated resources [53], or atomic resources [34]. Additionally, many analysis modelers leave details of resources implicit, resulting in inconsistent and incomplete representations.

Unified resource terminology and behavioral definitions simplify modeling and analysis of resource planning and scheduling problems. Resource definitions in this section are drawn mostly from the OZONE ontology [34], which builds upon the Generic Enterprise Resource Ontology [54] and [55], as well as the Dynamic Resource Allocation language [51]. [56] propose an object-oriented manufacturing resource modeling language to encapsulate manufacturing system knowledge. MANDATE [37, 38] considers three aspects of resources: (1) their description (the way of using and maintaining them); (2) the description of the activities, operations and functions a resource is able to achieve (its capacity and capability); and (3) the model of information needed to define, operate, trigger, estimate and monitor the resource.

The resource model is organized as a taxonomy with orthogonal branches covering multiple aspects of resources. These aspects can be combined to describe a single resource object. The first branch describes capability (section 3.1.1), the second availability (when work can be assigned) (section 3.1.2), and the third aggregated resources and resource networks to enable greater capability or capacity (section 3.1.3).

3.1.1 Capability: Active and Passive Resources

One distinction in resource behavior is some resources execute transformations (`Active`), while others are inputs to transformations (`Passive`). In most cases, resource objects are only one of these at any particular time: other things flow through them (active) or they flow through other things (passive). Some analysis models, like process-oriented petri nets, conflate these by modeling active resource, such as machines, as “flowing” to process executions; see for example, [57].

The model library reflects this distinction by specializing `Resource` into `Active Resource` and `Passive Resource` (figure 8). `Active Resources` are specialized from `Flow Network` to facilitate commodity flows through a network of resources. Passive Resources are specialized from `Commodity`, enabling them to flow. For simple analyses modeling passive flow, the flow semantics of `Flow Network` can be reused directly (where `Active Resources` play the *flowNodes* roles and are connected by *flowEdges*). Active

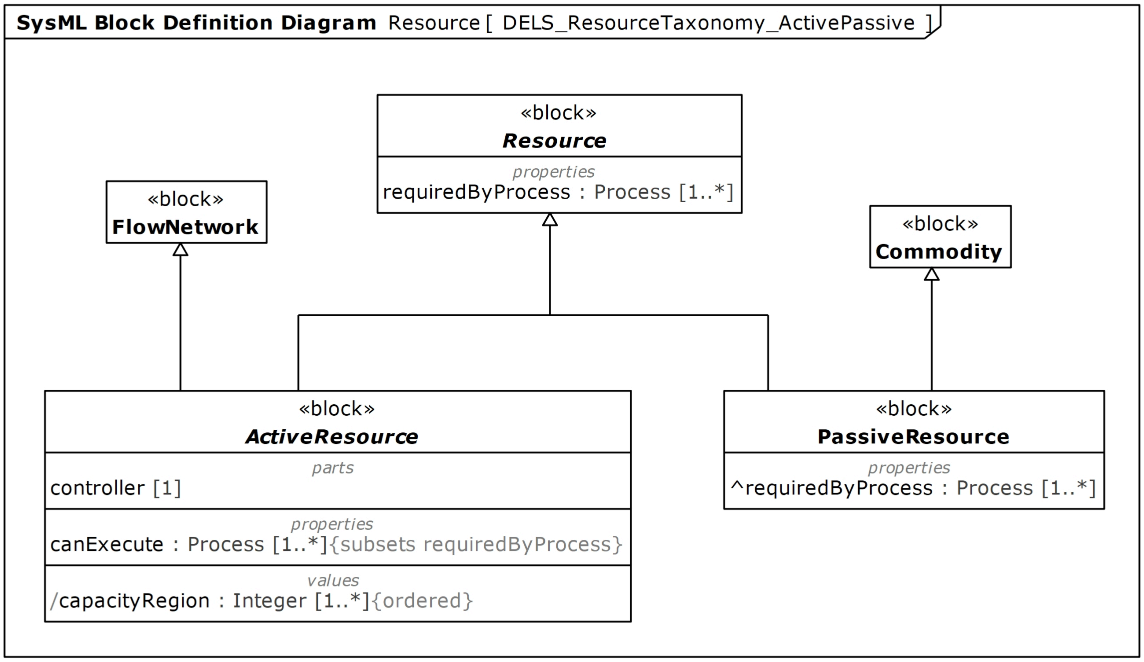

Fig. 8. Branch of the Resource taxonomy distinguishing resources that execute processes (active) from inputs to processes (passive).

and passive resources are used in [51, 58].

`Active resources` are typically regarded as performing the process, where `passive resources` are used or consumed during the execution of a process. From [51], "Active resources are the resources that we are managing. Passive resources enable the active resources to do their job (and if there are not enough of them, then they prevent active resources from doing their job)." Formally, `Active Resources` have a property *canExecute* typed by `Process`. `Passive Resources` type the *requiredInputResources* property of `Processes`.

An `Active Resource`'s *controller* property denotes a requirement for an unambiguous definition of how the behavior is executed, including some information processing involved in executing the behavior (i.e. not a hammer or mousetrap). Intuitively, we would expect an `Active Resource` to implement a callable-operation for invoking each `Process` that it *canExecute*. This may be modeled by a single *do(Process)* parameterized by the process's *typeID*, similar to passing a control program to a machine and saying *start/execute()*. This is a simplification of the implementation details, but sufficient for developing conceptual models.

Distinguishing `Active` and `Passive` resources also helps codify common analysis modeling techniques /transformations, such as those noted in ROPN versus POPN [57] or incremental simulation building [58]. In some cases, the target analysis is not concerned with how behaviors (processes) are executed and does not assume resources can control the processes they execute, treating resources as inputs to their processes.

For each `Process` that an `Active Resource` *canExecute* (its capability), it has an *expected capacity for that capability* defined as the expected number of times a `Process` can be executed during some length of time. It is more difficult to estimate the capacity of resources that have multiple capabilities, i.e. can perform multiple kinds of processes. For a set of capabilities, the `Active Resource` has an expected *capacity region*. In multi-dimensional newsvendor formulations, the capacity region is defined as the region of feasible combinations of products (or activities) that can be created (executed) given a level of resources [59, 60].

DELS are Active Resources

DELS are networks of interconnected resources, specifically equipment and other DELS. This is achieved by modeling `DELS` as specialized `Active Resources`, which are specialized `Flow Networks` and `Resources` (figure 9). Since `Resources` are composed of *memberResources* (typed by `Resource`), DELS can be composed into self-similar systems where the parent DELS control their child DELS uniformly [17], i.e. requesting and allocating capacity (availability) for a particular capability.

`Active Resource` is specialized into `DELS` and `Equipment`. The main distinction between these is how they control execution (and advertisement) of their capabilities, specifically controller capabilities. Equipment behaviors typically are controlled by a `Realtime Controller` that executes simple, real-time, deterministic logic, typically embodied in a PLC. In contrast, DELS have more flexibility in their decision-making, embodied in operations management software control (`Operational Controller`), described in section 3.8.3. From the operations viewpoint, equipment can be characterized by the inability to refuse work or do tasks out of order, and preemption and sequencing decisions are handled by the operations controller. From this perspective, equipment behaviors are *invoked*, where DELS behaviors are *requested*.

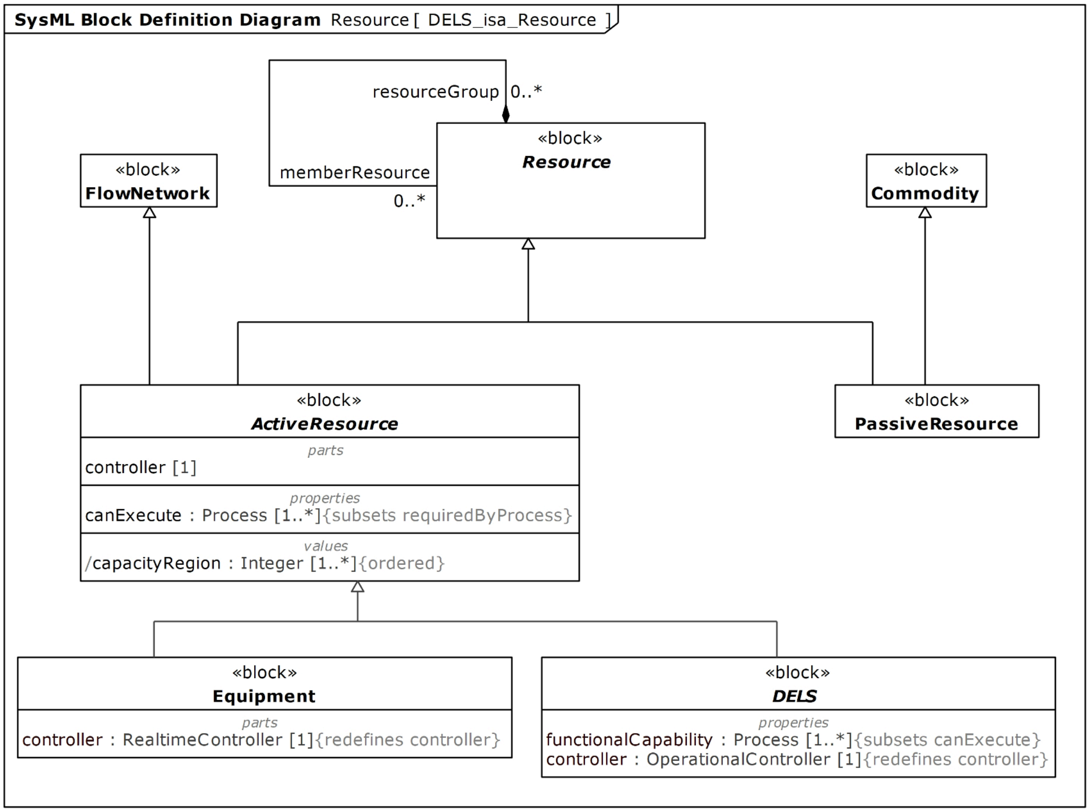

Fig. 9. DELS are specialized from Active Resources, capable of executing Processes. Their controllers are operational. Equipment are the other branch, with realtime controllers, such as PLCs.

3.1.2 Availability: Capacitated vs Discrete-State

Resource availability is concerned with assigning work to particular resources. It distinguishes between resources that must be in a particular *state* to be assigned a particular task (`discrete state`), e.g., a particular set-up or location to execute a particular process; while other resources are pooled with a finite, countable quantity available that can be assigned to tasks (*capacitated*), e.g., if the required number of resources is available in the pool, then they can be assigned. The model library reflects this distinction by specializing `Resources` into `CapacitatedResources` and `DiscreteStateResources` (figure 10).

`Capacitated Resources` (or rather the pool they are contained in) have a *capacityMeasure* and *currentCapacity* to track how much of its capacity can be allocated to work. It defines operations to *allocateCapacity()* and *deallocateCapacity()* (remove and return a unit to the pool, respectively) and operations to *increaseCapacity()* or *decreaseCapacity()*, which actually might be referring to putting more objects in the pool or increasing the

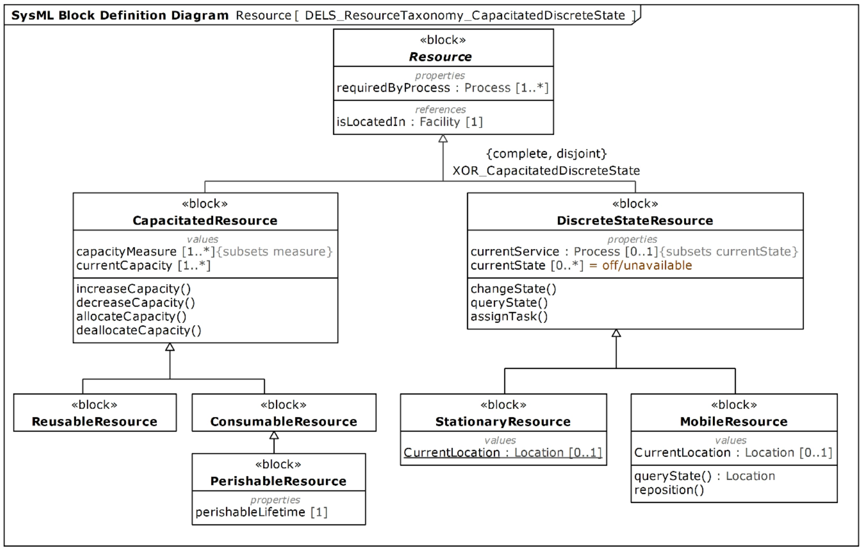

Fig. 10. Capacitated and Discrete-state Resources specify how work can be allocated to a resource.

capacity measure.

Additional specializations of `Capacitated` and `Discrete State Resources` include (figure 10):

- `Reusable Resources` can be involved in more than one process execution (sequentially). After one process using them is completed, they are returned to their pool, or made available again.

- `Consumable Resources` can be involved in no more than one process execution, because they are "used up" during processing.

- `Perishable Resources` can have *capacityMeasures* that degrade (decrease) over time until they are not longer usable (its *perishableLifetime*). Other resources can degrade over time, but usually not simply because of the passage of time; for example, tool wear is based on it usage in processing.

- `Stationary Resources` have constant location states.

- `Mobile Resources` location states are not necessarily constant, changed by *reposition*(), a specialized kind of changeState() operation.

A `Discrete State Resource` behavior can be modeled by specifying its *classifier behavior* using a state-machine (figure 11). These can be extended to incorporate additional behaviors that affect resource availability, such as failure states and transitions. Buzacott et al. [61] classify interruptions as run-based (interruptions are a function of job arrivals) or time-based. Wu et al. [62] classify queueing models for workstations with interruptions by augmenting run-based vs time-based failure events with preemptive vs non-preemptive behaviors. It also refines run-based, non-preemptive interruptions into state-induced (e.g., a warm-up after being idle) or product-induced (e.g. set-up machine) interruptions.

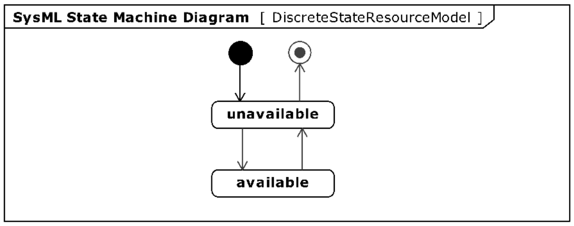

Fig. 11. A simple state machine to start defining a discrete state resource's classifier behavior.

Separating resources by how their availability is modeled is common in analysis models, though many terms other than discrete state and capacitated are used. Hackman et al. [63] classifies process inputs and outputs into products or materials and non-storable services, such as labor and machine time (discrete state). These classes may be mapped to capacitated (possibly consumable) and discrete state resources, respectively. [64] examine capacity allocation decisions for 'make-to-stock' manufacturing firms that allocate available inventory and 'make-to-order' manufacturing firms that essentially hold production capacity "in stock" by idling discrete state resources. However, when coping with demands in excess of capacity, both 'make-to-stock' and 'make-to-order' firms formulate nearly identical analysis models to allocate available capacity to customers with varying priority levels. Newsvendor Network models use the terms stock and resources [60]. There are also methods for approximating discrete-state resources as capacitated ones (e.g. machine X has 8 hours of capacity per day) [65]. These models may give some additional insight into constructing more precise behaviors models for these kinds of resources.

3.1.3 Organization: Atomic vs Aggregate Resources

Processes often require multiple resources other than a machine, such as fixtures, auxiliary tools, input materials, sub-components, an operator, etc. `Aggregate resources` are composed of multiple `resources`, sometimes enabling them to execute a limited number of processes simultaneously. [51] define `primitive resources` as supporting one process at a time (indivisible), with a fixed set of attribute types and predefined behavior. Their framework forms `composite` (or compound) `resources` by joining two or more resources (potentially different types) to "create" a resource with more valuable capabilities than the individual ones.

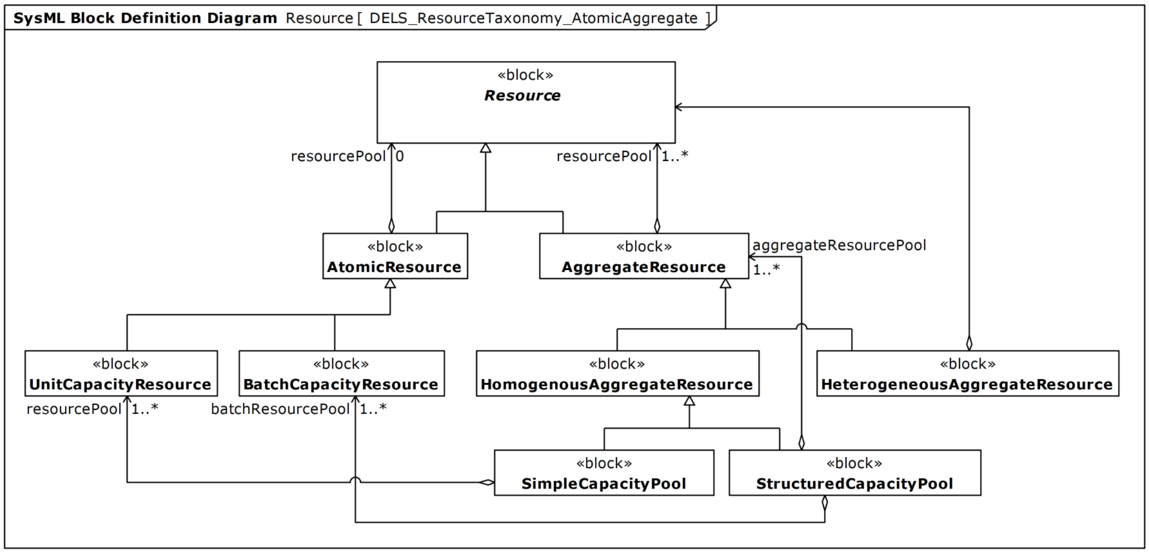

Fig. 12. Aggregate and Atomic Resources specify how resources are combined to form resources with different (greater) capability than its components.

The other kinds of resources described in OZONE [34] include: `Atomic Resource`, `Unit Capacity`, `Batch Capacity`, `Aggregate Resource`, `Homogeneous Aggregate`, `Simple Capacity Pool`, `Structured Capacity Pool`, and `Heterogeneous Aggregate`. More rigorous definitions of these resource types are deferred to future revisions.

3.2 Active Resource Relationships

Networks can be used to model coordination between multiple `Active Resources` by specializing them from `Flow Network` (figure 13). Active resources participate in two kinds of relationships: one for modeling resource groups with advanced capabilities greater than the capability of the individuals, for example, more complex processes or ones requiring coordinating simultaneous execution by multiple resources. This kind of relationship is modeled by *relationshipBetween* typed by `Active Resource Relationship`. In some modeling frameworks, the coordinating resources are modeled as a new temporary active resource, a resource federation [66]. The whole-part composition relationship inherited from `Resource` can be used to model the relationship between the new active resource (the *resource group* or federation) and its *member resources*. This modeling approach can also be used to model long-term or permanent resources groups as well, see for example, the *parent-child DELS* relationship in figure 13.

The second kind of relationship models flows between active resources by reusing *flowEdges* (typed by `Flow Network Link`) inherited directly from `Flow Network`. `Flow Network Links` between active resources, including both equipment and other DELS, can be further specialized into `Material Handling Channels` that require using resources with *move* capabilities to facilitate flow across the *flow edge*. `Material Handling Channel` is a special kind of part-part relationship between `Active Resources` specialized from `Flow Network Link`. `Material Handling Channels` are parts of `DELS` typing connectors between its `equipment` or other `DELS` (figure 9). As a kind of flow edge, analyses of active resource networks can be constructed using both active flows using material handling edges or more abstract passive flows using only flow edges, which do not specify the flowing mechanism in detail.

`Active Resource Relationships` are a placeholder to capture necessary attributes modeling collaboration and coordination between active resources. For example, `Active Resource Relationship` may be specialized to capture relationships governed by smart contracts[2] (figure 13), contract net [67], orchestration schemes [68], among other options.

ISA-95 Resources

The ANSI/ISA-95 (IEC 62264) [1] specification includes specialized resource classes for material, equipment, and personnel (figure 14). These specialized resources reduce the gap between the abstract resource types developed in OZONE [34] and this report and more concrete model libraries, such as m-SysML [69]. These specialized resources classes also create a classification of processes by the types off resources required to execute the process (see figure 17 in section 3.3).

While the standard does not specify behavior of the specialized resources beyond colloquial meaning, they can be mapped to (via generalization) the `Resource` role classes defined in section 3.1.2 (figure 10). For example, `Material` is generalized by `Consumable Resource` (a kind of `Capacitated Resource`) and `Personnel` by `Discrete State Resources`. `Equipment` could be generalized by either `Discrete State Resource`

---

[2] https://doveltech.com/innovation/what-belongs-in-a-service-contract/

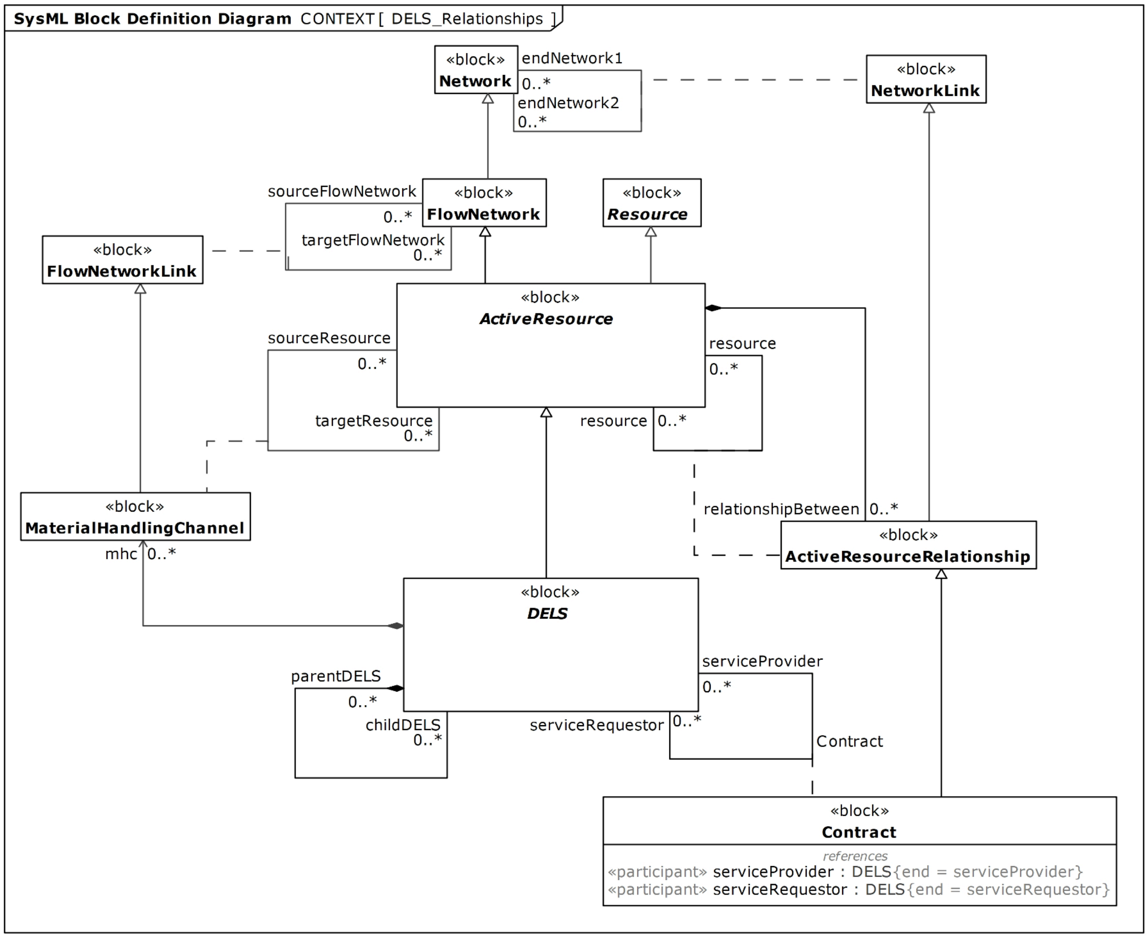

Fig. 13. DELS have contract connectors and material handling connectors.

or `Capacitated` depending on how the controller manages its availability. For example, an single, identifiable fixture for a specific part would be treated as a `Discrete State Resource`, but a pool of interchangeable fixtures would be treated as `Reusable Resources` (a kind of `Capacitated Resource`). New resource classes specialized from `Equipment` could specify corresponding equipment state machine model (figure 15) using any one of several machine information standards, such as MTConnect (ANSI/MTC1.5-2019) [70], PACKML (ISA-88) [71], computer-aid manufacturing XML (CAMX) (IPC-2501) [72], equipment behavior catalogue (EBC) (ISO 16400) [73], etc.

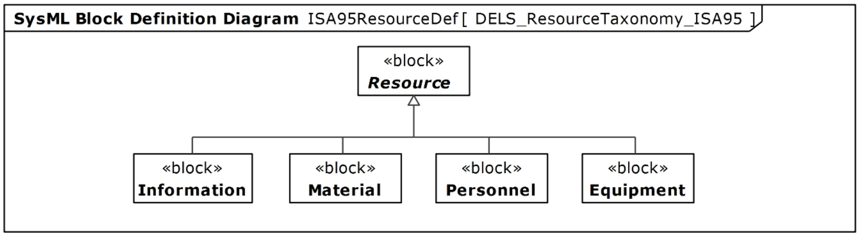

Fig. 14. The ANSI/ISA-95 (IEC 62264) [1] specification includes specialized resource classes for material, equipment, and personnel.

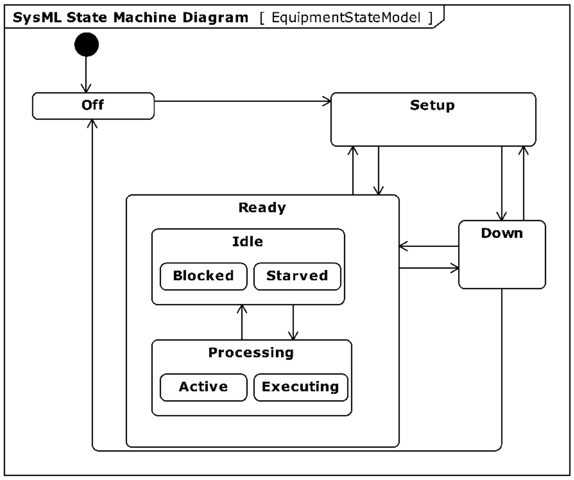

Fig. 15. Equipment state model from CAMX can be a starting point to define equipment's classifier behavior.

3.3 Process

The DELS `Process` definition is specialized from `Process Network` to specify a production or logistics transformation (figure 16). This approach keeps the network abstractions (section 2) self-contained, abstractly focused on commodity flows and queueing network analyses. It also does not clutter the abstraction with DELS concepts, such as product and resource flows (specialized commodities).

OZONE defines an equivalent modeling construct to process, as: "*Operations* are used to represent different actions taken during a production or transportation process. Generally speaking, an operation is a specification of the set of constraints that define a particular activity (e.g. resource requirements, duration constraints, temporal relation relative to other activities, etc.) Since operations relate to each other through *temporal relations* which specify the temporal and causal ordering of operations, they allow the formation of operation graphs (networks or sequences of operations). Operations can also be organized hierarchically to describe transportation processes at different levels of details." [74]

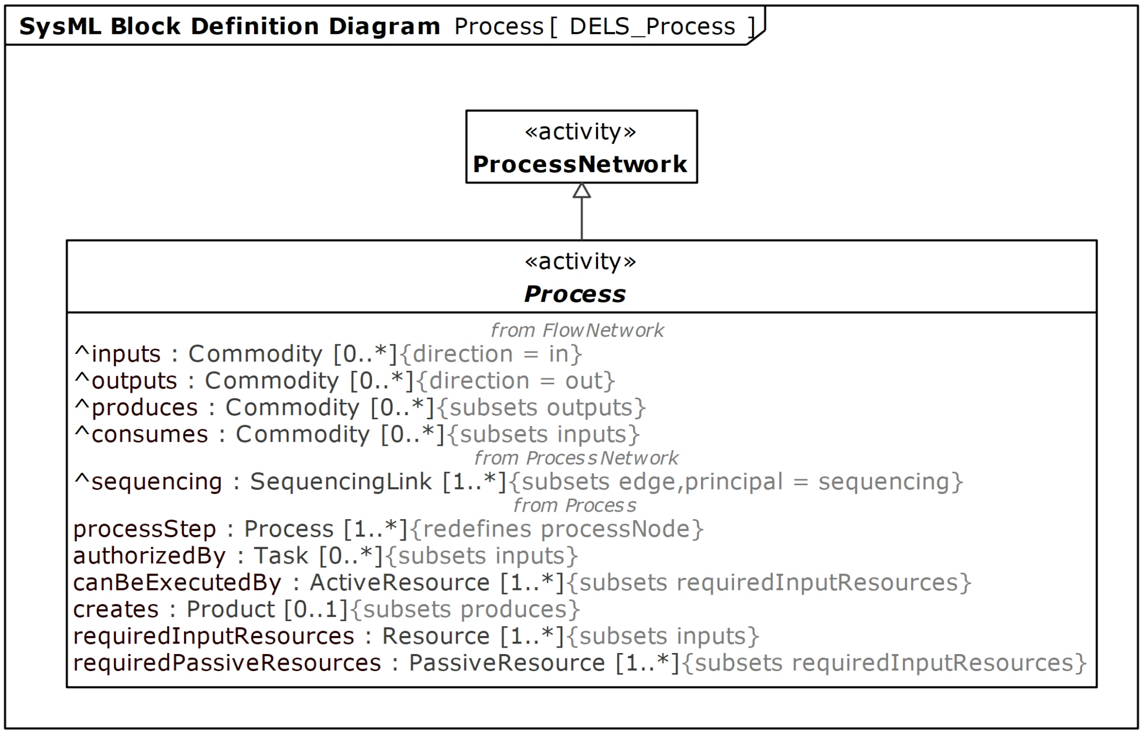

Fig. 16. DELS Process is specialized from CFN's Process Network.

In the DELS `Process` definition, *inputs* (typed by `Commodity`) required to execute `Process` are specialized (subset) into *requiredInputResources* (typed by `Resource`). *requiredInputResources* can be further specialized into *requiredPassiveResources* (typed by `Passive Resource` and resources that the process *canBeExecutedBy* (typed by `Active Resource`. See section 3.1.1 for more discussion on these kinds of resource. Additionally, executing the process often needs to be *authorizedBy* a task (discussed in section 3.6), which also subsets the *inputs* to the process. Finally, the `Product` that the `Process` *creates* is a subset of things that the `Process` *produces* (itself a subset of the *outputs*).

There are two aspects to describing processes: kinds of process steps (processes) and how to compose them into larger process plans.

Kinds of Process Steps DELS `Processes` are organized into two (orthogonal) branches (figure 17). The first organizes processes by function: changing fit, form, and function (`Make`); age (`Store`), location (`Move`), flow (`Control`), or verification (`MeasureTest`) of commodities. The second branch organizes processes by the types of resources (see section 3.1) required to execute the process (see IEC 62264-1 [1]). The base `Process` has an option (denoted by [0..*] multiplicity) for `Material`, `Personnel`, `Equipment` resources, and has several specializations: `Semi-Automated Processes` require material, personnel, and equipment; `Manual Processes` do not require equipment (denoted by the [0] multiplicity); `Non-material Processes` do not require material; and `Automated Processes` do not require personnel.

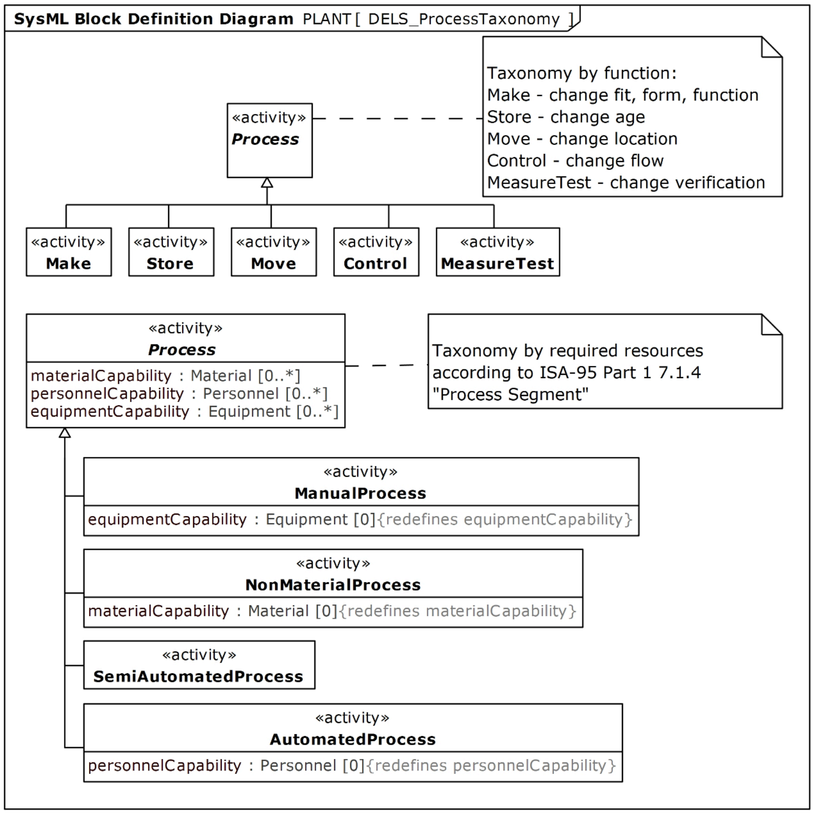

Fig. 17. DELS Process is elaborated with taxonomies of specialized transformations.

Organization of Process Steps Process plans organize the execution of processes in DELS using precedence or sequencing relations (typed by `Sequencing Links`. *ProcessPlan* redefines the *parentProcessNetwork* role in the whole-part relationship (composition association) between `Processes` and their finer-grained *process steps*. Process plans define a sequence of functional transformations (*processSteps* typed by `Process`), the inputs/outputs from each transformation (parameter nodes), and pre/post-conditions on the transformed object. Process plans link functional capabilities provided by resources (modeled as Processes) and required capabilities of `Products`(and Services).

Planning and scheduling models based on the disjunctive graph formulation are generally attributed to [75]. Disjunctive graphs have been used in job-shop scheduling problems because of their ability to capture processing alternatives in multi-processor environments [76–79]. AND/OR digraphs extend the disjunctive graph semantics by defining alternative task and sequence requirements using OR junctions to represent alternative paths and AND junctions for parallel paths without specifying a particular execution sequence, see, e.g. [33, 80, 81]. Applications with complex scheduling requirements have applied AND/OR digraphs to manage the complexity of representing alternative processing sequences [82]. AND/OR digraphs exhibit several important advantages for representing process plans [33]. First, each node can nest its own digraph decomposing the process into smaller processing steps. Second, they present a process to produce a serialized process list from the digraph, which is their definition of the planning and scheduling problem. Third, they capture the duality of a Product traversing its process plan as a control graph that formalizes the processing requirements of all the tasks to be processed by a controller.

The manufacturing literature defines process plan formalisms for planning and scheduling that extend the required capabilities of process plan representations, including "explicit parallel and alternate sequences, multi-job synchronization, hierarchical task decomposition, resource management primitives, and user extensibility" [83]. Formal languages such as the *Process Specification Language* (PSL) (ISO 18629) [30, 84] or *A Language for Process Specification* (ALPS) [81] may be used to specify process plans in the DELS domain.

In the DELS modeling framework, SysML activities are used to specify process plans. Each *processStep* is specified as a callOperation or callBehavior action. The Method/Behavior is a `Process` and the target object of the call is an `Active Resource`. The instance values of the **canExecute** relationship between `ActiveResources` and `Process` define a sort of "reverse dispatch table" (runtime polymorphism). That is, when the system asks who can execute this behavior (`Process`), it uses the table of valid resource/process assignments to figure out which active resource object to invoke the behavior on (or assign the execution).

3.4 Product

In manufacturing systems, products are defined by a bill of material (BOM) and a process plan, i.e., transforming (which could be just assembling) this list of materials per this process plan will result in the desired product. In warehousing, products can be defined similarly as a pick list and a process plan specifying a route to and from the required storage locations. However in transportation logistics, products are inputs and their geographic location is transformed (a service). Similarly by storing products (or any objects), their age is transformed. The common idea across all of these system descriptions is that products are flowing through and being transformed by the system.

OZONE defines product with the similar goal of unifying systems producing physical outputs and others providing services: "*Products* represent knowledge about how to turn demands into operation graphs. In the manufacturing domain the definition of the term *product* is clear: products are descriptions of the objects produced by the manufacturing systems. In the transportation domain, however, a 'product' is a collection of information about how to move 'packages' from one place to another, i.e., products are general descriptions of *missions*." [74].

The `Process` and its steps (process plan) specify *requiredInputResources* — equipment, raw materials, operators, and information — to create a `Product` (see section 3.1). The *billOfMaterial*, on the other hand, is part of the `Product` description. Since material is a specialized resource, the materials in a BOM are a subset of the *requiredInputResources* for creating the product. There are other resources required to produce a product that are not included in the bill of material.

Much like balancing commodity consumption and production in `Flow Networks` (section 2.2), DELS require balance between what is consumed by a DELS (its *inputResources*) and what is produced by each DELS upstream of it (their *outputs* or *outputProducts*). However, moving away from generic commodity to domain-specific and scope-specific terminology such as input material, intermediate products, parts, sub-components, etc; it becomes difficult to reconcile type/quantity balance. Here we follow the ISA-95 convention where parts, sub-components, intermediate products, etc. are all specialized `Material`, emphasizing material flow/handling and consumption of materials to produce products (input/output). A role-based modeling approach defines each term as a role type, reclassifying objects depending on the context.

The product taxonomy has two layers (figure 18): one distinguishing aggregated versus assembled products, and a second that further refines aggregated products into homogeneous and heterogeneous aggregated products. From an analysis perspective, these layers help tracking objects before and after they are input into a product; for example, assembled components are typically expected to be only referred to by the type of assembly, while

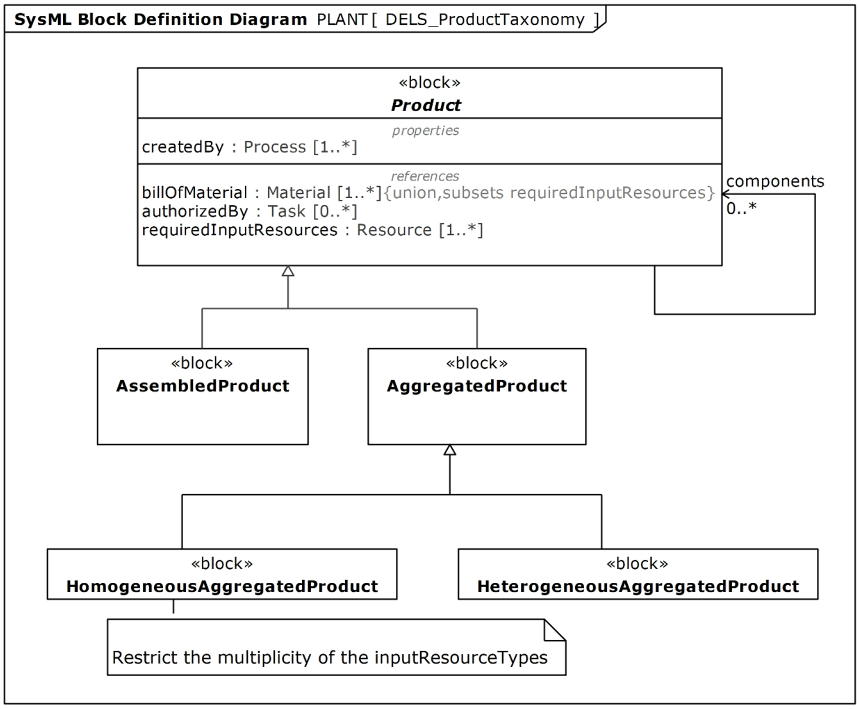

Fig. 18. DELS Product is specialized to capture the composition and handling of the product.

aggregated components would be regarded as a bundle of individual commodities.

The first layer of classification is about how the product is constructed from input components. `Aggregated Products` are defined as `Products` that can be reverted to their original components. For example when warehouses aggregate commodities (typed by stock keeping unit (SKU)) into shipments, these commodities are viewed as *inputResources* into the `PACK`() and `SHIP`() processes producing the shipment. This shipment (`Aggregated Product`) can be taken apart in the future and each input commodity should retain its product identity (defined by its SKU). However, `Assembled Products` are single artifacts that cannot be disassembled into their input components. While dis-assembly processes can separate target object into its components, these components are generally not regarded as identical to the inputs in their fit, form, and function.

`Assembled Products` typically are composed of many kinds of input resources (heterogeneous), while in `Aggregated Products` the bill of material is often not heterogeneous. This is reflected in figure 18 as a specialization layer distinguishing `Heterogeneous` and `Homogeneous Aggregated Products`. For example, a shipment from a warehouse is the aggregation of a (not necessarily homogeneous) set of SKUs (product type). Full pallets are modeled as `Homogeneous Aggregated Products` while mixed pallets are

`Heterogeneous Aggregated Products`. In this setting, the distinction usually guides which make, move, and store behaviors handle the products.

Product Definition Standards Computer aided engineering methods and technologies for capturing product specifications, such as product data management (PDM) and product lifecycle management (PLM), are more mature and integrated into manufacturing engineering methods than in other fields. Building on the ISO 10303 [85] and IEC 62264 [86] standards, product ontologies formalize technical data and concepts associated with products [87–89].

3.5 Facility

`Facility` describes the geometric characteristics of physical DELS artifacts, including `Layout` and `Placement` of its *containedResources* and spatial relationships between those resource objects (figure 19). `Resources` have an inverse role of *isLocatedIn*, which `DELS` inherits.

Industrial engineering methods have long used similar facility models and analysis methods to analyze both physical buildings, such as factories, as well as geographically distributed components, such as supply chains. For example, [90] defined the facility layout problem as configuring the facility to minimize cost of transporting materials between between components. [91] and [92] provide overviews of the facility layout and facility location problems, respectively. This definition does not require the DELS to own the facility (or `Physical Space`) that it operates in, enabling modeling of material handling systems, transportation systems, and supply chains.

Material handling systems require layout information to execute their function. The message-based part state graph (MPSG) formalism specifies addressable locations, physical locations to which a material handling device has access to pick objects up or put objects down, and uses the network of addressable locations to create sequences of material handling process steps [93]. In Core Manufacturing Simulation Data (CMSD), layout information defines spatially-oriented characteristics, including location, footprint, and orientation of each resource within a facility; and interrelationships for logical and physical entities carrying out production activities [94]. m-SysML specifies an extensive layout and geometry model [69]. Other standards such as The Open Geospatial Consortium (OGC) IndoorGML [95] and Building Information Model (BIM) [96, 97] are useful for capturing the facility description.

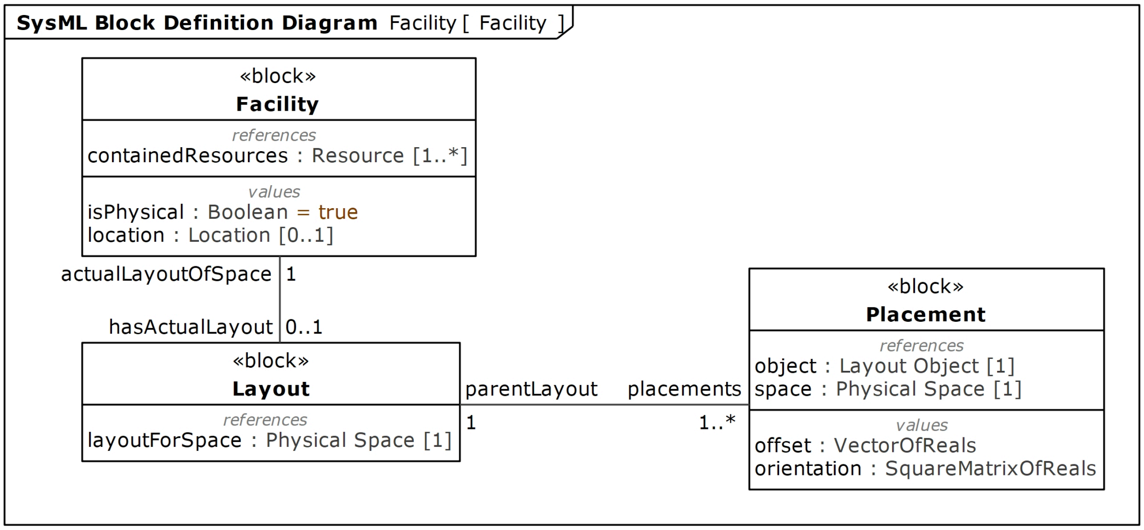

Fig. 19. DELS Facility describes the geometric characteristics of physical DELS artifacts, including size and layout of resources and spatial relationships between resource objects within a DELS.

3.6 Task

`Tasks` authorize `Process` execution. They cover traditional orders for products and orders for services or logistical processes, such as transportation, storage, and testing / quality / verification. A uniform description of tasks enables planning and scheduling of plant-level production orders matching customer demands to work authorizations, as well as machine-level machining activities (invoking or authorizing automation tasks).

`Task` bridges two distinct but complementary views of "work". First, is the automation (computational) view focusing on function/process execution with initial and goal states [98, 99]. `Task` is defined by [99] as "a problem assigned to an agent, where a problem is defined as an initial state, goal states, and failure states". In the distributed decision-making literature, tasks are decomposeable into subtasks that can be assigned or contracted to other systems or agents [98–100]. This *execute function* view is similar to how manufacturing roughly defines jobs, orders, and operations. Specialized `Tasks`, such as production orders, work orders, jobs, etc., authorize the execution of a specialized process *Make*(`Product`). Customer orders (also a kind of `Task`) authorize a *Deliver()* process execution. Then depending on the customer order decoupling point, the *Deliver()* process might trigger one of several kinds of *Make()* process: engineer to order, purchase to order, make to order, assemble to order, or deliver from stock (make to stock) [101].

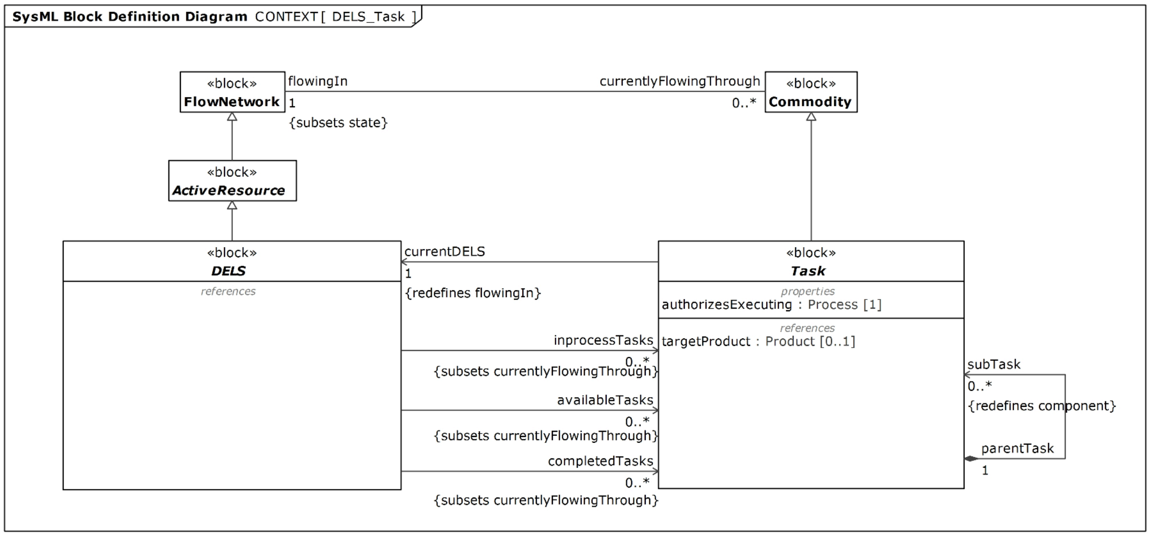

Fig. 20. Tasks, a kind of commodity flowing through DELS, authorize DELS to execute a Process.

The second view of work is "jobs flowing around a factory," often including the required input and auxiliary resources, such as the workpiece to be operated on, fixtures, and raw materials, etc. `Task` is specialized from `Commodity` to enable them to flow through networks of resources (DELS). *Order Holons* in PROSA [102] represent tasks in a manufacturing system and the cited paper includes example taxonomies and system models. OZONE defines *Demands* that "specify requests for specific quantities of products or services to be produced/undertaken within specific time constraints, as well as client-dependent priority information. In other words, demands are used for representing customer orders, move requirements, and other external demands to the scheduling system." [74].

`Tasks` often consist of both physical and informational pieces. The physical part of a task, consisting of a workpiece, kits, routing sheets, etc.; is directed to the plant. It is stored in an input queue, physically operated on by equipment, and requires material handling to flow through the system. The information component of a task is directed to a controller, providing instructions (and authorization) on how to execute the required process. Sometimes information components may have both physical and digital representations, such as physical workorder or routing sheets.

`Tasks` play several roles in `DELS`, which are often dependent on the state of the task (figure 20). One role is *availableTasks*, which are tasks that have been accepted, admitted, and are waiting in the *availableTaskQueue* to be serviced. *completedTasks* have been serviced and are stored in an *completedTaskQueue* waiting to depart the system. *inProcessTasks* are currently being served by the system and located in/at some *memberResource* (usually equipment).

`Tasks` may be decomposed into *subtasks* authorizing a `Process`'s *processSteps*. The decomposition associates a new *subtask* with each *processStep* in the parent *processPlan*. These subtasks usually follow the *sequencing* from the *processPlan* (typed by `Process`).

Consistent methods (and representations) for decomposing tasks are important for creating self-similar and uniform controller architectures where resource clusters can be dynamically formed to address a particular task, or in agent-based systems where "[the] agents can subcontract tasks to other agents, a process that involves breaking a task in a number of sub-tasks handled by different agents, or clustering a number of tasks into a super-task" [100].

3.7 Interface

`DELS` defines interfaces for handling flows of tasks and resources (figure 21). It has four ports enabling flow of tasks and resources in and out of the system. In SysML, ports expose components (parts) of the system, defining an interaction point with other systems. The «proxy» port stereotype on the composition association is an equivalent representation to the graphical white box on the edge of the block; see, for example, *incomingTasks* in figure 21.

The *incomingTasks* port is typed by an (abstract) interface block `inDELSTask`. It defines operations (*receiveTask*()) to be implemented by system components that move tasks (defined by the flow property) into the system. Inversely, `outDELSTask` defines properties and operations that move tasks out of the system.

The resource input and output interfaces (typed `inDELSResource` and `outDELSResource`, respectively) define operations (*receiveResource*() and *outputResource*(), respectively) to be implemented by system components that move resources (defined by the flow property) into and out of the system. These ports can be specialized to accommodate different kinds of resources, including raw materials, equipment, and parts/products. Parts and products are modeled as a type of material resource, see section 3.9 for more discussion.

The input and output interfaces are defined by ports typed by abstract interface blocks giving the modeler wide latitude to select system components to implement the interface. For example, a modeler may allocate the same system component to implement both resource and task interfaces, or both to handle both input and output of a kind of resource.

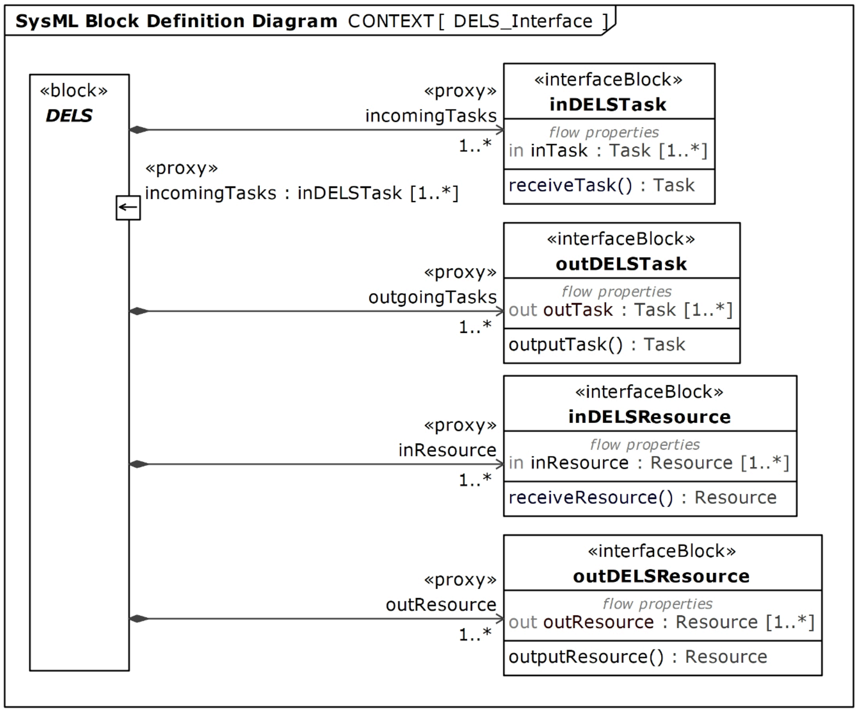

Fig. 21. DELS interface defines ports for handling the flow of tasks and resources across its boundary.

On the other hand, it may be necessary to provide separate system components to handle information and physical components separately.

3.8 Operational Control

The operations management layer of the ISA-95 hierarchy [86], broadly speaking, has the functional responsibility to match, and execute the matching, the capabilities provided by the system's resources to the capabilities required by requested products or customer demands. Operational control executes the matching by controlling material and resource flows through the system. That is, control of production capabilities and capacities is largely executed by supporting logistics functions, including inventory management and material handling. This control activity is generically defined as "scheduling". This section describes scheduling, not as a single monolithic activity or decision, but rather the coordination of several decisions and system actuators.

Modeling operational control is less mature, and potentially more difficult, than other aspects of the system. Operational control is built on top of the system specification (the plant) and implemented using a mix of existing system resources and dedicated resources.

For example, logistics and material handling resources are often allocated to dedicated systems but are interwoven into the production environment. This makes it difficult to clearly define control behaviors and allocate them to system resources. Further work is required to demonstrate how to apply the model library elements described in this section to model domain specific applications.

To provide the proper context for modeling operational control without elaborating a complete plant-controller architecture, consider the following mental model: there exists a controller that interacts [sense and actuate] with the base system (or plant) (figure 22). The controller consists of a decision-maker and decision support. The decision-maker observes the state of system and responds by querying the decision support with a question regarding actions that can be taken to effect changes in the base system. The decision-maker then uses the answer provided by the decision support to select an action to be executed by an actuator in the base system. An abbreviated sketch of this controller architecture can be found in [103] and a longer discussion in [5].

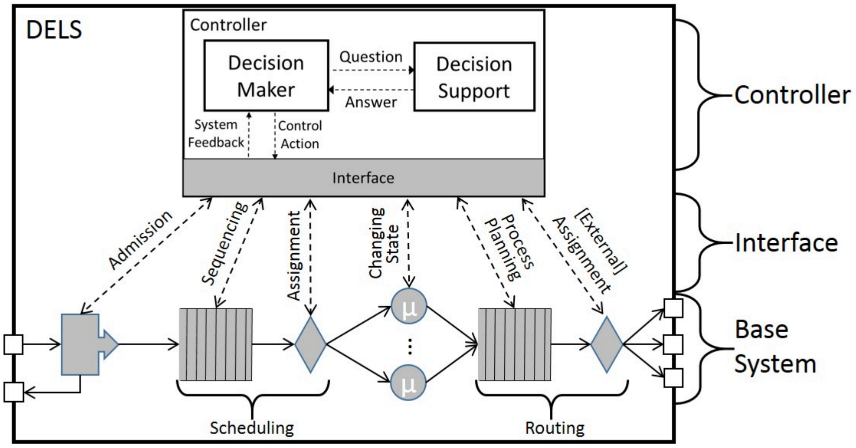

Fig. 22. A canonical set of control questions defines a comprehensive functional specification of all decision-making mechanisms that a controller needs to provide in order to manage the behavior of the system.

Control actions are derived from answers to control questions, and this model formalizes five kinds of questions (control functions) described in [104]. These control questions identify the functional control mechanisms (control actions) required to manipulate the flow of tasks and resources through the system (figure 22). These questions are:

1. “should a task be served?” (*admission*)
2. “when should the task be serviced?” (*sequencing*)

3. "by which resource(s)?" (*assignment*)
4. "what process step does the task require next?" (*dynamic process planning*)
5. "in which state does a resource need to be to service a task?" (*change-state*)

Scheduling and *Routing* are modeled as joint control functions, combining *sequencing* with *assignment* and *process planning* with *assignment* decisions, respectively.

The control questions provide an informal classification scheme and foundation to construct the model. Section 3.8.1 presents the interfaces for decision support for each control function. Section 3.8.2 presents the control processes and actuators in the plant to execute operational control. Finally, section 3.8.3 provides an overview the DELS operational controller, which is largely still a work in progress.

3.8.1 Operational Controller Decision Support

Each control function has an associated decision support class that helps the controller make decisions. The decision support for each control question is encapsulated in an abstract strategy class that defines an operation with a signature derived from the decision functions defined in [104] (figure 23).

Decision support algorithms are required to implement the signature and the decision function. Each control algorithm is responsible for formulating an appropriate analysis model, solving the analysis model, and translating the output into an actionable recommendation. This actionable recommendation output by the decision support is passed to an `Actuator` in the plant that executes the choice (section 3.8.2). Reusable, standard decision support classes allows the controller to access the decision support algorithms through a consistent interface, enabling progress towards interoperable decision support algorithms for DELS.

3.8.2 Operational Control (Plant) Model Library

Each control function has an associated structural element in the base system, an `Actuator` specialized from `ActiveResource`, that is responsible for executing the controller's choices. The `Actuator` also has a behavioral element `Control Process` (figure 24). Each `Actuator` is related to its corresponding `Control Process` through the **canExecute** relationship. System specifications provide details on how the `Actuator` and `Control Process` are implemented by specializing concrete system resources and providing them with methods to implement the control function.

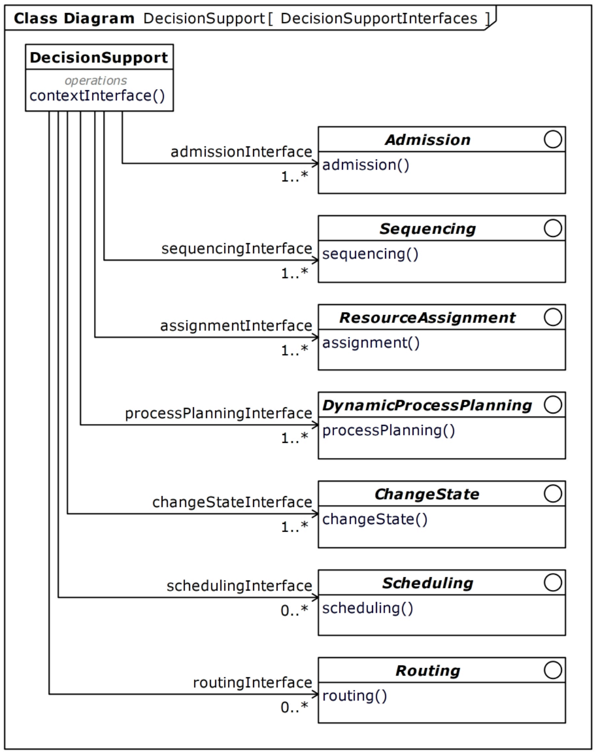

Fig. 23. Each control decision has a corresponding interface that defines an operation with a standard signature.

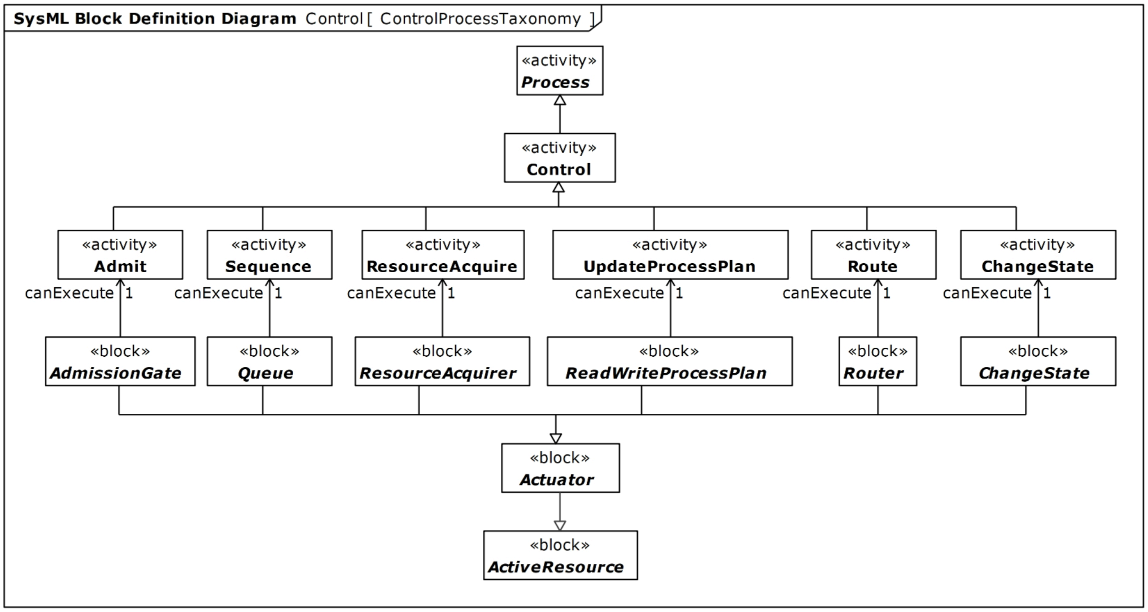

Fig. 24. Each control function has both an Actuator (specialized Resource) and a Actuator behavior (specialized Process)

3.8.3 Operational Controller

The `DELS Operational Controller` is responsible for implementing data collection and management functions, operational decision making and executing, and communication and coordination with with other controllers in the system. Conceptual architectures for DELS operations controllers are discussed in [5, 103]. This is an area of on-going research, in particular focused on control and controller architectures.

The stylized conceptual diagram of the controller depicts several required components: decision-making composed of monitoring and execution; decision-support composed of formulation, optimization, and implementation (top of figure 25). The current state of implementation is shown in the class diagram at the bottom of figure 25.

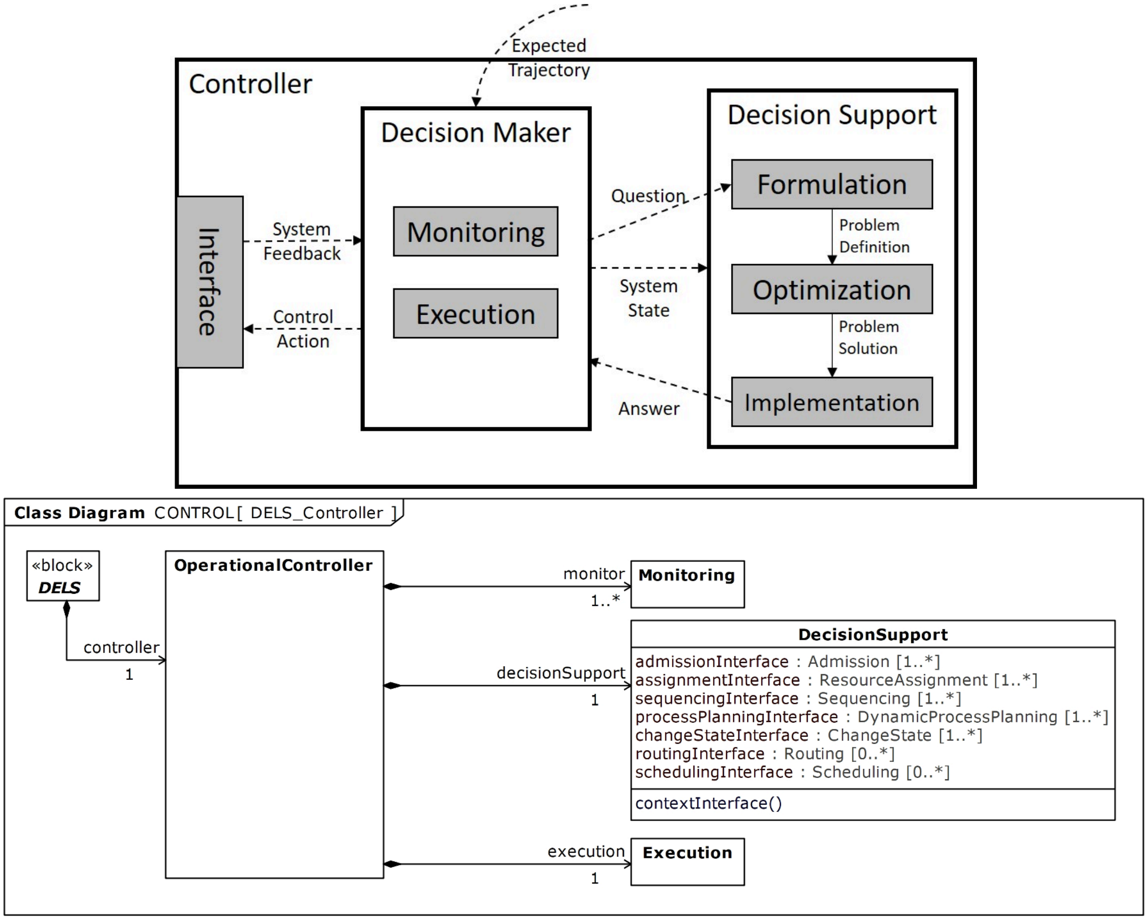

Fig. 25. The DELS Operational Controller consists of decision-making and decision support components.

The **Decision Maker** component maintains a representation of the system state using feedback collected from Monitoring. The **Decision Support**. The decision support module for each control question must be capable of formulating the analysis model from the system state (create a problem definition), must be able to solve the problem, and then must implement the problem solution; that is, reframe the analysis results in the context of the original question, providing a actionable answer to the decision maker. Given a standard decision support interface, the formulation, optimization, and implementation are tightly coupled to the solution method and are implemented together as part of creating the specialized decision support classes discussed in section 3.8.1.

The operational control model described in this section clearly separates the `Actuator`, actuator's behavior (`Control Process`), and `Decision Support`. This separation is common in other engineering disciplines and the goal here is to support practitioners in extracting the correct knowledge to explain how their system works and to develop implementable specifications. Well-defined, machine-readable operational control specifications can be connected to analysis models supporting optimization or validation and verification.

3.9 Overview of Extended DELS Definition

DELS are defined by their products, process, resources, and facility; the tasks that define requests for these products and processes, and an operational controller to control the flow of resources and execution of processes. This section summarizes the DELS models, tying together the different components and views described in the past few sections (figure 26). Section 3.9.1 then describes how the DELS model can be extended to create domain-specific production and logistics models.

`DELS` and `Equipment` are mapped to `Active Resource`, where the distinguishing factor is based on autonomy and operational control behaviors; that is, can the resource decide to not do something. This approach defines DELS as a natural extension of traditional Product, Process, and Resource (PPR) ontologies. DELS inherit flow properties modeling the flow of resources in (*inputResources*) and the flow of products out (*produces*). In addition to the input of passive resources, DELS themselves are composed of *member Active Resources*, some of which may be other DELS, its *child DELS*.

`Product` references its *bill of materials*. Following the OZONE/MANDATE model, `Product` is defined as a kind of `Material` allowing products to be easily incorporated into another product's bill of material. Additionally, `Material` is a `Passive Resource` allowing it to flow and participate in (be consumed by) `Process` executions, but not execute processes. Modeling `Product` as specialized `Material` allows the product to flow

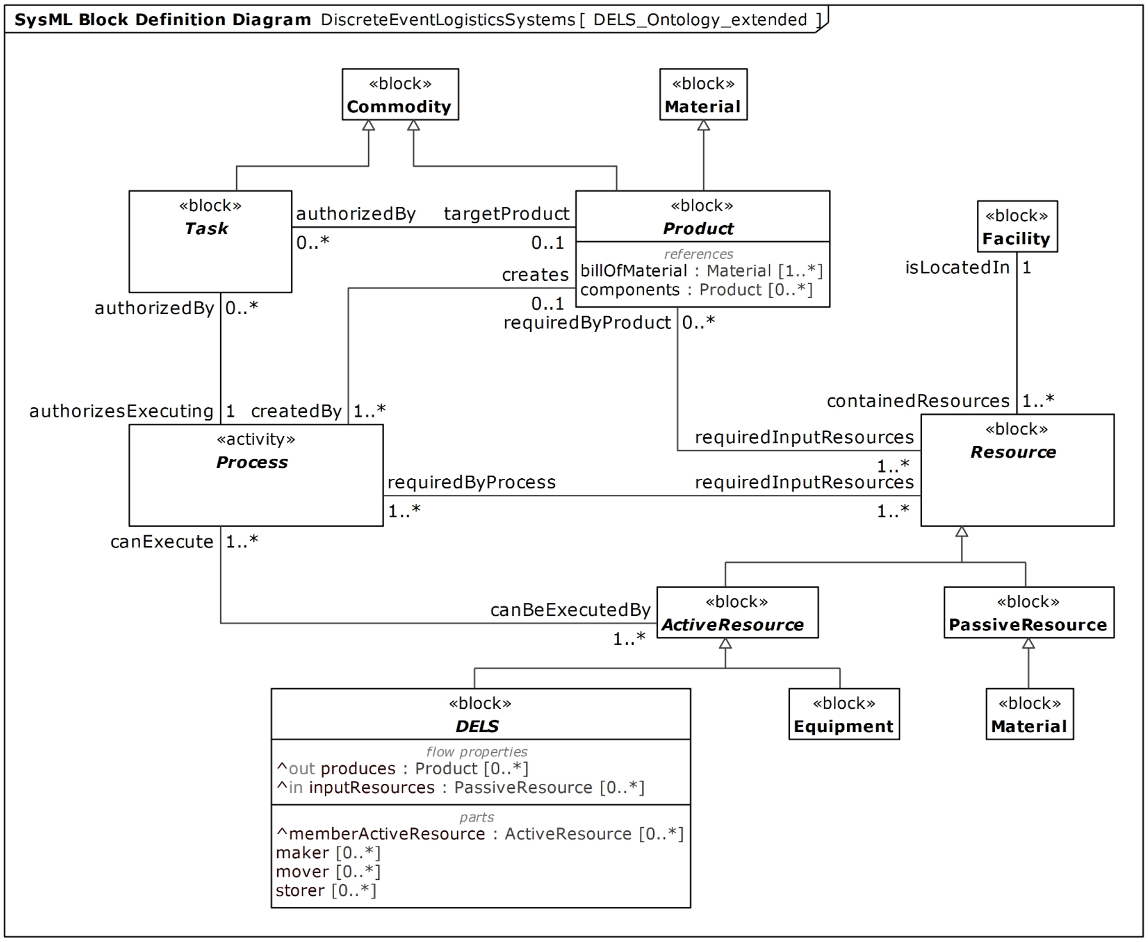

Fig. 26. DELS are defined by their products, process, resources, and facility; the tasks that define requests for these products and processes, and an operational controller to control the flow of resources and execution of processes.

through DELS using the same mechanisms that passive resources use to flow (extended from commodity flow).

Finally, DELS define *maker*, *mover*, and *storer* placeholder roles. These parts suggest a canonical functional decomposition of each DELS, where the system designer selects resources to satisfy those required roles for making, moving, and storing material in the system. The next section describes modeling specialized DELS to satisfy these roles in the ecosystem.

3.9.1 Specializing DELS

`DELS` can be extended via specialization to model many kinds of DELS, reusing the libraries described in previous sections as needed. For example, `Process` can be specialized into a taxonomy of basic DELS functions: make, move, and store (figure 17). These processes are allocated to specialized DELS for `Production`, and `Material Handling`, and `Storage`, respectively (Figure 27). Allocating a `Process` to a `DELS`, such as `MOVE` to a `Material Handling System`, denotes a requirement to add an operation that executes that process when the operation is invoked. The DELS must provide a behavior that implements the operation (a method) by defining *process Steps* and *required Input Resources* used to execute that operation.

Many DELS are composed of other DELS (figure 27). For example, `Supply Chain` is composed of `Manufacturing Plants`, `Transportation Systems`, and `Depots`; which specialize (subset) the *maker*, *mover*, and *storer* roles, respectively. The `Supply Chain` uses these components to execute its high-level functional *SOURCE*() components from *suppliers* (typed by `Supply Chain`), *MAKE*() them into higher-value items, and *DELIVER*() products to *customers* [105].

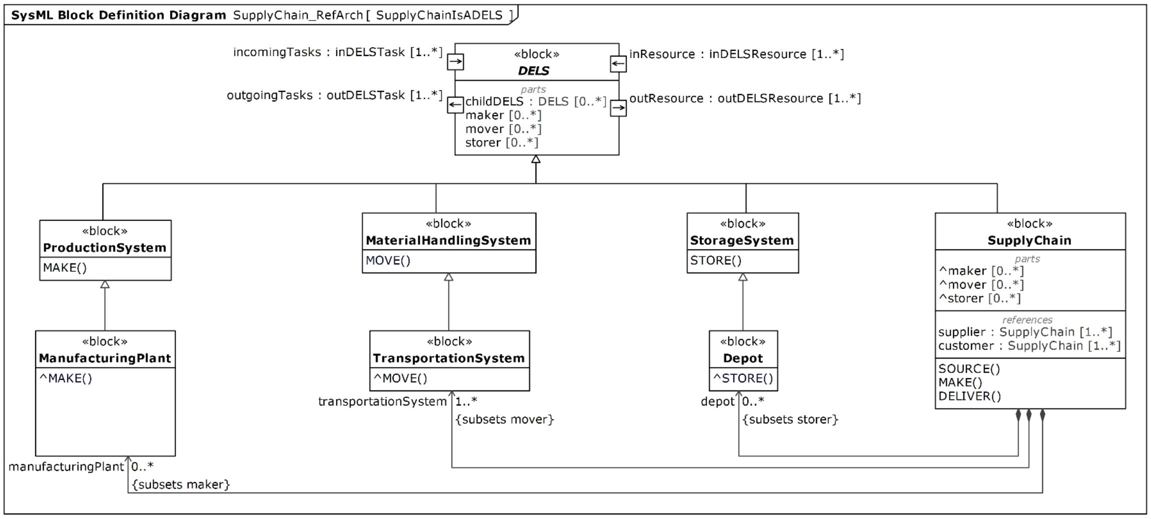

Fig. 27. Specialized systems can be created from the DELS definition. These specialized systems can be composed into new kinds of systems.

This composition-based modeling approach can be applied recursively, refining systems by identifying and modeling specialized subsystems to fulfill maker, mover, and storer roles. For example, manufacturing plants have production lines (specialized production systems), linked by material handling systems, and buffered by intermediate material buffers (specialized storage systems) (figure 28).

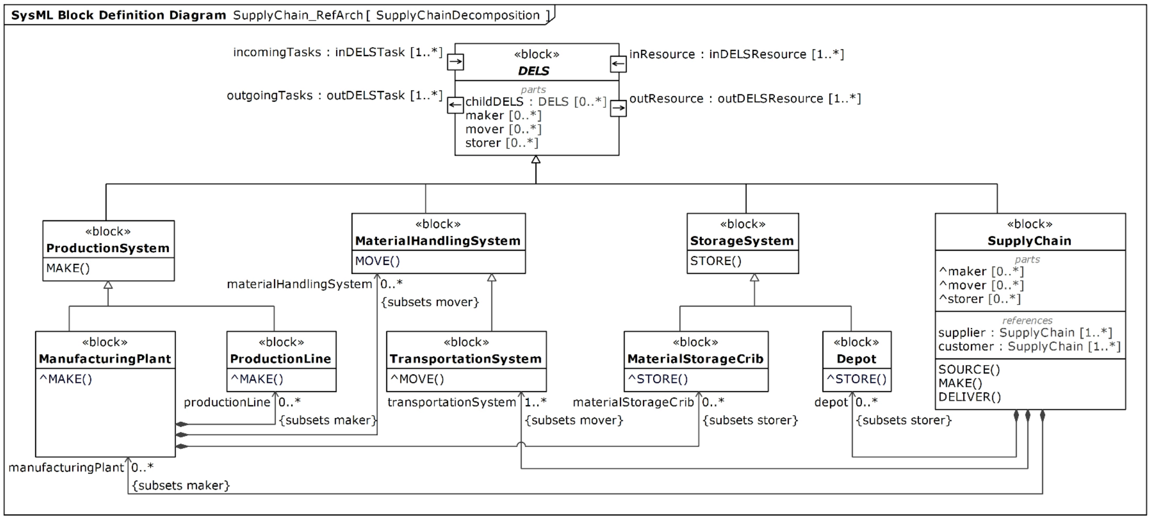

Fig. 28. Specialized DELS, such as Manufacturing Plants, are often themselves composed of other specialized DELS.

Each specialized kind of DELS can be further specialized to capture domain-specific features; for example, nuances between automotive and aerospace production lines. Composing DELS from specialized DELS, rather than defining monolithic systems composed of unique components, results in self-similar architectures which exhibit desirable qualities for designing, analyzing, and controlling these kinds of systems [17].

4. Discussion and Future Work

This paper documents a snapshot of the Commodity Flow Network (CFN) and Discrete Event Logistics Systems (DELS) models. The source models are archived here [2, 3].

This work fills an niche in the Industry 4.0 ecosystem, supporting analysis and functional design of heterogeneous production and logistics systems. There are a substantial number of standards providing detailed PPR specifications (see, e.g. ISO TC 184 activities, and surveys included in [42, 106, 107]). However much of the research is focused on the product being produced, leaving little in the way of linking detailed PPR specifications to analysis models supporting all lifecycle phases of the production system itself. There is a need for increased communication and collaborations between stakeholders that care about the product system and the production system. However, the art and science of production system design and specification must sufficiently advance to meet the detailed specifications typically found in the product engineering.

Additionally, many of these standards are domain-specific, focusing predominantly on smart manufacturing. However, modern enterprises integrate functionally heterogeneous systems that are often geographically distributed [13, 108]. Building an MBISE ecosystem based on the DELS model libraries provides a foundation to integrate or coordinate decision-making and execution across diverse systems as well as integrating the loosely-coupled Industry 4.0 research and development efforts spread out across the supply chain, transportation, production, and warehousing domains.

Releasing this document and the associated models represents a milestone in opening this work up to the community so that others can contribute to its development. The document and models remain living artifacts with open issues that continue to be identified and added to the living document as additional use cases and models are built from the model libraries and added to the ecosystem. The research goal focuses on building and expanding the MBISE ecosystem, including model libraries, reference architectures, supporting analysis tools, and design methodologies.

Acknowledgments

The authors would like to acknowledge Michael Brundage and Allison Barnard-Feeney at NIST and Thomas Hedberg at UMD/ARLIS for their helpful feedback.

References

[1] ANSI/ISA-95 (2010) ANSI/ISA-95.00.01-2010 (IEC 62264-1 mod) enterprise-control system integration - part 1: Models and terminology (International Society of Automation (ISA)), Standard.

[2] Sprock T GitHub\USNISTGov\discrete event logistics systems. Available at https://doi.org/10.18434/M32203.

[3] Sprock T, Bock C (2020) Model libraries supporting discrete event logistics systems (dels) models. *NIST Journal of Research* Available at https://doi.org/10.6028/jres.XXX.XXX.

[4] Thiers G (2014) *A Model-Based Systems Engineering Methodology to Make Engineering Analysis of Discrete-Event Logistics Systems More Cost-Accessible*. Ph.D. thesis. Georgia Institute of Technology, Atlanta, GA.

[5] Sprock T (2015) *A Metamodel of Operational Control of Discrete Event Logistics Systems (DELS)*. Ph.D. thesis. Georgia Institute of Technology, Atlanta, GA.

[6] OMG (2017) Omg systems modeling language (omg sysml) version 1.5, . Available at http://www.omg.org/spec/SysML/1.5/.

[7] Friedenthal S, Moore A, Steiner R (2014) *A Practical Guide to SysML: The Systems Modeling Language* (Morgan Kaufmann), .

[8] Bergenthal J (2011) Final report model based engineering (mbe) subcommittee. *NDIA Systems Engineering Division-M&S Committee* .

[9] Wang W, Tolk A, Wang W (2009) The levels of conceptual interoperability model: applying systems engineering principles to m&s. *Proceedings of the 2009 Spring Simulation Multiconference* (Society for Computer Simulation International), , p 168.

[10] Tolk A, Muguira JA (2003) The levels of conceptual interoperability model. *Proceedings of the 2003 Fall Simulation Interoperability Workshop*, Vol. 7, pp 1–11.

[11] Mellor SJ, Scott K, Uhl A, Weise D (2004) *MDA distilled: principles of model-driven architecture* (Addison-Wesley Professional), .

[12] MDA O (2014) OMG Model Driven Architecture (MDA) guide revision 2.0, . Available at https://www.omg.org/cgi-bin/doc?ormsc/14-06-01.

[13] Sprock T, Sharp M, Bernstein WZ, Brundage MP, Helu M, Hedberg T (2019) Integrated operations management for distributed manufacturing. *Proceedings of the 9th IFAC Conference on Manufacturing Modelling, Management and Control (MIM 2019)* (IFAC-PapersOnLine), , .

[14] Sprock T, McGinnis LF (2014) Simulation Model Generation of Discrete Event Logistics Systems (DELS) Using Software Patterns. *Proceedings of the 2014 Winter Simulation Conference* (IEEE Press), , pp 2714–2725.

[15] Thiers G, Sprock T, McGinnis L, Graunke A, Christian M (2016) Automated production system simulations using commercial off-the-shelf simulation tools. *Proceedings of the 2016 Winter Simulation Conference* (IEEE Press), , pp 1036–1047.

[16] Sprock T, Murrenhoff A, McGinnis LF (2017) A hierarchical approach to warehouse design. *International Journal of Production Research* 55(21):6331–6343.

[17] Sprock T (2018) Self-similar architectures for smart manufacturing and logistics systems. *Manufacturing Letters* 15:101–103.

[18] Sprock T, McGinnis LF (2015) A simulation optimization framework for discrete event logistics systems (DELS). *Proceedings of the 2015 Winter Simulation Conference* (IEEE Press), , pp 2776–2787.

[19] Cloutier RJ, Verma D (2007) Applying the concept of patterns to systems architecture. *Systems engineering* 10(2):138–154.

[20] Sprock T, Bock C (2017) Incorporating abstraction methods into system-analysis integration methodology for discrete event logistics systems. *Proceedings of the 2017 Winter Simulation Conference* (IEEE Press), , pp 966–976.

[21] SEBoK (2018) Systems engineering body of knowledge (SEBoK) – logical architecture model development, . Available at https://www.sebokwiki.org/wiki/Logical_Architecture_Model_Development.

[22] Cloutier R, Muller G, Verma D, Nilchiani R, Hole E, Bone M (2010) The concept of reference architectures. *Systems Engineering* 13(1):14–27.

[23] Drezner Z, Hamacher HW (1995) *Facility location* (Springer-Verlag New York, NY), .

[24] Ahuja RK, Magnanti TL, Orlin JB (1993) *Network flows: theory, algorithms, and applications* (Prentice Hall), .

[25] Walrand J (1988) *An introduction to queueing networks* (Prentice Hall), .

[26] Marzolla M (2010) The `qnetworks` toolbox: A software package for queueing networks analysis. *Analytical and Stochastic Modeling Techniques and Applications, 17th International Conference, ASMTA 2010, Cardiff, UK, Proceedings*, eds Al-Begain K, Fiems D, Knottenbelt WJ (Springer), *Lecture Notes in Computer Science*, Vol. 6148, pp 102–116.

[27] Smith CU, Lladó CM, Puigjaner R (2010) Performance model interchange format (pmif 2): A comprehensive approach to queueing network model interoperability. *Performance Evaluation* 67(7):548–568.

[28] Troya J, Vallecillo A (2014) Specification and simulation of queuing network models using domain-specific languages. *Computer Standards & Interfaces* 36(5):863–879.

[29] Herroelen W, De Reyck B, Demeulemeester E (1998) Resource-constrained project scheduling: a survey of recent developments. *Computers & Operations Research* 25(4):279–302.

[30] Schlenoff C, Gruninger M, Tissot F, Valois J, Lubell J, Lee J (2000) *The process specification language (PSL) overview and version 1.0 specification* (Citeseer), .

[31] Cassandras CG, Lafortune S (2008) *Introduction to discrete event systems* (Springer), .

[32] Fox MS, Barbuceanu M, Gruninger M (1995) An organisation ontology for enterprise modelling: preliminary concepts for linking structure and behaviour. *Proceedings of the Fourth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises* (IEEE), , pp 71–81.

[33] Wysk RA, Smith JS (1995) A formal functional characterization of shop floor control. *Computers & Industrial Engineering* 28(3):631–643.

[34] Smith SF, Becker MA (1997) An ontology for constructing scheduling systems. *Working Notes of 1997 AAAI Symposium on Ontological Engineering*, , pp 120–127.

[35] Madni AM, Lin W, Madni CC (2001) Ideon™: An extensible ontology for designing, integrating, and managing collaborative distributed enterprises. *Systems Engineering* 4(1):35–48.

[36] Lin HK, Harding JA, Shahbaz M (2004) Manufacturing system engineering ontology for semantic interoperability across extended project teams. *International Journal of Production Research* 42(24):5099–5118.

[37] Cutting-Decelle AF, Young RI, Michel JJ, Grangel R, Le Cardinal J, Bourey JP (2007) ISO 15531 MANDATE: a product-process-resource based approach for managing modularity in production management. *Concurrent Engineering* 15(2):217–

235.

[38] ISO-15531-1:2004 (2004) ISO-15531-1:2004 industrial automation systems and integration – industrial manufacturing management data – part 1: General overview (International Organization for Standardization), Standard.

[39] McLean CR, Lee YT, Shao G, Riddick F, Leong S (2005) *Shop data model and interface specification* (US Department of Commerce, Technology Administration, National Institute of Standards and Technology), .

[40] Lee YTT, Riddick FH, Johansson BJI (2011) Core Manufacturing Simulation Data - a manufacturing simulation integration standard: Overview and case studies. *International Journal of Computer Integrated Manufacturing* 24(8):689–709.

[41] Lemaignan S, Siadat A, Dantan JY, Semenenko A (2006) Mason: A proposal for an ontology of manufacturing domain. *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications* (IEEE), , pp 195–200.

[42] Lu Y, Morris KC, Frechette S (2016) Current standards landscape for smart manufacturing systems. *National Institute of Standards and Technology, NISTIR* 8107.

[43] Rouwenhorst B, Reuter B, Stockrahm V, Van Houtum G, Mantel R, Zijm W (2000) Warehouse design and control: Framework and literature review. *European Journal of Operational Research* 122(3):515–533.

[44] Kovacs G, Spens KM (2007) Humanitarian logistics in disaster relief operations. *International Journal of Physical Distribution & Logistics Management* 37(2):99–114.

[45] Pham DN, Klinkert A (2008) Surgical case scheduling as a generalized job shop scheduling problem. *European Journal of Operational Research* 185(3):1011–1025.

[46] Hulshof PJ, Kortbeek N, Boucherie RJ, Hans EW, Bakker PJ (2012) Taxonomic classification of planning decisions in health care: a structured review of the state of the art in or/ms. *Health Systems* 1(2):129–175.

[47] Golden B, Assad A, Levy L, Gheysens F (1984) The fleet size and mix vehicle routing problem. *Computers & Operations Research* 11(1):49–66.

[48] Salhi S, Rand GK (1993) Incorporating vehicle routing into the vehicle fleet composition problem. *European Journal of Operational Research* 66(3):313–330.

[49] Godfrey GA, Powell WB (2002) An adaptive dynamic programming algorithm for dynamic fleet management, i: Single period travel times. *Transportation Science* 36(1):21–39.

[50] Chan F, Wong T, Chan L (2006) Flexible job-shop scheduling problem under

resource constraints. *International Journal of Production Research* 44(11):2071–2089.

[51] Powell WB, Shapiro JA, Simao HP (2001) A representational paradigm for dynamic resource transformation problems. *Annals of Operations Research* 104(1):231–279.

[52] T'kindt V, Billaut JC (2006) *Multicriteria scheduling: theory, models and algorithms* (Springer Science & Business Media), .

[53] Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* 207(1):1–14.

[54] Fadel FG, Fox MS, Gruninger M (1994) A generic enterprise resource ontology. *Proceedings of Third Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises* (IEEE), , pp 117–128.

[55] Uschold M, King M, Moralee S, Zorgios Y (1998) The enterprise ontology. *The Knowledge Engineering Review* 13(01):31–89.

[56] Zhang Y, Feng SC, Wang X, Tian W, Wu R (1999) Object oriented manufacturing resource modelling for adaptive process planning. *International Journal of Production Research* 37(18):4179–4195.

[57] Zhou M, Wu N (2009) *System modeling and control with resource-oriented Petri nets*. Vol. 35 (CRC Press), .

[58] Zülch G, Fischer J, Jonsson U (2000) An integrated object model for activity network based simulation. *Proceedings of the 32nd conference on Winter simulation* (Society for Computer Simulation International), , pp 371–380.

[59] Van Mieghem JA (1998) Investment strategies for flexible resources. *Management Science* 44(8):1071–1078.

[60] Van Mieghem JA, Rudi N (2002) Newsvendor networks: Inventory management and capacity investment with discretionary activities. *Manufacturing & Service Operations Management* 4(4):313–335.

[61] Buzacott J, Hanifin LE (1978) Models of automatic transfer lines with inventory banks a review and comparison. *AIIE transactions* 10(2):197–207.

[62] Wu K (2014) Classification of queueing models for a workstation with interruptions: a review. *International Journal of Production Research* 52(3):902–917.

[63] Hackman ST, Leachman RC (1989) A general framework for modeling production. *Management Science* 35(4):478–495.

[64] Balakrishnan N, Sridharan V, Patterson JW (1996) Rationing capacity between two

product classes. *Decision Sciences* 27(2):185–214.

[65] Kacar NB, Uzsoy R (2015) Estimating clearing functions for production resources using simulation optimization. *IEEE Transactions on Automation Science and Engineering* 12(2):539–552.

[66] Mati Y, Xie X (2011) Multiresource shop scheduling with resource flexibility and blocking. *IEEE transactions on automation science and engineering* 8(1):175–189.

[67] Smith R (1980) The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers* 29:12.

[68] Jammes F, Smit H, Lastra JLM, Delamer IM (2005) Orchestration of service-oriented manufacturing processes. *2005 IEEE conference on emerging technologies and factory automation* (IEEE), Vol. 1, pp 8–pp.

[69] Melkote S (2012) Development of ifab manufacturing process and machine library (Georgia Institute of Technology, Atlanta, GA), DARPA/TTO Contract FA8650-11-C-7142.

[70] MTConnect Institute (2019) ANSI MTConnect Version 1.5.0 (ANSI/MTC1.5-2019). Available at https://www.mtconnect.org/standard20181.

[71] Organization for Machine Automation and Control (OMAC) (2015) Machine And Unit States: An Implementation Example Of ANSI/ISA-88.00.01 (ANSI/ISA TR88.00.02-2015).

[72] IPC: CAMX Frameworks Communication Committee (2003) ANSI/IPC-2501: Definition for Web-Based Exchange of XML Data. Available at http://www.ipc.org/4.0_Knowledge/4.1_Standards/IPC-2501.pdf.

[73] ISO/TC 184/SC 5 (2019) Automation systems and integration — Equipment behaviour catalogues for virtual production system (ISO/DIS 16400). Available at https://www.iso.org/standard/73384.html.

[74] Smith SF, Lassila O, Becker M (1996) Configurable, mixed-initiative systems for planning and scheduling. *Advanced Planning Technology* :235–241.

[75] Roy B, Sussmann B (1964) Les problemes d'ordonnancement avec contraintes disjonctives. *Note DS* 9.

[76] Balas E (1969) Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations research* 17(6):941–957.

[77] Dauzère-Pérès S, Paulli J (1997) An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research* 70:281–306.

[78] Kis T (2003) Job-shop scheduling with processing alternatives. *European Journal of Operational Research* 151(2):307–332.

[79] Burdett RL, Kozan E (2010) A disjunctive graph model and framework for constructing new train schedules. *European Journal of Operational Research* 200(1):85–98.

[80] Homem de Mello LS, Sanderson AC (1990) And/or graph representation of assembly plans. *IEEE Transactions on Robotics and Automation* 6(2):188–199.

[81] Catron BA, Ray SR (1991) Alps: A language for process specification. *International Journal of Computer Integrated Manufacturing* 4(2):105–113.

[82] Gillies DW, Liu JWS (1995) Scheduling tasks with and/or precedence constraints. *SIAM Journal on Computing* 24(4):797–810.

[83] Senehi MK, Barkmeyer EJ, Luce ME, Ray SR, Wallace EK, Wallace S (1991) Manufacturing systems integration initial architecture document. *NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, forthcoming* .

[84] Schlenoff C, Ciocoiu M, Libes D, Gruninger M (1999) Process specification language(psl): Results of the first pilot implementation. *AM SOC MECH ENG MANUF ENG DIV MED* 10:529–539.

[85] ISO-10303-242:2014 (2014) ISO 10303-242:2014 industrial automation systems and integration — product data representation and exchange — part 242: Application protocol: Managed model-based 3d engineering (International Organization for Standardization), Standard. Available at https://www.iso.org/standard/57620.html.

[86] 62264-1:2013 I (2013) IEC 62264-1:2013 enterprise-control system integration — part 1: Models and terminology (International Organization for Standardization), Standard. Available at https://www.iso.org/standard/57308.html.

[87] Fenves SJ, Foufou S, Bock CE, Bouillon N, Sriram RD (2005) Cpm 2: A revised core product model for representing design information. *NIST Interagency/Internal Report (NISTIR)-7185* .

[88] Tursi A, Panetto H, Morel G, Dassisti M (2009) Ontological approach for products-centric information system interoperability in networked manufacturing enterprises. *Annual Reviews in Control* 33(2):238–245.

[89] Panetto H, Dassisti M, Tursi A (2012) Onto-pdm: Product-driven ontology for product data management interoperability within manufacturing process environment. *Advanced Engineering Informatics* 26(2):334–348.

[90] Koopmans TC, Beckmann M (1957) Assignment problems and the location of economic activities. *Econometrica: Journal of the Econometric Society* :53–76.

[91] Drira A, Pierreval H, Hajri-Gabouj S (2007) Facility layout problems: A survey. *Annual Reviews in Control* 31(2):255–267.

[92] Owen SH, Daskin MS (1998) Strategic facility location: A review. *European Journal of Operational Research* 111(3):423–447.

[93] Smith J, Joshi S, Qiu R (2003) Message-based part state graphs (mpsg): a formal model for shop-floor control implementation. *International Journal of Production Research* 41(8):1739–1764.

[94] Riddick F, Lee YT (2008) Representing layout information in the cmsd specification. *Proceedings of the 2008 Winter Simulation Conference* (IEEE), , pp 1777–1784.

[95] OGC IndoorGML (2018) OGC indoorgml v.1.0.3 (Open Geospatial Consortium (OGC)), Standard.

[96] Eastman C, Teicholz P, Sacks R, Liston K (2011) *BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors* (John Wiley & Sons), .

[97] ISO 19650 (2018) ISO 19650-1:2018 organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) — information management using building information modelling — part 1: concepts and principles (International Standards Organization (ISO)), Standard.

[98] Russell S, Norvig P (1995) *Artificial Intelligence: A Modern Approach* (Prentice-Hall), .

[99] Thórisson KR, Bieger J, Thorarensen T, Sigurðardóttir JS, Steunebrink BR (2016) Why artificial intelligence needs a task theory. *Artificial General Intelligence* (Springer), , pp 118–128.

[100] Sandholm T, Lesser VR (1995) Issues in automated negotiation and electronic commerce: Extending the contract net framework. *ICMAS '95*, , pp 328–335.

[101] Wortmann J (1983) A classification scheme for master production scheduling. *Efficiency of manufacturing systems* (Springer), , pp 101–109.

[102] Van Brussel H, Wyns J, Valckenaers P, Bongaerts L, Peeters P (1998) Reference architecture for holonic manufacturing systems: Prosa. *Computers in Industry* 37(3):255–274.

[103] Sprock T, McGinnis LF (2015) A conceptual model for operational control in smart manufacturing systems. *IFAC-PapersOnLine* 48(3):1865–1869.

[104] Sprock T, Bock C, McGinnis LF (2018) Survey and classification of operational

control problems in discrete event logistics systems (dels). *International Journal of Production Research* :1–24.

[105] SCOR (2012) SCOR: The Supply Chain Reference Model Version 11.0 (Supply Chain Council, Inc), Technical report.

[106] Helu M, Joseph A, Hedberg Jr T (2018) A standards-based approach for linking as-planned to as-fabricated product data. *CIRP Annals* .

[107] Bernstein WZ, Hedberg Jr TD, Helu M, Feeney AB (2018) Contextualising manufacturing data for lifecycle decision-making. *International journal of product lifecycle management* 10(4):326.

[108] Hedberg T, Helu M, Sprock T (2018) A standards and technology roadmap for scalable distributed manufacturing systems. *ASME 2018 13th International Manufacturing Science and Engineering Conference* (American Society of Mechanical Engineers Digital Collection), , .