NIST Series Interagency Report
NIST IR 8571

SysML Extension for Logistics Modeling and Analysis (SysLMA)

Raphael Barbau
Conrad Bock

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8571

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Series Interagency Report
NIST IR 8571

SysML Extension for Logistics Modeling and Analysis (SysLMA)

Raphael Barbau
Associate, Smart Connected Systems Division
Communication Technology Laboratory
University of Maryland
College Park, MD

Conrad Bock
Smart Connected Systems Division
Communication Technology Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8571

April 2025

U.S. Department of Commerce
Howard Lutnick, Secretary

National Institute of Standards and Technology
Craig Burkhardt, Acting Under Secretary of Commerce for Standards and Technology and Acting NIST Director

Certain equipment, instruments, software, or materials, commercial or non-commercial, are identified in this paper in order to specify the experimental procedure adequately. Such identification does not imply recommendation or endorsement of any product or service by NIST, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

This publication was produced as part of grant awards #70NANB18H161, #70NANB18H165, and #70NANB23H024 with the National Institute of Standards and Technology. The contents of this publication do not necessarily reflect the views or policies of the National Institute of Standards and Technology or the US Government.

How to cite this NIST Technical Series Publication:
Raphael Barbau, Conrad Bock (2025) SysML Extension for Logistics Modeling and Analysis (SysLMA). (National Institute of Standards and Technology, Gaithersburg, MD), NIST IR 8571. https://doi.org/10.6028/NIST.IR.8571

Author ORCID iDs
Raphael Barbau: 0000-0002-0331-2929
Conrad Bock: 0009-0009-3172-120X

Abstract

Many engineers use analysis tools to predict behavior or find optimal values for properties of systems built to proposed designs, informing system design work by identifying cases where a design will perform incorrectly or in suboptimal ways. Design and analysis tools encode descriptions of systems in a variety of ways that are typically inconsistent, significantly slowing engineering processes due to the additional effort resolving these problems. Systems engineering addresses this by establishing a single source for system descriptions, typically expressed in the Systems Modeling Language (SysML®). Systems engineering models intentionally do not cover all aspects of other engineering disciplines, but must integrate with their tools to prevent adding to inconsistency in engineering information. Coordinating systems information across engineering analysis this way requires automated translation between systems models and discipline-specific analysis tools, based on adding information in systems models needed for analysis. This report takes this approach for logistics analysis, which predicts and optimizes movement of separate things between activities that potentially modify them, in particular, multi-commodity flow optimization, queuing analysis, and discrete event simulation. It presents a SysML extension for modeling systems to be analyzed, along with translation patterns to widely-used logistics analysis tools, examples of modeling logistics systems, as well as translation and analysis of these models.

Keywords

Logistics, Analysis, Systems engineering, SysML.

Table of Contents

List of Tables

List of Figures

1 Introduction

Many engineers use analysis tools to predict behavior or find optimal values for properties of systems built to proposed designs, avoiding significant costs and time building and testing physical prototypes. Results of analysis inform system design work by identifying cases where systems built to a design will perform incorrectly or in suboptimal ways. Designs are modified to address these problems and analyzed again to see if further work is needed. If not, designs are released for prototyping or production.

Design and analysis tools typically include descriptions of system components, their interconnections, some of their expected behavior, and desired interactions with the environment in which a system is expected to be operated. This information is encoded separately in each kind of tool, reflecting concerns of the engineering discipline involved (communications, control, material, electromechanical, production, and so on). The information is usually entered by hand and expressed in terms specific to those disciplines, based on documents that are primarily for human consumption and not formal enough for computational assistance. These manually constructed descriptions of a system being designed or analyzed naturally are not quite the same across tools at first and their differences grow over time as each discipline changes them without fully coordinating with the others. Overall engineering processes become significantly inefficient due to the additional effort of resolving these inconsistencies.

Systems engineering aims to coordinate the work of all these disciplines in part by establishing a single source for descriptions of system components, their interconnections, behaviors, and desired operational interactions and effects. A single source for system design information encourages alignment of redundant descriptions across the disciplines by providing coordinated input to their work and for modifications needed based on their results. This system information is typically expressed in the Systems Modeling Language (SysML®), the most widely-used standard for this purpose [1], based on a similar language for software systems, the Unified Modeling Language (UML®) [2]. Systems engineering models are shared between all disciplines and intentionally do not cover all aspects of those disciplines, but must integrate with their tools to prevent adding to redundancy and inconsistency in engineering information.

Coordinating systems information across engineering analysis requires automated translation between systems models and discipline-specific analysis tools in a way that produces the same or negligibly different results in all tools that perform the same kind of analysis. Such translations depend on system models recording additional information needed for translation. SysML enables additions to it with a *profile* mechanism, in this case for recording information needed to translate to and from analysis tools in each discipline. These can be applied to pre-built *model libraries* containing commonly needed system components in each discipline, giving engineers a palette of components to choose from when build their system models. Profiles and libraries together *extend* SysML.

This report takes the above approach for logistics analysis, which predicts and optimizes movement of separate things between activities that potentially modify them, refining some aspects of earlier work on this topic [3]. The systems being analyzed are collections of activities joined by paths along which things flow (logistics *networks*). The things flowing can be of any kind, as long they are

separably identifiable (“discrete”), including physical (as in factories, warehouses, supply chains, and transporation lines), human (as in hospitals and restaurants), or informational (as in data packets or email routes).

Three kinds of logistics analysis are addressed here:

- Multi-Commodity Flow Optimization (MCFO) finds the least expensive way to move things through a network.
- Queuing Analysis (QA) calculates long-term statistics in a network that involves things waiting for activities that potentially modify them.
- Discrete Event Simulation (DES) predicts moment-by-moment flow of things through a network.

The systems being analyzed are modeled with the SysML Extension for Logistics Modeling and Analysis (SysLMA), consisting of:

- Profiles for additional information needed to model and analyze logistics networks (for MCFO and QA).
- SysML libraries of logistics elements that can be reused in logistics network models.
- A human-usable textual notation for expressions (for DES).

The libraries are built up as illustrated on the left of Figure 1, with the lower ones adding to those above them. The three kinds of logistics analysis shown on the right point to the libraries they operate on. The libraries introduce the terms *commodity* for things flowing through a network, *node* for activities that operate on them, and *edge* for paths that commodities flow along between nodes.

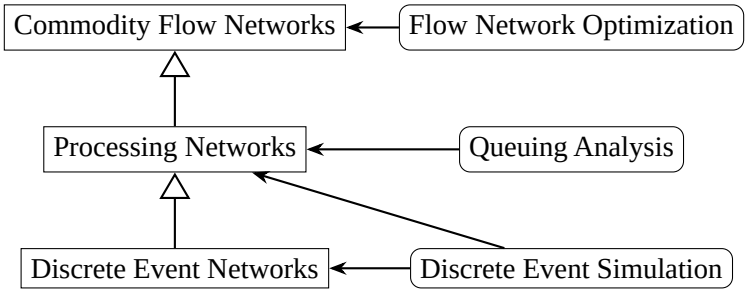

Figure 1: Network model libraries and corresponding analyses

- Commodity Flow Network (CFN), the most general library, is concerned with production and consumption rates of commodities at the boundaries of nodes (inputs/outputs), as well as commodity flows between these nodes. CFN does not specify the internal behavior of nodes. The SysML extension provides a CFN library and a profile to model CFNs and attach analysis information needed for flow network optimization, to find the least expensive way to move commodities in the network. See Section 2.

- Processing Network (PN) is a more specific library that contains nodes with a particular kind of internal behavior: all commodities flowing in them are queued and then serviced. The SysML extension provides a PN library and a profile to model PNs and attach analysis information needed for queuing analysis and discrete event simulation, to calculate long-term statistics in a network that involves queuing, servicing, and random commodity arrivals. See Section 3.
- Discrete Event Network (DEN) is the most specific library, the only one that simulates network behavior over time. Nodes can have specialized internal behavior corresponding to the primitives of DES tools, so logistics models can be better customized by a variety of events distributions, custom expressions associated with on node events, commodity routing based on expressions or probabilities, and combining and splitting commodities. The SysML extension provides a DEN library containing these primitives as UML activities, and a language that can be used for the modeling of DEN expressions in SysML. See Section 4.

Each section referenced above also provides:

- Example logistics networks modeled with the above libraries and profiles.
- Translation patterns between logistics networks modeled as above and two widely-used tools for each kind of analysis.
- Translations of the example models per the patterns above, as produced by a prototype software implementation developed for this report [4].
- Results of analyzing the example translations, showing they are the same for the tools used.

SysLMA enables coordination between the many encodings of logistics network components, their interconnections, some of their expected behavior, and desired operation, and eliminates manual recoding of every networks in each analysis tools. The libraries enable logistics models to be built more quickly by reusing library elements rather than reconstructing them for each application. Taken together, these capabilities provide a basis for more efficient engineering for logistics systems.

Section 5 defines a SysML extension to represent time intervals between randomly-distributed events. It can be used in all kinds of logistics networks to specify the time between two successive commodities passing through a location, or to specify the time it takes to process a commodity in PNs and DENs.

Section 6 summarizes the report and outlines future work. Annex A outlines a prototype software implementation that automates the translations presented in this report. Annex B gives case and font conventions for model and analysis tool element names. Annex C reviews aspects of SyML used in the report and gives case anbd font conventions to referring to model and analysis elements.

2 Commodity Flow Networks

This section presents Commodity Flow Networks (CFNs) and translations between extended SysML models and Multi-Commodity Flow Optimization (MCFO). These networks contain nodes and edges between them, where commodities flow within nodes and edges, and MCFO finds the least expensive way to move commodities from their origin to their destination in the network. Commodities of the same kind are interchangeable, though still distinguishable, for example by serial numbers. Subsection 2.1 reviews CFN and MCFO, and Subsection 2.2 defines an extension of SysML for CFN. Subsection 2.3 provides a simple example of CFN modeled using the SysML extension. Subsection 2.4 provides a translation between extended SysML models and MCFO tools, and Subsection 2.5 describes how the simple example is automatically translated into analysis tools.

2.1 Modeling and analysis

CFNs are made of nodes linked by edges. Nodes can take commodities as input and provide them as output while the network is operating, with these commodities flowing across edges from the outputs of a node to the input of other nodes, or to the same node.

Nodes also produce and consume commodities. Produced commodities appear in the network for the first time as outputs of the node producing them (no other node can produce the same ones, though other nodes might produce commodities of the same kind). Similarly, consumed commodities disappear from the network after being input to the node consuming them (no other node can consume them, though other nodes might consume commodities of the same kind). This means produced commodities are not input to the node producing them and consumed commodities are not output from the node consuming them.

In formal terms, the sets of input, output, produced, and consumed commodities for a node are related as follows:

$$produced \subseteq output \tag{1}$$
$$produced \cap input = \varnothing \tag{2}$$
$$consumed \subseteq input \tag{3}$$
$$consumed \cap output = \varnothing \tag{4}$$

One consequence of these definitions is the same commodity cannot be consumed and produced by the same node (though commodities of the same kind might be produced and consumed by the same node), reflected in lines 2 and 3, which imply $produced \cap consumed = \varnothing$. This means commodities flowing within a node are the same as its combined inputs and outputs:

$$flowingCommodity = input \cup output \tag{5}$$

Edges do not have inputs or outputs, since all commodities coming into them also flow out, and vice versa, but they have $flowingCommodity$, to identify the commodities moving along them.

The definitions of production and consumption above lead to a "balance equation" for the commodities flowing in and out of each node:

$$input \cup produced = output \cup consumed \tag{6}$$

which can be informally expressed as:

$$input \text{ and } produced = output \text{ and } consumed$$
$$\text{or}$$
$$input - consumed = output - produced$$

The second form above clarifies that input commodities not consumed in a node always "pass through" it as outputs that are not produced.

The following are special cases of Equation 6:

- All outputs are produced: $produced \neq \varnothing$ and $input = \varnothing$ implies $output = produced$
- All inputs are consumed: $consumed \neq \varnothing$ and $output = \varnothing$ implies $input = consumed$
- All inputs are also output: $produced = consumed = \varnothing$, implies $input = output$

MCFO is a kind of analysis performed on CFNs to find a way to move commodities from producing to consuming nodes so that the overall *cost* is minimal. This is often financial cost, but could be a measure of environmental effects, transportation time or other quantities. This kind of optimization requires additional information in the network for flow rates (production, consumption), costs (fixed costs, unit costs) and limits (flow rates limits). MCFO does not consider any behavior specified for nodes and edges (see Sections 3 and 4). With these additions to a CFN, MCFO attempts to find the optimal (least expensive) flow of commodities in a network, specifically an optimal flow rate on each edge for each kind of commodity.

Optimizer will not be able to find a solution in some cases, for example, when for any kind of commodity:

- the sum of production rates of all nodes in a network is not the same as the sum of consumption rates
- missing edges or flow rate limits on edges or nodes that prevent some or all commodities from moving from producing nodes to consuming nodes

Equation 6 implies a similar one about rates, for every node and every commodity kind, as shown in Equation 7.

$$input_{rate} + produced_{rate} = output_{rate} + consumed_{rate} = flowingCommodity_{rate} \tag{7}$$

Since produced commodities are also outputs and consumed commodities are also inputs, and all the above rates must be $\geqslant 0$, the following inequalities can be implied:

$$\begin{aligned} output_{rate} &\geqslant produced_{rate} \geqslant 0 \\ input_{rate} &\geqslant consumed_{rate} \geqslant 0 \end{aligned} \tag{8}$$

Note that the $produced_{rate}$ and $consumed_{rate}$ are provided by the modeler, while the $input_{rate}$ and $output_{rate}$ are the sum of the incoming and outgoing edge flow rates, as determined by optimization.

Limits can be set to rates of flow across edges. These limits can be for each kind of commodity separately, or for all commodities combined.

Finally, two kinds of costs contribute to total cost: fixed and unit. Fixed costs apply to nodes or edges only when commodities of any kind flow within them, regardless of the rate. Unit costs apply to edges and their contribution to total cost depends on the rate of flow each kind of commodity. The unit cost for each kind of commodity is multiplied by its flow rate through an edge and by a length of time time over cost is being computed, which must be the same for all unit costs. Solving for a duration of 1 effectively treats rates of commodities as amounts of commodities.

2.2 SysML extension

The SysML representation of CFNs consists of a:

- *Library* for modeling CFNs as described in Subsection 2.1, see Subsection 2.2.1. It is the most general library in this specification, with the others (indirectly) specializing it in Subsections 3.2.1 and 4.2.1.
- *Profile* to add analysis information necessary to perform MCFO as described in Subsection 2.1, see Subsection 2.2.2. It defines stereotypes applied to value properties that carry analysis information.

Modelers use the extension by specializing library blocks and properties, then applying stereotypes as needed for translation to analysis tools, as described in Subsection 2.4. See Appendix B for case and font conventions for model and analysis tool element names and Appendix C for an introduction to SysML.

2.2.1 Library

Figure 2 shows the CFN library modeling the concepts introduced in 2.1 (such as nodes, links, commodities), as well as ones specifically for MCFO (such as cost, rate limits), defined as a SysML taxonomy. Below are description of blocks in the taxonomy.

*FlowNetwork*s contain *FlowNode*s (identified by the property *node*) and *FlowNetworkLink*s (identified by the property *edge*). The *edge*s of a *FlowNetwork* must connect *node*s of that same network or one of their *port*s. *FlowNetwork*s are kinds of *FlowNode*s, to enable other networks to contain them. *FlowNetwork*s can identify some *AtomicFlowNode*s as *portNode*s, to be linked in those other networks (see *FlowNetworkLink*s below).

*FlowNetworkLink*s are between two *AtomicFlowNode*s, another kind of *FlowNode*. *AtomicFlowNode*s do not contain of any *FlowNode*s or *FlowNetworkLink*s, instead they define two flow properties

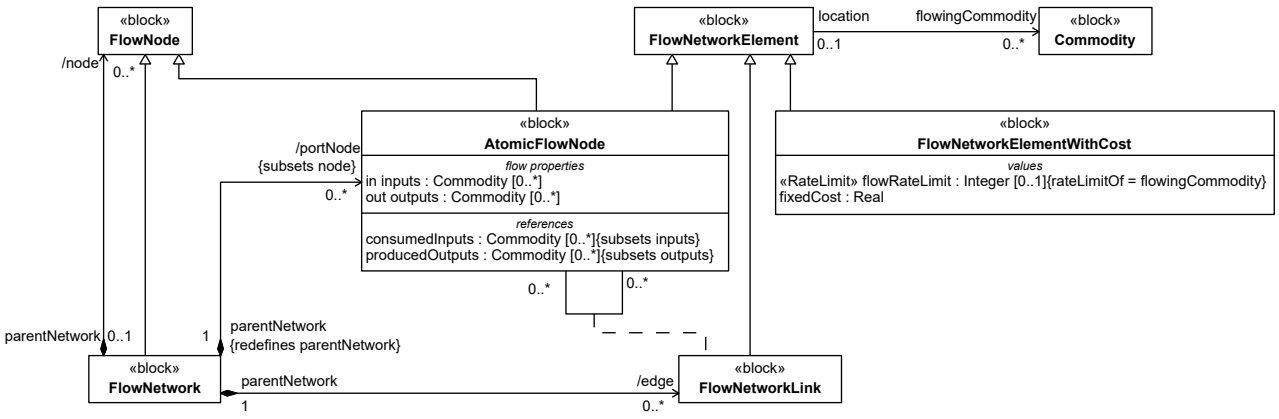

Figure 2: Commodity Flow Network Library

inputs and *outputs*, and two reference properties subsetting those to identify inputs that will be consumed (*consumedInputs*) and outputs that have been produced (*producedOutputs*). See Subsection 2.1 about production and consumption.

*FlowNetworkLink*s and *AtomicFlowNode*s are *FlowNetworkElement*s, enabling *Commodity*s to flow into, out of, or through them, identified by the *flowingCommodity* property. *Commodity*s are discrete (i.e. distinct, countable) and the *FlowNetworkElement* they flow within is their *location*. Commodities flow within *FlowNetwork*s also, but only due to their *FlowNetworkLink*s and *AtomicFlowNode*s.

*FlowNetworkElementWithCost*s add MCFO information to *FlowNetworkLink*s and *AtomicFlowNode*s (specializations of *FlowNetworkElementWithCost* must also specialize *AtomicFlowNode* or *FlowNetworkLink*):

- *flowRateLimit* is an integer giving the maximum flow rate for all kinds of commodities combined through a *FlowNetworkLink* or within an *AtomicFlowNode*;
- *fixedCost* is a real number giving a cost for an element that has any commodities flowing within it, regardless of how many, as long as there are some.

See Subsection 2.1 for more information about these properties.

2.2.2 Profile

The CFN profile contains stereotypes for analysis information, as shown in Figure 3 (see Appendix C about stereotypes). The stereotypes UnitCost, Rate, and RateLimit apply to value properties that give the information, identifying other properties for each kind of commodity the information is about via stereotype properties unitCostOf, rateOf, and rateLimitOf, respectively. When these properties can have multiple values, they are ordered, with each stereotyped property value about the commodity property in the same position, if any. The stereotypes are described in subsequent paragraphs.

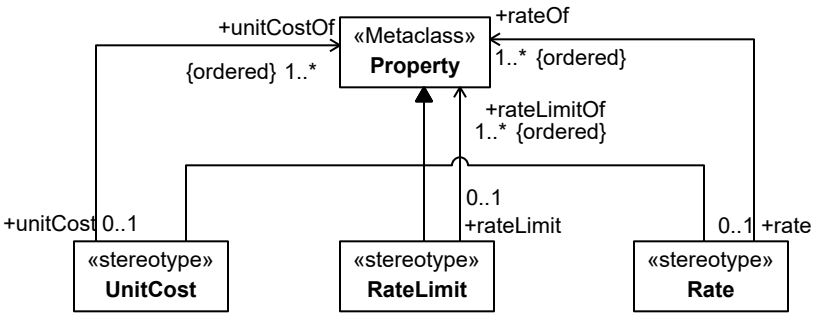

Figure 3: Commodity Flow Network Profile

Rate applies to properties giving input, output, and flowing commodities, including consumed inputs and produced outputs, as identified by the properties specified by rateOf. The values of the stereotyped property are the rate at which commodities are input and output to/from nodes, as well as the rate they flow within an element (including along an edge). When a property is stereotyped by both Rate and EventDistribution, its value is the rate of that distribution (see Section 5). The type of the stereotyped properties must be *Integer*, *Real*, or one of their specializations.

The value of the stereotyped property is a:

- production rate when the corresponding rateOf subsets *producedOutputs*,
- consumption rate when the corresponding rateOf subsets *consumedInputs*,
- output rate when the corresponding rateOf subsets *output*, but not *producedOutputs*,
- input rate when the corresponding rateOf subsets *input*, but not *consumedInputs*,
- flow rate within nodes or edges when the corresponding rateOf subsets *flowingCommodity*.

RateLimit applies to properties giving a maximum rate that commodities can be input, output, and flowing, including consumed inputs and produced outputs, as identified by the properties specified by rateLimitOf. Values of the stereotyped property limit the rate that commodities are input and output to/from nodes, as well as flowing within an edge or a node. Rate limits are optional, allowing the stereotyped property to have less values than the number of corresponding commodities, not limiting rates of the remaining commodities. The type of the stereotyped properties must be *Integer*, *Real*, or one of their specializations.

The application rules for RateLimit are the same as for Rate.

UnitCost applies to properties giving a cost for each flowing commodity identified by the properties specified by unitCostOf. The values of the stereotyped property are the cost of transporting one unit of a commodity through an element, where the kind of commodity corresponding to each value is specified by the type of the property at the same position in unitCostOf. The types of the stereotyped properties must by *Integer*, *Real*, or one of their specializations.

2.3 Commodity Flow Network example

Figure 4 shows blocks involved in simple clothing distribution network example. *ClothingDistributionElement* is the generic definition of elements of the clothing distribution network. It specializes *FlowNetworkElementWithCost* from the CFN library. There are two kinds of commodities in the network: *Shirt* and *Shoes*. The block *ClothingDistributionElement* includes subsets of *flowingCommodity* for these two kinds of commodities: *flowingShirt* typed by *Shirt*, and *flowingShoes* typed by *Shoes*. *ClothingDistributor* is the generic definition of nodes in the distribution network. It specializes *ClothingDistributionElement* and *AtomicFlowNode* from the CFN library. It includes subsets of *producedOutputs* and *consumedInputs* for the two kinds of commodities involved: *producedShirtOutputs*, *producedShoesOutputs*, *consumedShirtInputs*, and *condumedShoesInputs*. It also defines a flow rate property *rates* referring to these outputs/inputs using the *rateOf* property of the *Rate* stereotype.

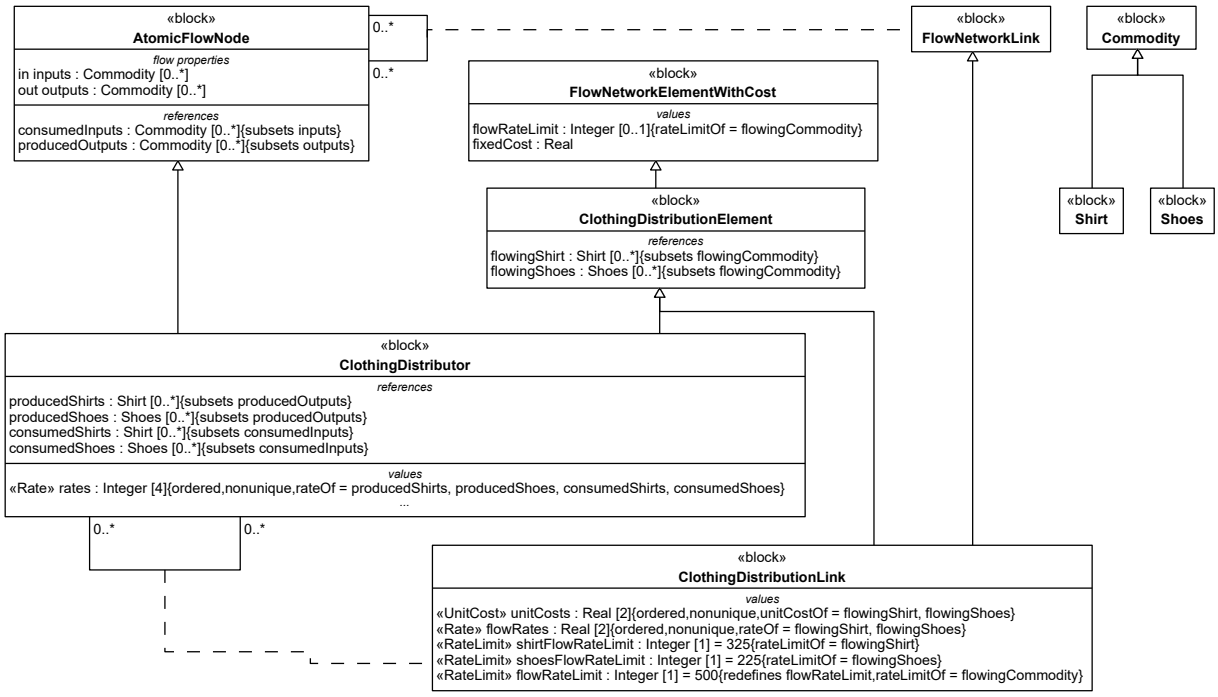

Figure 4: Distribution network in SysML, blocks

ClothingDistributionLink is the generic definition of edges in the distribution network. It specializes *ClothingDistributionElement* and *FlowNetworkLink* from the CFN library. It includes five additional properties:

- *unitCosts* provides unit costs for each commodity, using the *unitCostOf* property of the *UnitCost* stereotype;
- *flowRates* stores the optimal flow rate for each commodity, resulting from MCFO), using the *rateOf* property of the *Rate* stereotype;

- *shirtFlowRateLimit* and *shoesFlowRateLimit* provide the flow rate limit for the respective commodities, using the *rateLimitOf* property of the *RateLimit* stereotype; They have default values of 325 and 225 respectively;
- *rateLimit* is a redefinition of the property of the same name, with a default value of 500.

Figure 5 shows the four nodes that compose the distribution network, as well as their initial values:

- *la* fixed cost is 8000, shirt consumption rate is 350, and shoes production rate is 250;
- *omaha* fixed cost is 1000, and a maximum flow rate is 500;
- *nyc* fixed cost is 4000 and shoes consumption rate is 250;
- *miami* fixed cost is 2000 and shirt production rate is 350.

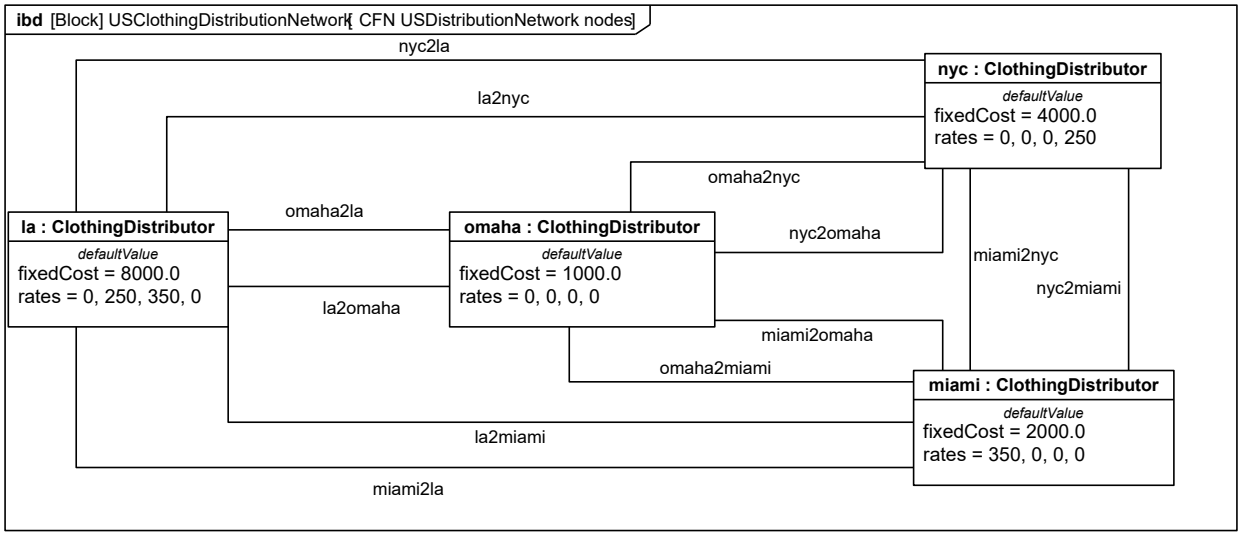

Figure 5: Distribution network in SysML, nodes

Figure 6 shows the edges that connect these four nodes, as well as their initial values for these connections:

- *la* to *omaha* fixed cost is 8000, shirt unit cost is 23, and shoes unit cost is 46;
- *la* to *nyc* fixed cost is 16000, shirt unit cost is 42, and shoes unit cost is 84;
- *la* to *miami* fixed cost is 16000, shirt unit cost is 39, and shoes unit cost is 78;
- *omaha* to *la* fixed cost is 1000, shirt unit cost is 23, and shoes unit cost is 46;
- *omaha* to *nyc* fixed cost is 1000, shirt unit cost is 20, and shoes unit cost is 40,
- *omaha* to *miami* fixed cost is 1000, shirt unit cost is 25, and shoes unit cost is 50;

- *nyc* to *la* fixed cost is 8000, shirt unit cost is 42, and shoes unit cost is 84;
- *nyc* to *omaha* fixed cost is 4000, shirt unit cost is 20, and shoes unit cost is 40;
- *nyc* to *miami* fixed cost is 4000, shirt unit cost is 19, and shoes unit cost is 38;
- *miami* to *la* fixed cost is 4000, shirt unit cost is 39, and shoes unit cost is 78;
- *miami* to *omaha* fixed cost is 2000, shirt unit cost is 25, and shoes unit cost is 50;
- *miami* to *nyc* fixed cost is 2000, shirt unit cost is 19, and shoes unit cost is 38.

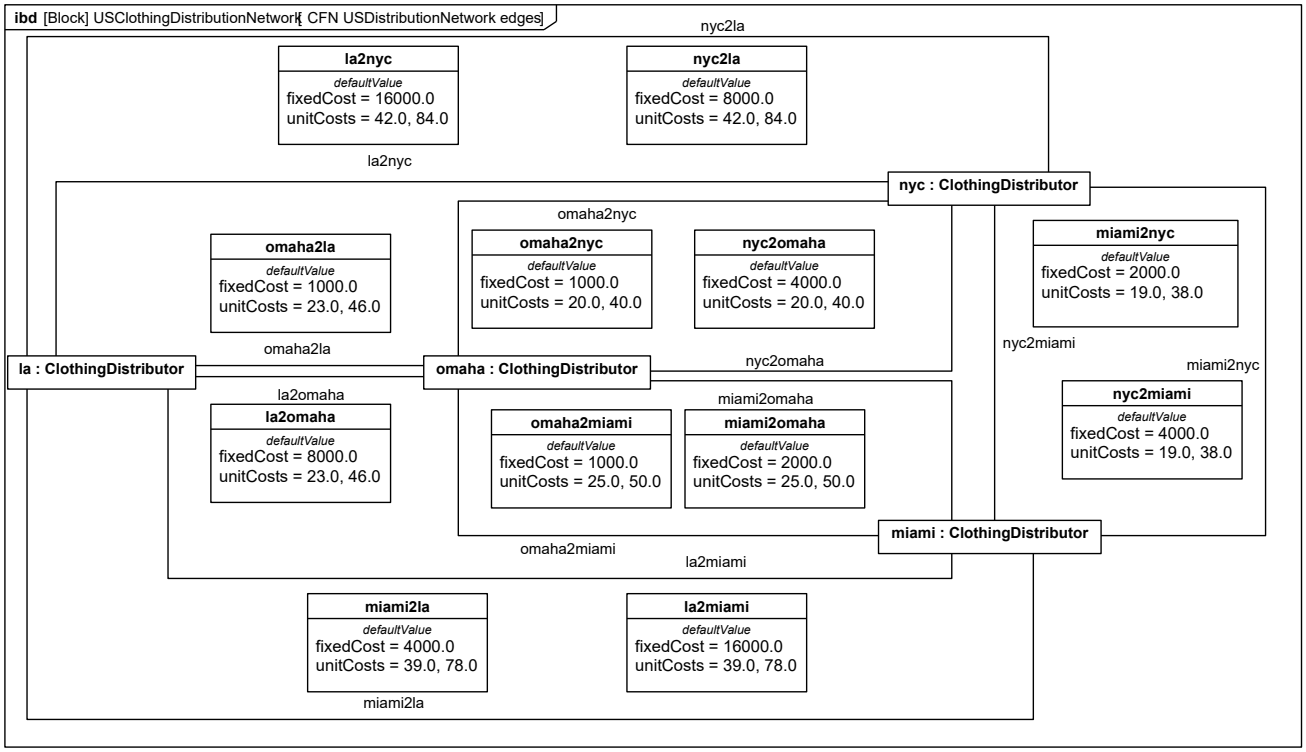

Figure 6: Distribution network in SysML, edges

2.4 Translating between Commodity Flow Networks and optimization tools

This subsection gives translations between SysML models extended for CFN, see Subsection 2.2, and A Modeling Language for Mathematical Programming (AMPL) and MATLAB, to express MCFO problems, consisting of constraints and objective functions, and to be solved by Linear Programming (LP). AMPL is a language specifically for LP, while MATLAB is a more general-purpose language for numerical analysis that includes LP. LP optimizers take vectors and matrices as input. The analysis files corresponding to SysML CFN models in this translation are:

- AMPL code, which is written in a higher-level language that AMPL tools automatically translate to LP vectors and matrices.

- MATLAB code giving LP vectors and matrices directly.

The MCFO results should be the same on all analysis tools.

The translation takes as input a specialization of *CommodityFlowNetwork* that is referred to as the *root* network.

2.4.1 Additional SysML model constraints for translation

The translation in Subsections 2.4.3 and 2.4.4 require SysML models to meet additional constraints. These are about the following kinds of commodity types in the root network being translated:

- Production types: For all properties that are 1) directly or indirectly in owned atomic nodes, 2) stereotyped by Rate, and 3) subset of *producedOutputs*, these are the types of all the properties identified by rateOf.
- Consumption types: For all properties that are 1) directly or indirectly in owned atomic nodes, 2) stereotyped by Rate, and 3) subset of *consumedInputs*, these are the types of all the properties identified by rateOf.
- Flow rate type: For all properties that are 1) directly or indirectly in owned atomic nodes, 2) stereotyped by Rate, and 3) subset of *flowingCommodity*, these are the types of all the properties identified by rateOf.
- Flow rate limit types: For all properties that are 1) directly or indirectly in owned atomic nodes, 2) stereotyped by RateLimit, and 3) subset of *flowingCommodity*, these are the types of all the properties identified by rateLimitOf.
- Unit cost types: For all properties that are 1) directly or indirectly in owned atomic nodes, 2) stereotyped by UnitCost, and 3) subset of *flowingCommodity*, these are the types of all the properties identified by unitCostOf.

The translation in this subsection applies when the following conditions are met:

- Consumption types are the same as production types.
- Production types are disjoint with each other (no commodity is of one more than one type).
- Flow rate types are also production types.
- Flow rate limit types are also production types.
- Unit cost types are also production types.

The translation ignores stereotype applications that are not listed above, such as:

- consumption rates, flow rate limits, unit costs on commodity types that are not also production types,
- rates applied to the *flowingCommodity* property in nodes,
- rate limits applied to properties subsetting *input* or *output* and not subsetting *consumedInput* or *producedOutput*,
- unit costs in nodes.

2.4.2 Commodity Flow Networks and Multi-Commodity Flow Network optimization

This subsection gives the correspondence between CFN and MCFO terms.

2.4.2.1 Translating nodes

Composite networks are supported by *flattening* the network, so that 1 or more nodes in the SysML model correspond to one node in the analysis model, and 0 or more nodes and one edge in the SysML model correspond to one edge on the analysis model. Composite networks must be connected through edges between ports, which are also nodes.

2.4.2.2 Selecting rate values

For most properties in the CFN model, values are obtained using the regular default/initial value mechanism specified in SysML. One notable exception is for rate properties.

The rate property values used in translation are selected in the following ways, depending on whether it has a corresponding "initial value" slot:

- With initial value slot that:
  - has EventDistribution applied, the selected rates are the inverses of the event distribution's mean;
  - does not have EventDistribution applied, the selected rates are the values of the slot.
- Without initial value slot, if the rate property:
  - has EventDistribution applied, the selected rates are the mean rates of that distribution;
  - does not have EventDistribution applied, the selected rate is the default value of the slot.

The above method is applied to obtain values of *production rates* and *consumption rates*.

2.4.3 AMPL

An AMPL problem is specified in a model file and a data file. A model file declares variables, constraints, and objective functions, while a data file assign values to the variables. The same model file is used for all translations as shown in the following code and provided to an AMPL solver.

```
set NODES;
set EDGES within (NODES cross NODES);
set COMMODITIES;

param duration default 1;

param supply {NODES,COMMODITIES} >= 0;
param demand {NODES,COMMODITIES} >= 0;

check {k in COMMODITIES}: sum {i in NODES} supply[i,k]
  = sum {j in NODES} demand[j,k];

param ucost {EDGES,COMMODITIES} >= 0;
param edge_limit {EDGES,COMMODITIES} >= 0;
param total_edge_limit {EDGES} >= 0;
param total_node_limit {NODES} >= 0;

var Values {EDGES,COMMODITIES}  >= 0;

param node_fcost {NODES} >= 0;
var NodeUse {NODES} binary;

param edge_fcost {EDGES} >= 0;
var EdgeUse {EDGES} binary;

minimize TotalCost:
   sum {(i,j) in EDGES, k in COMMODITIES} ucost[i,j,k]
     * Values[i,j,k] * duration
 + sum {(i,j) in EDGES} edge_fcost[i,j] * EdgeUse[i,j]
 + sum {i in NODES} node_fcost[i] * NodeUse[i];

subject to BalanceConstraint {i in NODES, k in COMMODITIES}:
   supply[i,k] + sum {(j,i) in EDGES} Values[j,i,k]
      = demand[i,k] + sum {(i,j) in EDGES} Values[i,j,k];

subject to EdgeLimitConstraint {(i,j) in EDGES, k in COMMODITIES}:
   Values[i,j,k] <= edge_limit[i,j,k];

subject to TotalEdgeLimitConstraint {(i,j) in EDGES}:
   sum {k in COMMODITIES} Values[i,j,k]
     <= total_edge_limit[i,j] * EdgeUse[i,j];

subject to TotalNodeLimitConstraint {i in NODES}:
   sum {(i,j) in EDGES, k in COMMODITIES} Values[i,j,k]
 + sum {k in COMMODITIES} demand[i,k]
```

```
<= total_node_limit[i] * NodeUse[i];
end;
```

In this AMPL model:

- `NODES`, `EDGES`, and `COMMODITIES` correspond to *node*s, *edge*s, and specializations of *Commodity*, respectively, see Subsection 2.2.1. `EDGES` are pairs of `NODES`;
- `duration` correspond to the length of time over which the objective function is calculated;
- `supply` and `demand` correspond to node production and consumption rates respectively, see Rate in Subsection 2.2.2 as it applies to *producedOutputs* and *consumedOutputs* of *AtomicFlowNode*s. They associate a non-negative real or integer to a pair of nodes and commodities;
- a constraint states that the total `demand` of all nodes combined must be equal to the total `supply` for all nodes combined;
- `ucost` and `edge_limit` correspond respectively to the unit cost and flow rate limit for each kind of commodity flowing along each edge, see Cost and RateLimit in Subsection 2.2.2 as they apply to *FlowNetworkLink*s. They associate a non-negative real or integer to a pair of edge and commodity;
- `total_edge_limit` and `total_node_limit` correspond to the total flow rate limits for edges and nodes respectively, see *flowRateLimit* in Subsection 2.2.1;
- `Values` correspond to flow rates on each edge for each kind of commodity, see Rate in Subsection 2.2.2 as it applies to *flowingCommodity* in *FlownetworkLink*s. These rates are the result of MCFO;
- `node_fcost` and `edge_fcost` correspond to the fixed cost associated with nodes and edges, respectively, see *fixedCost* on *FlowNetworkElementWithCost* in Subsection 2.2.1, as specialized by specializations of *AtomicFlowNode*s and *FlowNetworkLink*s;
- `NodeUse` and `EdgeUse` are binary variables indicating whether any commodities flow in a node or edge, equal to 1 if any do, 0 if none do;
- `TotalCost` is the objective function to minimize, defined as the sum of all flow rates multiplied by the duration and unit cost on all edges for all commodities, plus fixed costs for all used nodes and edges;
- `BalanceConstraint` states that for each commodity within each node, the consumption rate plus the sum of all the node's outgoing flow rates must be equal to the production rate plus the sum of all the node's incoming flow rates. In other words, what is removed from the node's stock (consumed or shipped) must be equal to what is added to the node's stock (produced or received);
- `EdgeLimitConstraint` states that for each commodity in each edge, the flow rate must be less than or equal to the maximum flow rate;

- `TotalEdgeLimitConstraint` states that in each edge, the total flow rates must be less than or equal to the edge's maximum total flow rate if the edge is in use (EdgeUse=1 for that edge), or 0 if the edge is not in use (EdgeUse=0 for that edge). This constraint forces the value of EdgeUse to be 1 if any commodity flow is allowed through, knowing that the value of EdgeUse will be 0 otherwise (in order to minimize the objective function);

- `TotalNodeLimitConstraint` states that in each node, the total consumption plus the total outgoing flow rates must be less than or equal to the node's maximum total flow rate if the node is in use (NodeUse=1 for that node), or 0 if the node is not in use (NodeUse=0 for that node). Note that it's possible to replace the total consumption plus the total outgoing flow rates by the total production plus the total incoming flow rates, since these have the same value due to `BalanceConstraint`.

The data file shown in the following code is the pattern for giving values to parameters defined in the model previously shown:

```
data;
set NODES := ...;
set LINKS := (...,...) ...;
set COMMODITIES := ...;
param supply default ... :=   ... ;
param demand default ... :=  ... ;
param max_total_node_cap default ... := ... ;
param max_total_edge_cap default ... := ... ;
param max_edge_cap default ...  := ... ;
param ucost     :=    ... ;
param node_fcost default ... := ... ;
param edge_fcost default ... := ... ;
```

Production and consumption rates are selected from the model as described in 2.4.2.2.

2.4.4 MATLAB

Translation into MATLAB involves creating vectors and matrices that are input to LP solvers. AMPL solvers do this internally.

Linear optimization toolbox typically offer various kinds of solvers, optimized for a given data set. The regular LP solver is used for translation, unless some of the unknown variables are integers. In that case, a Mixed-Integer Linear Programming (MILP) solver is used, and an additional vector indicates which variables are integers. This ensures unknown variables can only change by increment, and cannot have fractional values.

LP problems have the following form:

$$
\begin{array}{rrcl}
\text{Minimize:} & C \cdot \mathbf{x} & & \\
\text{Subject to:} & Aineq \cdot \mathbf{x} & \leqslant & bineq \\
& Aeq \cdot \mathbf{x} & = & beq \\
& lb & \leqslant \mathbf{x} \leqslant & ub
\end{array}
$$

In which:

- x is a row vector for the (unknown) variables,
- C is a column vector for the optimal function to minimize,
- $Aineq$ is a matrix for the inequality constraints,
- $bineq$ is a column vector for the inequality constraints,
- Aeq is a matrix for the equality constraints,
- beq is a column vector for the equality constraints,
- lb is a column vector for the lower bound of the unknown variables,
- ub is a column vector for the upper bound of the unknown variables.

MCFO problems are transformed to LP problems as follows:

- Row vector x is made of columns for:
  - *flow rate variables* for each commodity kind allowed through each edge: this is a number that will correspond to the flow rate for that commodity in that edge;
  - *node usage variables* for each node with a fixed cost: this is a binary variable equal to 1 when that node is used, and 0 if not;
  - *edge usage variables* for each edge with a fixed cost: this is a binary variable equal to 1 when that edge is used, and 0 if not;
- Column vector C is made of rows indicating:
  - the product of the duration and the unit cost associated with the corresponding flow rate in x;
  - the fixed cost associated with the corresponding node usage in x;
  - the fixed cost associated with the corresponding edge usage in x;
- The lower bounds vector has rows equal to 0 for every variable x;
- The upper bounds vector has rows equal to:
  - for flow rate variables, the maximum flow rate of the corresponding commodity kind in the edge. If none is indicated, it can be set to the total production or consumption rate in the network for that commodity;
  - for node and edge usage variables, the maximum is 1;
- Equality constraint matrix Aeq and its vector beq is made of the following:
  - One row for each commodity kind in each node, in which:

    - in the matrix Aeq, the column corresponding to every incoming edge for that node and commodity is set to 1, the column corresponding to every outgoing edge for that node and commodity is set to -1, and all other columns are set to 0;
    - in the matrix beq, the value is the node's consumption rate of that commodity minus the node's production rate of that commodity;

- Inequality constraint matrix $Aineq$ and its vector $bineq$ is made of the following:
  - One row for each node with fixed cost or maximum total flow rate, in which:
    - if the node has a fixed cost:
      - in the matrix $Aineq$, the column corresponding to every outgoing edge is set to 1, the column corresponding to the node usage is set to minus the maximum total flow rate for that node (if absent, this value is the sum of all production rates from every node), and all other columns are set to 0;
      - in the matrix beq, the value is minus the sum of all that node's consumption rates;
    - if the node does not have a fixed cost:
      - in the matrix $Aineq$, the column corresponding to every outgoing edge for that node and commodity is set to 1, and all other columns are set to 0;
      - in the matrix beq, the value is the node's maximum total flow rate for that node (if absent, this value is the sum of all production rates from every node) minus the sum of all that node's consumption rates;
  - One row for each edge with fixed cost or total flow rate, in which:
    - if the edge has a fixed cost:
      - in the matrix $Aineq$, the columns corresponding to that edge (one per commodity allowed through) is set to 1, the column corresponding to the edge usage is set to minus the maximum total flow rate for that edge (if absent, this value is the sum of all production rates from every node), and all other columns are set to 0;
      - in the matrix beq, the value is 0;
    - if the edge does not have a fixed cost:
      - in the matrix $Aineq$, the columns corresponding to that edge (one per commodity allowed through) is set to 1, and all other columns are set to 0;
      - in the matrix beq, the value is the edge's maximum total flow rate.

Production and consumption rates are selected from the model as described in 2.4.2.2.

2.5 Translation and optimization of Commodity Flow Network example

This subsection shows how the example of commodity flow network in Subsection 2.3 is translated and optimized in AMPL and MATLAB [4].

2.5.1 AMPL representation

In the proposed translation, the AMPL model file stays the same (see Subsection 2.4.3) and the data file is generated from the SysML model.

The following code shows the assignment of values for the nodes, edges, and commodities in the data file (4 nodes, 12 edges, and 2 kinds of commodities).

```
data;
set NODES := miami omaha la nyc;
set EDGES := (omaha,miami) (la,miami) (la,omaha) (omaha,la)
  (miami,la) (miami,omaha) (omaha,nyc) (nyc,omaha) (nyc,miami)
  (miami,nyc) (la,nyc) (nyc,la);
set COMMODITIES := Shoes Shirt;
```

The following code shows the assignment of values for the supply, which is set to 0 by default, with values provided for the nodes that produce commodities: `miami` produces 350 `shirts`, and `la` produces 250 `shoes`.

```
param supply default 0 :=
miami Shirt            350
la Shoes               250
;
```

The following code shows the assignment of values for the demand, which is set to 0 by default, with values provided for the nodes that consume commodities: `la` consumes 350 `shirts`, and `nyc` consumes 250 `shoes`.

```
param demand default 0 :=
la Shirt               350
nyc Shoes              250
;
```

The following code shows the assignment of values for the maximum total flow rate for nodes, which is set to 600 by default, with a value of 500 provided for `omaha`.

```
param max_total_node_cap default 600 :=
omaha                               500
;
```

The following code shows the assignment of values for the maximum total flow rate for edges, which is set to 500 by default, and there is no exception.

```
param max_total_edge_cap default 500 :=
;
```

The following code shows the assignment of values for the maximum total flow rate for each commodity in each edge, which is set to 600 by default, and set to 325 for shirts and 225 for shoes.

```
param edge_limit default 600.0  :=
omaha miami Shoes              225
omaha miami Shirt              325
la miami Shoes                 225
la miami Shirt                 325
la omaha Shoes                 225
la omaha Shirt                 325
omaha la Shoes                 225
omaha la Shirt                 325
miami la Shoes                 225
miami la Shirt                 325
miami omaha Shoes              225
miami omaha Shirt              325
omaha nyc Shoes                225
omaha nyc Shirt                325
nyc omaha Shoes                225
nyc omaha Shirt                325
nyc miami Shoes                225
nyc miami Shirt                325
miami nyc Shoes                225
miami nyc Shirt                325
la nyc Shoes                   225
la nyc Shirt                   325
nyc la Shoes                   225
nyc la Shirt                   325
;
```

The following code shows the assignment of values for the unit cost for each commodity in each edge.

```
param ucost        :=
omaha miami Shoes 50.0
omaha miami Shirt 25.0
la miami Shoes    78.0
la miami Shirt    39.0
la omaha Shoes    46.0
la omaha Shirt    23.0
omaha la Shoes    46.0
omaha la Shirt    23.0
miami la Shoes    78.0
miami la Shirt    39.0
miami omaha Shoes 50.0
miami omaha Shirt 25.0
omaha nyc Shoes   40.0
omaha nyc Shirt   20.0
nyc omaha Shoes   40.0
nyc omaha Shirt   20.0
nyc miami Shoes   38.0
nyc miami Shirt   19.0
miami nyc Shoes   38.0
miami nyc Shirt   19.0
la nyc Shoes      84.0
la nyc Shirt      42.0
```

```
nyc la Shoes      84.0
nyc la Shirt      42.0
;
```

The following code shows the assignment of values for the fixed cost for each node.

```
param node_fcost default 2000.0 :=
omaha                           1000.0
la                              8000.0
nyc                             4000.0
;
```

The following code shows the assignment of values for the fixed cost for each edge.

```
param edge_fcost default 1000.0 :=
la miami                        16000.0
la omaha                        8000.0
miami la                        4000.0
miami omaha                     2000.0
nyc omaha                       4000.0
nyc miami                       4000.0
miami nyc                       2000.0
la nyc                          16000.0
nyc la                          8000.0
;
```

The following code shows the optimal total cost when giving the model and data to an AMPL solver such as GNU Linear Programming Kit (GLPK).

```
Objective:  TotalCost = 81925 (MINimum)
```

2.5.2 MATLAB representation

The MATLAB code has a very different structure. It is lower-level, since all matrices corresponding to the optimization problem need to be constructed.

The unknown variables are organized as follows. For each edge, there is 1 variable corresponding to the edge usage, and 2 variables corresponding to the 2 commodity flow rates within the edge. Since there are 12 edges, 36 variables are obtained. Then, for each node, there is 1 variable corresponding to the node usage. Since there are 4 nodes, there are 4 variables. The total number of variables is 40.

The following code shows the costs for the objective function in MATLAB.

```
f = [1000.0 50.0 25.0 16000.0 78.0 39.0 8000.0 46.0 23.0 1000.0 46.0 23.0 4000.0 78.0\
     39.0 2000.0 50.0 25.0 1000.0 40.0 20.0 4000.0 40.0 20.0 4000.0 38.0 19.0 2000.0\
     38.0 19.0 16000.0 84.0 42.0 8000.0 84.0 42.0 2000.0 1000.0 8000.0 4000.0];
```

Since there are 40 variables, there are 40 associated costs. The association follows the same order as the variables, with values for edge fixed costs (1st, 4th, 7th, etc.), values for unit costs (2nd, 3rd,

5th, 6th, 8th, 9th, etc.), and at the end values for node fixed cost (37th, 38th, 39th, 40th).

The following code shows the lower bounds for the variables, which are set to 0.

```
lb = zeros(1, 40);
```

The following code shows the upper bounds for the variables, which are set to 1 for edge usage variables (1st, 4th, 7th, etc.) and node usage variables (37th, 38th, 39th, 40th), and to 225 or 325 corresponding to the maximum flow rate associated with the flow rate.

```
ub = [1.0 225.0 325.0 1.0 225.0 325.0 1.0 225.0 325.0 1.0 225.0 325.0 1.0 225.0 325.0\
      1.0 225.0 325.0 1.0 225.0 325.0 1.0 225.0 325.0 1.0 225.0 325.0 1.0 225.0 325.0\
      1.0 225.0 325.0 1.0 225.0 325.0 1.0 1.0 1.0 1.0];
```

The following code shows the inequality matrix and vector for edges only, with one row for each edge (vector values are on separate rows, but shown horizontally for compactness):

```
Aineq = [
-500.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 -500.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0\
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 -500.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0\
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -500.0 1.0 1.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -500.0 1.0 1.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -500.0 \
  1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 -500.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 -500.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 -500.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -500.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -500.0 1.0 1.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
```

```
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  -500.0 1.0 1.0 0.0 0.0 0.0 0.0
...];
bineq = [0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ...];
```

For each row (corresponding to an edge) above:

- the value of the column corresponding to the fixed variable is the negative of the total capacity limit of the edge. For example, in the first row, the 1st column corresponds to the fixed variable and has a value of -500, the negative of the total capacity of that edge;
- the values of the columns corresponding to the flow rates of the edge are 1. For example, in the first row these are the 2nd and 3rd columns;
- the value in the vector is set to 0.

The following code shows the inequality matrix and vector for nodes only, with one row for each node (vector values are on separate rows, but shown horizontally for compactness):

```
Aineq = [...
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 \
  1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 -600.0 0.0 0.0 0.0
0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 -500.0 0.0 0.0
0.0 0.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 \
  0.0 0.0 0.0 0.0 0.0 -600.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  1.0 1.0 0.0 0.0 0.0 -600.0];
bineq = [...  -0.0 -0.0 -350.0 -250.0];
```

and the value in the vector is the negative of the the total consumption rate of that node (-350 for `la`, -250 for `nyc`, 0 otherwise).

The equality matrix and vectors are then defined.

For each commodity in each node, columns corresponding to a flow rate of that commodity in an incoming flow have a value of 1, columns corresponding to a flow rate of that commodity in an outgoing flow have a value of -1, and the vector has a value of the node's consumption rate of that commodity minus that node's production rate of that commodity.

For each row (corresponding to a node) above:

- the value of the column corresponding to the fixed variable is the negative of the total capacity limit of the node. For example, in the first row, the 37th column corresponds to the fixed variable and has a value of -600, the negative of the total capacity of that node;

- the values of columns corresponding to the outgoing flow rates from that node are 1. For example, in the first row these are the 11th and 12th columns;
- the value in the vector is the negative of the total consumption rate of that node. For example, in the third row, corresponding to `la`, the value is -350.

The following code shows the equations for nodes, there is one row for each commodity in each node.

```
Aeq = [
0.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 0.0 0.0 \
  -1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 -1.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 0.0 \
  0.0 -1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 -1.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 -1.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 -1.0 0.0 0.0 0.0 0.0 0.0 \
  1.0 0.0 0.0 -1.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 -1.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 -1.0 0.0 0.0 0.0 0.0 \
  0.0 1.0 0.0 0.0 -1.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 -1.0 0.0 0.0 -1.0 0.0 0.0 1.0 0.0 0.0 1.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 \
  0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 -1.0 0.0 0.0 -1.0 0.0 0.0 1.0 0.0 0.0 1.0 0.0 \
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  -1.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 1.0 0.0 0.0 -1.0 0.0 0.0 -1.0 0.0 0.0 1.0 0.0 0.0 \
  1.0 0.0 0.0 -1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 \
  0.0 0.0 0.0 0.0 1.0 0.0 0.0 -1.0 0.0 0.0 -1.0 0.0 0.0 1.0 0.0 \
  0.0 1.0 0.0 0.0 -1.0 0.0 0.0 0.0 0.0];
beq = [0.0 -350.0 0.0 0.0 -250.0 350.0 250.0 0.0];
```

Next, it is necessary to indicate which of the unknown variables are integers. The following code shows the vector that contains the indices of those integer variables.

```
intcon = [1 4 7 8 10 12 13 15 16 18 19 20 22 25 28 31 32 34 37 38 39 40]
```

Finally, the following code shows the solver call with that defined data (the MILP solver from MATLAB's Optimization toolbox is used here).

```
[sol, fval, flag, output] = intlinprog (f, intcon, Aineq, bineq, Aeq, beq, lb, ub);
fprintf('Values = %f\n', fval);
```

The following code shows the result provided by the solver. It is the same as obtained in Subsection 2.5.1.

```
Values = 81925.000000
```

3 Processing Networks

This section presents Processing Networks (PNs) and translations between extended SysML models on one side, and Queuing Analysis (QA) and Discrete Event Simulation (DES) tools on the other. Subsection 3.1 reviews PN and analysis. Subsection 3.2 defines an extension of SysML for PN, while Subsections 3.4 and 3.6 give translations between extended SysML models and QA and DES models, respectively.

3.1 Modeling and analysis

PNs are specialized Commodity Flow Networks (CFNs) that introduce behaviors in nodes to *queue* commodities then *service* them, including to add value by modifying them.[1] It is assumed that commodities entering processing nodes or being produced by them instantly move to the queue, and that after service completion, commodities instantly leave processing nodes or are consumed by them. Processing nodes characterize these by:

- Queue size: The maximum number of commodities a queue can hold. It is often taken as infinity by default. Commodities coming in while the queue is full are typically rejected.
- Queuing discipline: Includes where incoming commodities are placed in the queue, which one is pulled from the queue when a server becomes available, and whether commodities being served can be replaced by newer or higher-priority incoming commodities (preemption). The most common types of discipline are First-In First-Out (FIFO), Last-In First-Out (LIFO), priority queues, and LIFO queues with preemptive resume.
- Service time or rate: The time taken to service a single commodity (service time, τ) or the number of commodities served per time (service rate, μ). Service time is the inverse of service rate.
- Number of servers: A single node can have multiple servers processing commodities at the same time.

Some statistics are useful when commodities are moved or served at randomly distributed times, such as these for every node:

- Utilization ratio: Average percentage of time that servers are busy taking care of a commodity.
- Response time: Average duration a commodity spends in a node.
- Average number in node: Average number of commodities in a node.
- Average number in queue: Average number of commodities in a node queue.

[1] Servicing that does not modify commodities, such as moving and storing, can happen at nodes in between others that do [3][5].

- Throughput: Average flow rate of commodities in a node.

These statistics can be obtained in two ways:

- QA: Applying mathematical formulas to find the values these statistics converge to over time.
- DES: Running a simulation model of the network to find values over a given amount of time.

QA requires strict conditions (such as unlimited queue size, exponential distributions for rates and service times, particular queuing disciplines), but provides exact values, while DES can cover a broader range of networks, but yields values that might differ at every run. In situations where QA can be used, DES values should converge over simulated time towards QA results.

Translations of PN to QA and DES are given in Subsections 3.4 and 3.6, respectively.

3.2 SysML extension

The SysML representation of PNs and analysis consists of a:

- *library* for modeling PNs as described in Subsection 3.1, see Subsection 3.2.1. It specializes the CFN library in Subsection 2.2.1;
- *profile* to add analysis information necessary to perform analysis as described in Subsection 3.1, see Subsection 3.2.2. It defines stereotypes applied to value properties that carry analysis information.

Modelers use the extension by specializing library blocks and properties, then applying stereotypes as needed for translation to analysis tools, as described in Subsections 3.4 and 3.6. See Appendix B for case and font conventions for model and analysis tool element names and Appendix C for an introduction to SysML.

3.2.1 Library

Figure 7 shows the PN library. It defines elements: *ProcessingNetwork*, *AtomicProcessingNode*, and *QueueDiscipline*.

*ProcessingNetwork*s are specializations of *FlowNetwork*s that can contain *AtomicProcessingNode*s, which are specializations of *AtomicFlowNode* that introduce properties about queuing and serving commodities: *concurrentServingCapacity* indicates how many commodities can be served at a time, *storageCapacity* indicates how many commodities awaiting service can be queued at a time, *discipline* indicates the order in which commodities go in and out of the queue, and *preemptiveResume* indicates whether incoming commodities with a higher priority can interrupt commodities being served (service of preempted commodities resume where they were interrupted later on). Queue disciplines can be First In First Out (FIFO), Last In First Out (LIFO), or based on priority.

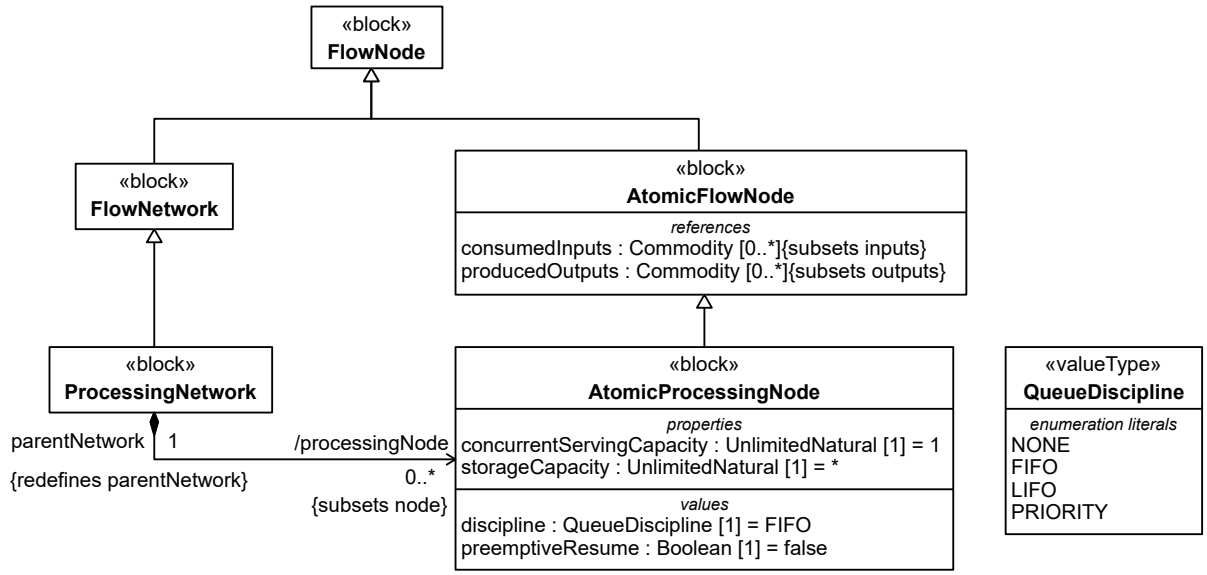

Figure 7: Processing Network Library

3.2.2 Profile

The PN profile contains stereotypes for queuing analysis-related information. The ServiceTime stereotype is for parameters of analysis, while the others are for results of analysis. The ServiceTime, Utilization, ResponseTime, AverageNumberInNode, and AverageNumberInQueue stereotypes apply to value properties that give this information, identifying other properties for each kind of commodity that the information is about via properties serviceTimeOf, utilizationOf, responseTimeOf, averageNumberInNodeOf, and averageNumberInQueueOf, respectively. When these properties can have multiple values, they are ordered and have the same number of values, with each stereotyped property value about the commodity property in the same position. The stereotypes are described in the subsequent paragraphs.

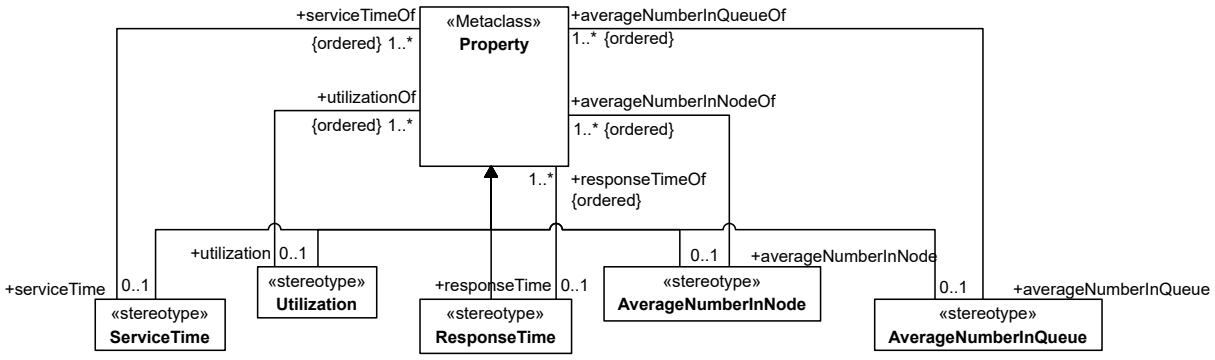

Figure 8: Processing Network Profile

The AverageNumberInNode stereotype applies to properties giving the average number of com-

modities in the node (in the queue or being serviced), as identified by the averageNumberInNodeOf property. The lower and upper multiplicity must be the same as the number of averageNumberInNodeOf values. It must be ordered and non-unique when the upper multiplicity is more than 1. Its type must be *Real* or one of its specializations.

The AverageNumberInQueue stereotype applies to properties giving the average number of commodities in the queue, as identified by the averageNumberInQueueOf property. The lower and upper multiplicity must be the same as the number of averageNumberInQueueOf values. It must be ordered and non-unique when the upper multiplicity bound is more than 1. Its type must be *Real* or one of its specializations.

The ResponseTime stereotype applies to properties giving the sum of the average time a commodity identified by the responseTimeOf property spends in a queue, and the average time it spends being served. The lower and upper multiplicities of the stereotyped property are the same as the number of responseTimeOf values. It must be ordered and non-unique if the upper multiplicity is more than 1. Its type must be *Real* or one of its specializations.

The ServiceTime stereotype applies to properties giving an average service time to flowing commodities identified by the serviceTimeOf property. The lower and upper multiplicity must be the same as the number of serviceTimeOf values. The stereotyped property must be ordered and non-unique when the upper multiplicity is more than 1. Its type must be *Integer*, *Real*, or one of their specializations.

The Utilization stereotype applies to properties giving node utilization to flowing commodities identified by the utilizationOf property. The number is the percentage of time that the servers are busy serving a given kind of commodity. The lower multiplicity and upper multiplicity must be the same as the number of utilizationOf values. The stereotyped property must be ordered and non-unique when the upper multiplicity is more than 1. Its type is *Real* or one of its specializations.

3.3 Processing Networks example

The examples for queuing analysis are variations of a simple processing network with two atomic processing nodes and two regular (non-processing) nodes. The first variation defines a single kind of commodity, while the other defines two. Figure 9 shows blocks for the variation that uses one kind of commodity. *CustomCommodity* is a specialization of *Commodity*. *CustomFlowNetworkElement* is a specialization of *FlowNetworkElement*, and it has a property *flowingCustomCommodity* that subsets *flowingCommodity* called and is typed by *CustomCommodity*. *AtomicCustomFlowNode* is a specialization of both *CustomFlowNetworkElement* and *AtomicFlowNode*. *CustomFlowNetworkLink* is an association between two *AtomicCustomFlowNode*s, and it specializes both *CustomFlowNetworkElement* and *FlowNetworkLink*. It also has a *rateAcross* property with the Rate stereotype applied that refers to *flowingCustomCommodity*. Finally, *AtomicCustomProcessingNode* is a specialization of both *AtomicProcessingNode* and *AtomicCustomFlowNode*. It has a subset of *producedOutputs* called *producedCustomCommodity*, a subset of *consumedInputs* called *consummedCustomCommodity*, a *rates* property stereotyped by Rate and referring to the previous two properties, and finally a *serviceTime* property stereotyped by ServiceTime and referring to *flowingCustomCommodity*.

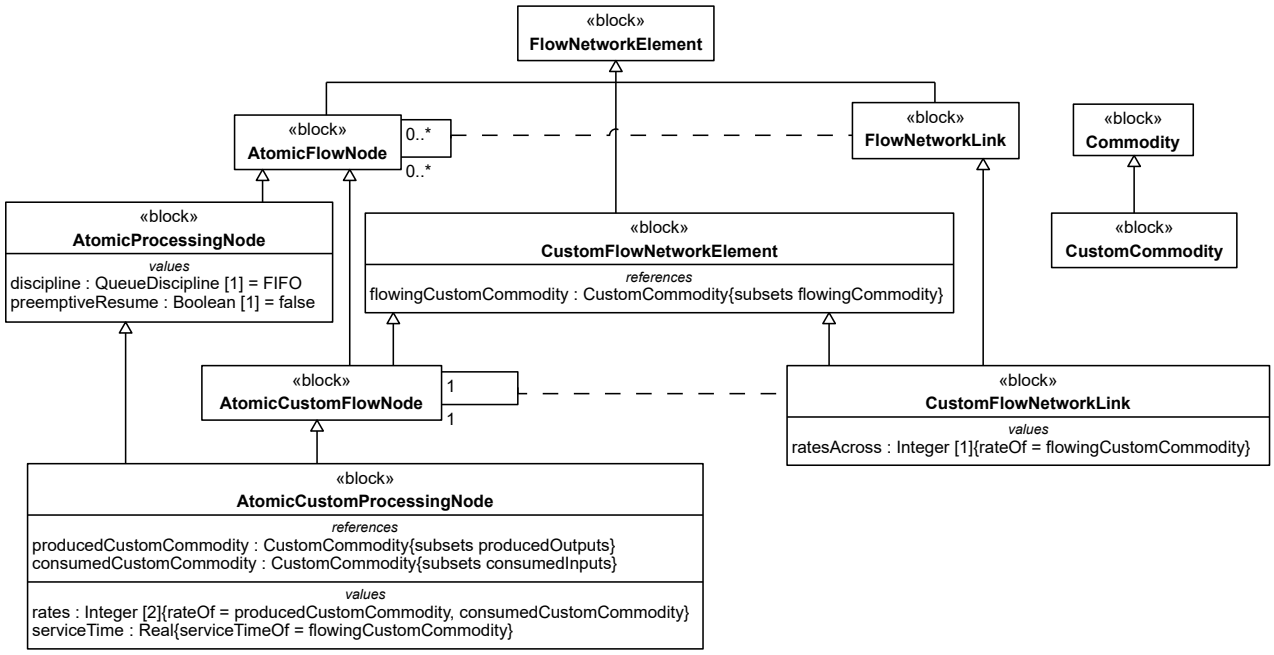

Figure 9: Processing Network Example, single commodity, bdd

CustomFlowNetwork specializes *FlowNetwork* to introduce the internal structure in Figure 10. This block has 2 properties typed by *AtomicCustomFlowNode* (*cfn1* and *cfn2*) as well as two properties typed by *AtomicCustomProcessingNode* (*apn1* and *apn2*). The *apn1* property has initial values of 1 for *concurrentProcessCapacity*, * for *storageCapacity*, 0.018 for *serviceTime*, and the two values 20 and 0 for *rates*. The *apn2* property has initial values of 2 for *concurrentProcessCapacity*, * for *storageCapacity*, 0.036 for *serviceTime*, and the two values 0 and 20 for *rates*. *CustomFlowNetwork* also has 4 connectors with initial values for the flow rate. The flow rate is 20 from *cfn1* to *apn1*, 50 from *apn1* to *apn2*, 10 from *apn2* to *apn1*, and 20 from *apn2* to *cfn2*.

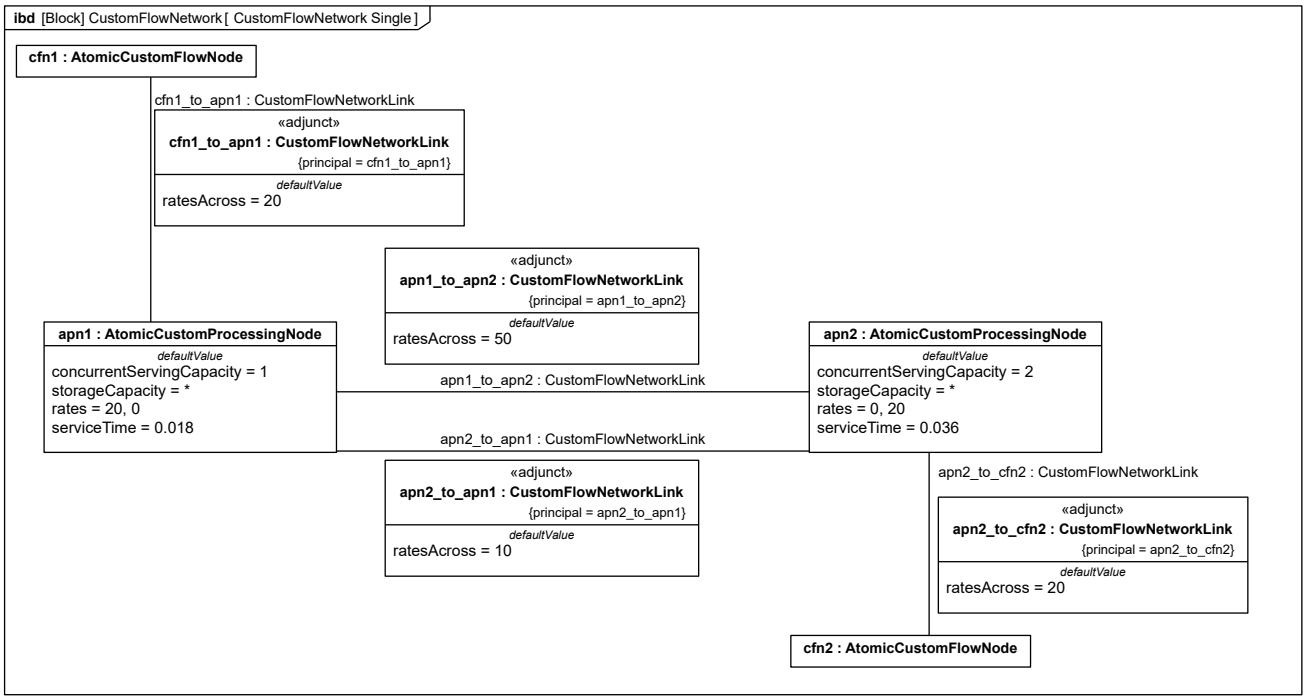

Figure 10: Processing Network Example, single commodity, ibd

The variation of the example that defines two kinds of commodities is similar to the first, with the exception of the following:

- two kinds of commodities flow in the network,
- the concurrent process capacity on all nodes is set to 1,
- the service times on all nodes are readjusted so that the various commodity numbers from the previous example are split 60/40 between two commodities,
- the queue discipline on all nodes is changed to LIFO and preemptive resume is enabled.

These changes are necessary to meet the queuing analysis requirements with multiple commodities.

Figure 11 shows blocks in the second variation. Compared to Figure 9, *Commodity* has two specializations named *CustomCommodityA* and *CustomCommodityB* . *CustomFlowNetworkElement* has two property named *flowingCustomCommodityA* and *flowingCustomCommodityB* that subset *flowingCommodity*, and they are respectively typed by the two previous commodity specializations. The *flowRate* property of *CustomFlowNetworkLink* refers to these two subsets and is of multiplicity 2. Finally, in *AtomicCustomProcessingNode*, there are two subsets of *producedOutputs* called *producedCustomCommodityA* and *producedCustomCommodityA*, as well as two subsets of *consumedInputs* called *consumedCustomCommodity* and *consumedCustomCommodityB*. The *rates* property refers to the previous four properties and is of multiplicity 4. Finally the *serviceTime* property refers to the two flowing commodities and is of multiplicity 2.

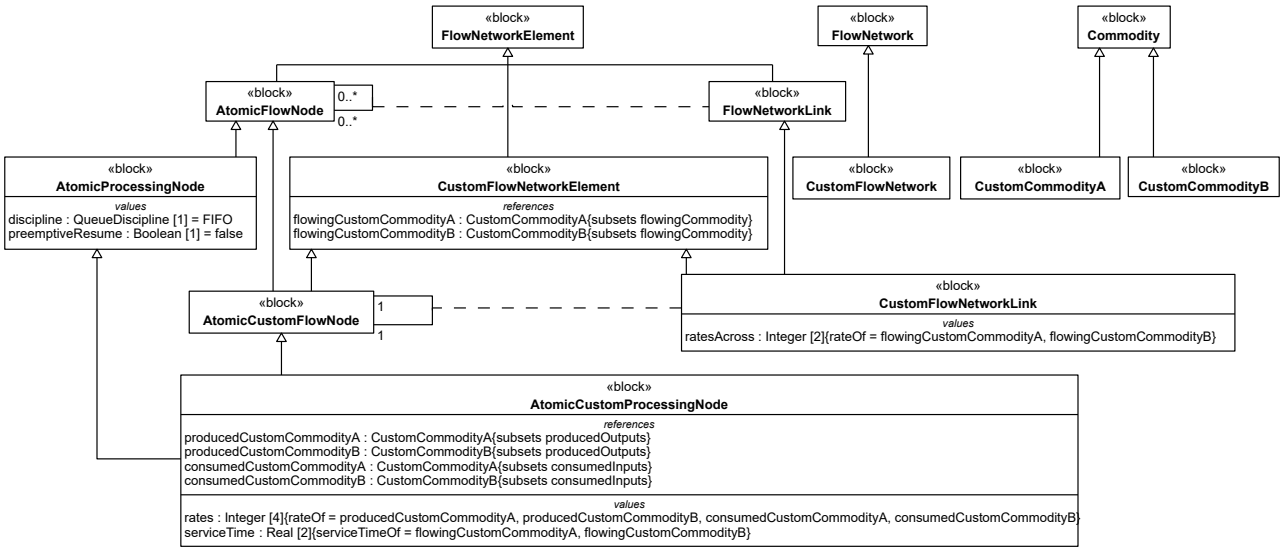

Figure 11: Processing Network Example, multiple commodities, bdd

The internal structure of *CustomFlowNetwork* shown in Figure 12 gives the *apn1* property initial values of 1 for *concurrentProcessCapacity*, * for *storageCapacity*, 0.018 and 0.018 for *serviceTime*, and the 12, 8, 0 and 0 for *rates*. The *apn2* property has initial values of 1 for *concurrentProcess-Capacity*, * for *storageCapacity*, 0.02 and 0.015 for *serviceTime*, and 0, 0, 12 and 8 for *rates*. Regarding the connectors in *CustomFlowNetwork*, the flow rates are 12 and 8 from *cfn1* to *apn1*, 30 and 20 from *apn1* to *apn2*, 6 and 4 from *apn2* to *apn1*, and 12 and 8 from *apn2* to *cfn2*.

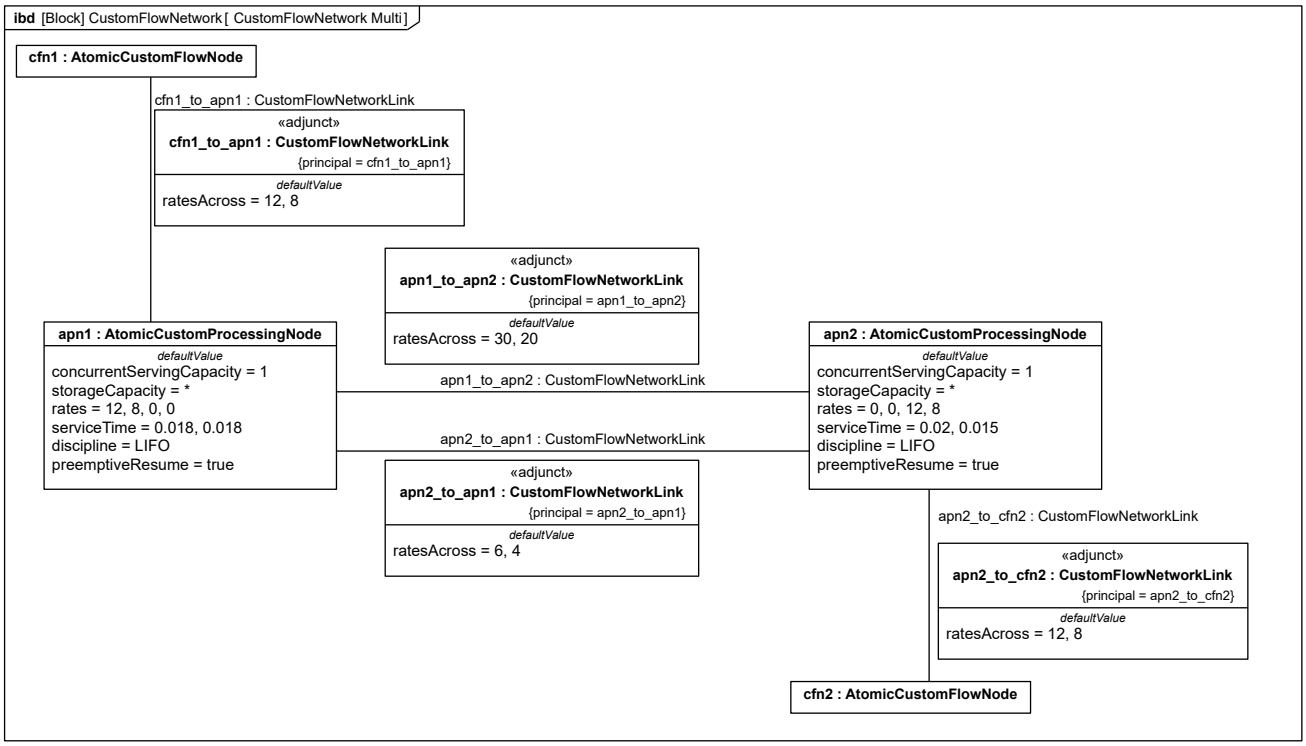

Figure 12: Processing Network Example, multiple commodities, ibd

3.4 Translating between Processing Networks and queuing analysis tools

This subsection gives translations between SysML models extended for PNs, see Subsection 3.2, and Octave and MATLAB to model Queuing Networks (QNs), consisting of matrices used for the calculation of QA. MATLAB is a general-purpose language for numerical analysis that includes QA, while Octave is open-source software for scientific computations, with a language very similar to MATLAB [6].

3.4.1 Additional SysML model constraints for translation

The translations in Subsections 3.4.3 and 3.4.4 require SysML models to meet additional constraints, expressed with the following terms defined for atomic processing nodes:

- Production rate properties: All the properties that are 1) stereotyped by Rate, and 2) subset of *producedOutputs*.
- Production types: For production rate properties, the types of all the properties identified by rateOf.
- Consumption rate properties: All the properties that are 1) stereotyped by Rate, and 2) subset of *consumedInputs*.

- Consumption types: For all consumption rate properties, the types of all the properties identified by rateOf.
- Incoming rate properties: All the properties in the type of all incoming connectors that are 1) stereotyped by Rate, and 2) subset of *flowingCommodity*.
- Incoming types: For all incoming rate properties, the types of all the properties identified by rateOf.
- Outgoing rate properties: All the properties in the type of all outgoing connectors that are 1) stereotyped by Rate, and 2) subset of *flowingCommodity*.
- Outgoing types: For all outgoing rate properties, the types of all the properties identified by rateOf.
- Service time properties: All the properties that are 1) stereotyped by ServiceTime, and 2) subset of *flowingCommodity*.
- Service types: For all service time properties, the types of all the properties identified by serviceTimeOf.

The translation specified in this subsection applies only when the following conditions are met:

- The production types and incoming types combined are the same as the production types and outgoing types combined and the same as the service types.
- The production types and incoming types are all disjoint
- If the number of types in the bullet above is 1, the queue discipline must be FIFO. Otherwise, the queue discipline must be LIFO and *preemptiveResume* must be true.
- Properties and slots for production rate, consumption rate, flow rate, and service time must be stereotyped by ExponentialDistribution or have no event distribution stereotype applied (only if the translator is configured to assume exponential distributions when no event distribution is specified).

3.4.2 Processing Networks and Queuing Analysis models

This subsection presents how the PN terms correspond to QA terms.

3.4.2.1 Translating nodes

PNs are more expressive than QNs because PNs can include CFN nodes, which do not have queues/servers, while QNs nodes all have queues and servers (queuing nodes). Only processing nodes correspond to queuing nodes, but the flows between processing nodes and non-processing nodes affect the behavior of queuing nodes.

Composite networks are supported by *flattening* the network, so that a tuple of 1 or more nodes in the SysML model corresponds to one node in the analysis model, and a tuple of 0 or more nodes plus one edge in the SysML model corresponds to one edge on the analysis model. Composite networks must be connected through edges between ports, which are themselves nodes.

3.4.2.2 Selecting rate and time values

PN rates are selected as described in Subsection 2.4.2.2, however QN models the flow of commodities in a different way. This subsection explains the correspondence between PN terms and QA terms.

PNs and QA model networks in a different fashion and they use a different terminology, and two key distinctions are how commodities start and stop circulating in the networks. Only atomic processing nodes in PN correspond to queuing nodes in QA, since other kinds of nodes (i.e. regular, non-processing CFN nodes) do not have the required behavior. The information needed for QA needs to be derived from PNs in a specific manner, which will be described in the subsequent paragraphs.

First, QA uses the notion of *external arrival rates* for commodities in nodes. These rates are considered external because commodities are supposed to be coming from outside the network. Another way to describe this rate is to consider the rate at which start circulating in the queuing network. In PN, commodities start circulating when they are produced by nodes. If the network is restricted to only atomic processing node, commodities arriving from non-processing nodes should also be considered as starting circulating in the network.

As a result, the external arrival rate for queuing nodes need to take into account the production rates for commodities and the flow rates from non-processing nodes. The rate at which items arrive in queuing nodes is derived from the following rates in PN:

- the flow rate of commodity coming from non-processing node,
- the production rate of commodities.

QA requires all rates to be rates of exponential distributions. If some rates are not explicitly exponential distribution rates, the translation can be configured to treat non-distributed rates as being exponential distribution rates to allow QA.

Second, QA uses the notion of transition probabilities for commodities in notes. These probabilities are given for each node to all other nodes and for all kinds of commodities. The sum of these probabilities may not add up to 1, which indicates that commodities are departing from the network, or stop circulating in it.

In PN, commodities stop circulating when they are consumed by nodes. If the network is restricted to only atomic processing node, commodities departing to non-processing nodes should also be considered as stopping circulating in the network.

As a result, transition probabilities, which only apply to flows going from processing nodes to processing nodes, still need to take into account the consumption rates for commodities and the flow rates going to non-processing nodes. The routing probability in QA is derived from the following rates:

- the flow rate of commodities going to another processing node,
- the flow rate of commodities going to a non-processing node,
- the flow rate of commodities being consumed.

In QN, probabilities are only given to edges between queuing nodes (first case). However, when determining the edge probabilities, these three cases are taken into account. The probability for a given edge and a given commodity is computed by dividing the flow rate of commodities along that edge over the sum of flow rates of 1) commodities consumed by the source node and commodities going out the node, to either processing or non-processing nodes. As for *external arrival rates*, QA requires all rates to be rates of exponential distributions. If some rates are not explicitly exponential distribution rates, the translation can be configured to treat non-distributed rates as being exponential distribution rates to allow QA.

Finally, QA also requires service times to follow the exponential distributions. If some service times do not, the translation can be configured to treat non-distributed times as being exponentially distributed to allow QA.

QA can provide statistics (convergence over time) for each processing node, such as the average queue length or the response time. QA can provide a result only if a given QN can, on average, process commodities faster than they arrive. The randomness of the commodity arrival of the service time leads to some limited stacking up in the queue, and to some limited delay between the time when a commodity enters and leaves a processing node. If a node cannot process commodities faster than they arrive, there will be infinite stacking up and some commodities would never leave the node.

Over time, the input and output rates for a node should converge to a same number, which is consistent with the balance equation mentioned in Subsection 2.1.

This subsection describes the mathematical formulas that provide the analysis results, as well as the computational forms that are run by the numerical analysis software.

3.4.3 Octave

Octave is open-source software for scientific computations [6]. It is an open-source equivalent of MATLAB, and its language is very similar to MATLAB.

Octave comes with a queuing analysis library, with multiple functions to perform analysis on Markov chains (occupancy probabilities, mean time to absorption, expected and time-averaged sojourn times, mean first passage times) and queuing networks (convolution, mean value analysis).

For processing networks, this report covers mean-value analysis. The library provides a function that will be used to perform this analysis: `qnom`, which stands for Queuing Network, Open and Multi-class.

3.4.3.1 qnom Function

The translation in this report calls the Octave function `qnom` with four input parameters:

- a 2-dimensional matrix giving the external arrival rates for each commodity at each queuing node;
- a 2-dimensional matrix giving the mean service time at each queuing node for each commodity kind;
- a 4-dimensional matrix giving the transition probability between pairs of queuing nodes and commodity kinds;
- a 1-dimensional vector giving the number of servers at each queuing node.

The `qnom` function has 4 output parameters:

- a 2-dimensional matrix giving the utilization for each commodity kind at each queuing node;
- a 2-dimensional matrix giving the response time for each commodity kind at each queuing node;
- a 2-dimensional matrix giving the average number of requests for each commodity kind at each queuing node;
- a 2-dimensional matrix giving the throughput for each commodity kind at each queuing node.

Each SysML property typed by *AtomicProcessingNode* or a specialization of it corresponds to a queuing node.

The external arrival rates matrix is generated by computing for each edge the total rate of commodity incoming from non-queuing nodes and being produced by that node. That number includes commodities coming from non-queuing nodes and commodities being produced because these are considered to be “arriving at” or “appearing in” the network (they start circulating in the network). Commodities arriving from queuing nodes are not considered external, so they are excluded. Edge flow rates and production rates are selected from the model as described in 2.4.2.2.

The mean service time matrix is generated using service time values, which are selected from the model as described in 2.4.2.2 with the following modification:

- service time properties are not stereotyped by Rate;
- when an EventDistribution stereotype is used, the *mean* of the distribution is selected instead of the rate.

The transition probability matrix for nodes and commodity is generated by first computing the total rate of commodities going out of the queuing node (to both queuing nodes and non-queuing nodes) and being consumed, and then, for each outgoing edge, by dividing the flow rate of that

outgoing commodity by that total. The total includes commodities going to non-queuing nodes and commodities being consumed because these are considered to be “leaving” or “disappearing from” the network (they stop circulating in the network). Edge flow rates and consumption rates are selected from the model as described in 2.4.2.2.

Finally, the number of servers is given by the value of *concurrentServingCapacity* for each queing node.

3.4.4 MATLAB

This subsection gives the MATLAB code for performing queuing analysis.

3.4.4.1 qnom Function

The MATLAB equivalent to Octave’s `qnom` (multi-class queuing analysis) and `qnos` (single-class queuing analysis) functions is shown in the following code:

```
function [U, R, Q, X, L] = qnom(lambda, S, P, m)

  if ndims(lambda) ~= 2
    error("lambda must have 2 dimensions");
  end
  if any( lambda(:) < 0 )
    error( "all lambda values must be >= 0" );
  end
  [C,K] = size(lambda);

  if ndims(S) ~= 2
    error("S must have 2 dimensions");
  end
  if size(S) ~= [C,K]
    error( "S dimensions must be [%d,%d]", C, K );
  end

  if ndims(P) == 4
    if size(P) ~= [C,K,C,K]
      error( "P dimensions must be %dx%dx%dx%d",C,K,C,K );
    end
  elseif ndims(P) == 3
    if size(P) ~= [C,K,C]
      error( "P dimensions must be %dx%dx%d",C,K,C );
    end
  elseif ndims(P) == 2
    if size(P) ~= [C,K]
      error( "P dimensions must be %dx%d",C,K );
    end
  else
    error("P must have 2 or 4 dimensions");
  end
```

```
  if any( m(:) ~= 1 )
    error( "all m values must be = 1" );
  end

  V = qnomvisits(P, lambda);

  lambda = sum(lambda, 2);

  U = zeros(C,K);
  R = zeros(C,K);
  Q = zeros(C,K);

  X = sum(lambda)*V;
  rho = X .* S * diag( 1 ./ m );
  [Umax kmax] = max( sum(rho,1) );
  if Umax >= 1
    error( "Processing capacity exceeded at center %d", kmax );
  end

  for k=1:K
    for c=1:C
      if ( m(k) > 1 )
 [U(c,k)] = qsmmm( X(c,k), 1/S(c,k), m(k) );
      else
 [U(c,k)] = qsmm1( X(c,k), 1/S(c,k) );
      end
    end
  end

  k_fcfs = find(m>=1);
  for c=1:C
    Q(c,k_fcfs) = U(c,k_fcfs) ./ ( 1 - sum(U(:,k_fcfs),1) );
    R(c,k_fcfs) = Q(c,k_fcfs) ./ X(c,k_fcfs);
  end
  L = Q .* sum(U);
end

function V = qnomvisits(P, lambda)
  [C,K] = size(lambda);
  A = eye(K*C) - reshape(P,[K*C K*C]);
  b = reshape(lambda / sum(lambda(:)), [1,K*C]);
  V = reshape(b/A, [C, K]);
  V = max(0,V);
end

function V = qnosvisits(P, lambda)
  K = size(P,1);
  lambda = lambda(:)';
  A = eye(K)-P;
  b = lambda / sum(lambda);
  V = b*pinv(A);
  V = max(0,V);
end
```

```
function [U, R, Q, X, L] = qnos(lambda, S, P, m)

  if ndims(lambda) ~= 2
    error("lambda must have 2 dimension");
  end
  if any( lambda(:) < 0 )
    error( "all lambda values must be >= 0" );
  end
  [C, K] = size(lambda);

  if C ~= 1
    error("There must be only one kind of commodity");
  end

  if ndims(S) ~= 2
    error("S must have 2 dimensions");
  end
  if size(S) ~= [C,K]
    error( "S dimensions must be [%dx%d]", C, K );
  end

  if ndims(P) == 2
    if size(P) ~= [K,K]
      error( "P dimensions must be %dx%d",K,K );
    end
  else
    error("P must have 2 dimensions");
  end

  if any( m(:) < 1 )
    error( "all m values must be >= 1" );
  end

  V = qnosvisits(P, lambda);
  l = sum(lambda)*V;

  ii = find(l == 0);
  if numel(ii)
      U(ii) =0; R(ii) = 0; Q(ii) = 0; X(ii) = 0; L(ii) = 0;
      m(ii) = NaN;
  end

  ii = find( m == 1 );
  if numel(ii)
    [U(ii), R(ii), Q(ii), X(ii), L(ii)] = qsmm1( l(ii), 1./S(ii) );
  end

  ii = find( m>1 ); % multiple stations queueing centers
  if numel(ii)
    [U(ii), R(ii), Q(ii), X(ii), L(ii)] = qsmmm( l(ii), 1./S(ii), m(ii) );
  end
end
```

```
function [U, R, Q, X, L] = qsmm1(lambda, mu)
  lambda = lambda(:)';
  mu = mu(:)';
  rho = lambda ./ mu; % utilization
  U = rho;
  Q = rho ./ (1-rho);
  R = 1 ./ ( mu .* (1-rho) );
  X = lambda;
  L = (rho .* rho) ./ (1-rho);
end

function [U, R, Q, X, L] = qsmmm(lambda, mu, m)
  lambda = lambda(:)';
  mu = mu(:)';
  m = m(:)';
  X = lambda;
  rho = lambda ./ (m .* mu );
  U = rho;
  pm = erlangc(lambda ./ mu, m);
  Q = m .* rho + rho ./ (1-rho) .* pm;
  R = Q ./ X;
  L = rho ./ (1-rho) .* pm;
end

function C = erlangc(A, m)
  rho = A ./ m;
  B = erlangb(A, m);
  C = B ./ (1 - rho .* (1 - B));
end

function B = erlangb(A, m)
  B = arrayfun( @erlangb_compute, A, m);
end

function B = erlangb_compute(A, m)
  Binv = 1.0;
  for k=1:m
    Binv = 1.0 + k/A*Binv;
  end
  B = 1.0 / Binv;
end
```

Since the the queuing analysis function in MATLAB takes the same inputs and the outputs as in Octave, the relationship between these and the SysML model is the same as described in 3.4.3.

3.5 Translation and Queuing Analysis of Processing Network example

The two variations of processing networks presented in Subsection 3.3 can be translated for queuing analysis [4]. The QA results should be the same on all analysis tools for any given example.

3.5.1 Octave representation of single-commodity Processing Network example

The Octave representation relies on matrices, so each atomic process node and commodity has a given index in the matrix.

The following code shows the indices assigned for the current example.

```
#1: apn1
#2: apn2
#1: CustomCommodity
```

There is only one commodity (*CustomCommodity*), and since only atomic process nodes are represented as queuing nodes in a queuing analysis, there are only two nodes translated for analysis: *apn1* and *apn2*.

The following code shows the first line in the Octave script, which loads the queuing library.

```
pkg load queueing;
```

Then, the matrices that serve as input of the qnos function are created.

The matrix `lambda` gives the external arrival rate for each commodity in each node. In the example, it is a 1x2 matrix with values given for the two nodes. The external arrival rate combines production flow rates and flow rates from non-atomic process nodes. For the node *apn1*, there is a production rate of 20 and a flow rate of 20 coming in from *cfn1*, so the external arrival rate is 40. For the node *apn2*, there is no production and no flow rate coming in from non-atomic process nodes, so the external arrival rate is 0.

The following code shows the code generated for the external arrival rates.

```
lambda = zeros([1,2])
lambda(1)=40.0;
```

The matrix `P` gives the transition probability between each node. In the given example, it is a 2x2 matrix with values corresponding to the amount of commodities going to the given nodes over the total amount of commodities going ”out” of the node. That total amount includes commodities going to both atomic process nodes and non-atomic process nodes, as well as commodities consumed.

From the node *apn1* to the node *apn2*, there is a flow of 50. The total amount of commodities going out of *apn1* is also 50, because there is no other flow going out of it and there is no consumption in the node. So, the probability from *apn1* to *apn2* is 50/50=1.

From the node *apn2* to the node *apn1*, there is a flow of 10. The total amount of commodities going out of *apn2* is 50, because there is a flow of 20 going out to *cfn2* and there is 20 of consumption in the node. So, the probability from *apn2* to *apn1* is 10/50=0.20.

The following code shows the code generated for the transition probabilities.

```
P = zeros([2,2])
P(1,2)=1.0;
P(2,1)=0.2;
```

The matrix *s* gives the service time for each commodity in each queuing node. In the given example, it is a 1x2 matrix with values computed corresponding to the service time provided in the SysML model.

The following code shows the code generated for the service times.

```
S = zeros([1,2])
S(1)=0.018;
S(2)=0.036;
```

The following code shows the vector m, which gives the number of servers for each node. Since *apn2* has 2 servers and *apn1* has 1, the following code is generated:

```
m = ones([1,2])
m(1,2)=2;
```

After all the generation of all input parameters, The following code shows the function calls for the queuing analysis.

```
  V = qnosvisits( P, lambda )
  [U, R, Q, X] = qnos (sum(lambda), S, V, m)
  L = Q - m .* U
```

The following code shows the result of the analysis.

```
U =

   0.9000   0.9000

R =

   0.1800   0.1895

Q =

   9.0000   9.4737

X =

   50   50

L =

   8.1000   7.6737
```

As these results show, the utilization is 90% and the throughput is 50. Compared to *apn1*, *apn2* has a slightly higher response time, slightly more commodities in it at any given time, but has a shorter queue length.

3.5.2 MATLAB representation of single-commodity Processing Network example

The following code shows the MATLAB representation equivalent to the Octave representation, see above for description:

```
%1: apn1
%2: apn2
%1: CustomCommodity

P = zeros([2,2]);
P(1,2)=1.0;
P(2,1)=0.2;

lambda = zeros([1,2]);
lambda(1)=40.0;

S = zeros([1,2]);
S(1)=0.018;
S(2)=0.036;

m = ones([1,2]);
m(1,2)=2;

[U, R, Q, X, L] = qnos(lambda, S, P, m)
```

The following code shows the result of the analysis. It is the same as obtained in Subsection 3.5.1.

```
U =

    0.9000    0.9000


R =

    0.1800    0.1895


Q =

    9.0000    9.4737


X =

   50.0000   50.0000


L =

    8.1000    7.6737
```

3.5.3 Octave representation of multi-commodity Processing Network example

For the multi-commodity example, the following code shows the updated matrix indices that reflect the new model.

```
#1: apn1
#2: apn2
#1: CustomCommodityA
#2: CustomCommodityB
```

Compared to the single-commodity example, there are now two commodities (*CustomCommodityA* and *CustomCommodityB*).

The matrix `lambda` that gives the external arrival rate for each commodity in each node is now a 2x2 matrix because there are two commodities.

For the node *apn1*, there is a production rate of 12 *CustomCommodityA* and 8 *CustomCommodityB*, as well as a flow rate of 12 *CustomCommodityA* and 8 *CustomCommodityB* coming in from *cfn1*, so the external arrival rates are 24 and 16. For the node *apn2*, there is no production and no flow rate coming in from non-atomic process nodes, so the external arrival rates are 0.

The following code shows the updated external arrival rates.

```
lambda = zeros([2,2])
lambda(1,1)=24.0;
lambda(2,1)=16.0;
```

The matrix `P` gives the transition probability between each node. In this multi-commodity example, it is now a 4-dimensional, 2x2x2x2 matrix. The values and the totals are computed per node and per commodity like for the single commodity example.

From the node *apn1* to the node *apn2*, there is a flow of 30 *CustomCommodityA* and 20 *CustomCommodityB*. The total amount of commodities going out of *apn1* are also 30 and 20, respectively, because there is no other flow going out of it and there is no consumption in the node. So, the probabilities from *apn1* to *apn2* are 30/30=1 for *CustomCommodityA*, and 20/20=1 for *CustomCommodityB*.

From the node *apn2* to the node *apn1*, there is a flow of 6 *CustomCommodityA* and 4 *CustomCommodityB*. The total amount of commodities going out of *apn2* are 30 and 20 respectively, because there are also flows of 12 *CustomCommodityA* and 8 *CustomCommodityB* going out to *cfn2*, and there are 12 and 8 of consumption in the node. So, the probability from *apn2* to *apn1* is 6/30=0.20 of *CustomCommodityA* and 4/20=0.20 *CustomCommodityB*.

The following code shows the updated external arrival rates.

```
P = zeros([2,2,2,2])
P(1,1,1,2)=1.0;
P(2,1,2,2)=1.0;
P(1,2,1,1)=0.2;
P(2,2,2,1)=0.2;
```

The matrix *s* gives the service time for each commodity in each queuing node. In the given example, it is a 2x2 matrix with values computed corresponding to the service time provided in the SysML model.

The following code shows the updated service times.

```
S = zeros([2,2])
S(1,1)=0.018;
S(2,1)=0.018;
S(1,2)=0.02;
S(2,2)=0.015;
```

The following code shows the updated number of servers. Both *apn1* and *apn2* have 1 server, so the following code is generated:

```
m = ones([1,2])
```

The following code shows the updated function calls for multi-commodity networks.

```
  V = qnomvisits( P, lambda )
  [U, R, Q, X] = qnom (sum(lambda,2), S, V, m)
  L =  Q .* sum(U)
```

Finally, the following code shows the result of queuing analysis.

```
U =

   0.5400   0.6000
   0.3600   0.3000

R =

   0.1800   0.2000
   0.1800   0.1500

Q =

   5.4000   6.0000
   3.6000   3.0000

X =

   30   30
   20   20

L =

   4.8600   5.4000
   3.2400   2.7000
```

As these results show, the combined (i.e. adding up values for the two kinds of commodities) utilization is 90%, and the combined throughput is 50. There are slight difference between the two

nodes, explained by the fact that *apn2* has different service times for each commodities. *apn2* has a slightly lower combined response time (0.035 vs 0.036), but both nodes have the same number of commodities in the node (9) and the same average queue length (8.1).

3.5.4 MATLAB representation of multi-commodity Processing Network example

The following code shows the MATLAB representation equivalent to the Octave representation above.

```
%1: apn1
%2: apn2
%1: CustomCommodityA
%2: CustomCommodityB

P = zeros([2,2,2,2]);
P(1,1,1,2)=1.0;
P(2,1,2,2)=1.0;
P(1,2,1,1)=0.2;
P(2,2,2,1)=0.2;

lambda = zeros([2,2]);
lambda(1,1)=24.0;
lambda(2,1)=16.0;

S = zeros([2,2]);
S(1,1)=0.018;
S(2,1)=0.018;
S(1,2)=0.02;
S(2,2)=0.015;

m = ones([1,2]);

[U, R, Q, X, L] = qnom(lambda, S, P, m)
```

Finally, the following code shows the result of queuing analysis in MATLAB. It is the same as obtained in Subsection 3.5.3.

```
U =

    0.5400    0.6000
    0.3600    0.3000


R =

    0.1800    0.2000
    0.1800    0.1500


Q =
```

```
    5.4000    6.0000
    3.6000    3.0000


X =

   30.0000   30.0000
   20.0000   20.0000


L =

    4.8600    5.4000
    3.2400    2.7000
```

Process networks can also be translated into Discrete Event Simulation models, which when simulated after a long enough time, should provide statistics that converge towards the queuing analysis values.

3.6 Translating between Processing Networks and Discrete Event Simulation tools

This subsection gives translations between SysML models extended for PN, see Subsection 3.2, and SimEvents and AnyLogic models.

3.6.1 Additional SysML model constraints for translation

In addition to the constraints specified in Section 3.4.1, the services types must be concrete because the DES models instantiate them.

3.6.2 Processing Networks and Discrete Event Simulation models

As described in Subsection 3.1, PNs can include atomic processing nodes, which define behaviors that queue and service all commodities flowing within them. These behaviors can be simulated by DES tools, which rely on various kinds of inter-related elements (also called “blocks”, including when they act as ports) that exchange items through port blocks. Their behavior is governed by events triggered by other events or by a timer, where simulation steps cannot be “broken down” into substeps. The following kinds of DES blocks are used to model the internal behavior of atomic processing nodes:

- Input port blocks, through which commodities arrive in the node from other nodes.
- Output port blocks, through which commodities depart from the node to other nodes.
- Source blocks that create commodities at specified time intervals.

- Queue blocks, in which a certain quantity of commodities can wait for the next block to become available.
- Service blocks, which take a certain amount of time to process (delay) commodities and may serve multiple commodities concurrently.
- Sink blocks that destroy commodities.
- Decision blocks, which direct an commodities to one of many paths according to specified probabilities or conditions.
- Merge blocks, which bring together different paths.

Figure 13 illustrates atomic node behavior in DES. Rectangles indicate DES blocks. Diamonds are DES decision and merge blocks, distinguished by whether multiple arrows lead out of or into them, respectively. It proceeds as follows:

1. A node's inputs from other nodes appear in its input port, and they are combined using a DES merge block with commodities created in the node by DES source blocks (see Subsection 3.6.2 for creation rate).
2. All these commodities go to a DES queue block.
3. All these commodities go to a DES service block, and stay there for an average time that depends on the commodity type.
4. Commodities that will no longer circulate in the network are routed by a DES decision block to a DES sink block, based on the probability to either be consumed or go to a non-processing node (See Subsection 3.6.2 for destruction probability)
5. Other commodities are combined using a DES merge block and go to the node's output port.

Figure 14 illustrates edge behavior in DES. It proceeds as follows:

1. Node outputs are split by type by a Decision block,
2. Commodities are split by destination by another Decision block, based on the probability of going to a particular destination (see Subsection 3.6.2 for routing probability),
3. Commodities are merged by a Merge node before going to their destination's input.

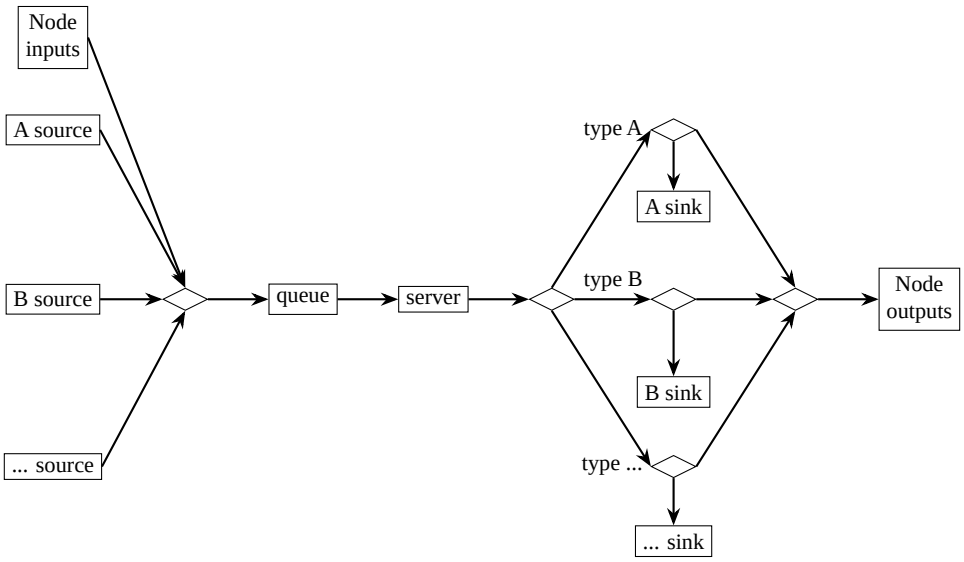

Figure 13: Representing atomic processing nodes in discrete event simulation

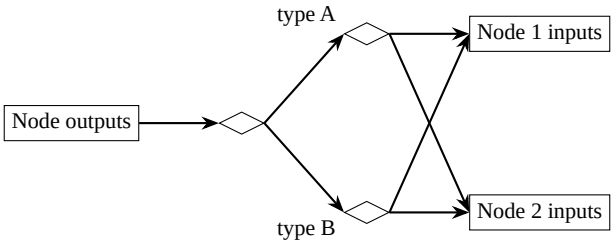

Figure 14: Representing edges between processing nodes in discrete event simulation

The translation of PNs models into DES models requires three kinds of computations: the creation rate for each commodity in each queuing node, the destruction rate for each commodity in each queuing node, and the probability of a commodity to be going from one queuing node to a particular queuing node. Rates are selected from the model as described in 2.4.2.2.

Given:

- C: set of commodities,
- K_{all}: set of all nodes (queuing and non-queuing),
- $K_{queuing}$: set of all queuing nodes ($K_{queuing} \subset K_{all}$),
- $K_{nonqueuing}$: set of all non-queuing nodes ($K_{nonqueuing} = K_{all} - K_{queuing}$),
- $Prod_{c,k}$: the production rate of commodity c in the node k ($c \in C, k \in K_{queuing}$),
- $Cons_{c,k}$: the consumption rate of commodity c in the node k ($c \in C, k \in K_{queuing}$),
- $Flow_{c,k,l}$: flow rate of commodity c from k to l ($c \in C, k, l \in K_{all}$).

The creation rate of commodity $c \in C$ in node $k \in K_{queuing}$ is:

$Prod_{c,k} + \sum_{j \in K_{nonqueuing}} Flow_{c,j,k}$

The destruction probability of commodity $c \in C$ in node $k \in K_{queuing}$ is:

$\frac{Cons_{c,k} + \sum_{l \in K_{nonqueuing}} Flow_{c,k,l}}{Cons_{c,k} + \sum_{l \in K_{all}} Flow_{c,k,l}}$

The probability of commodity $c \in C$ to go from node $k \in K_{queuing}$ to node $l \in K_{queuing}$ is:

$\frac{Flow_{c,k,l}}{Cons_{c,k} + \sum_{m \in K_{all}} Flow_{c,k,m}}$

3.6.3 SimEvents

Table 1 shows the term equivalence between PN/DES and SimEvents.

PN / DES terms	SimEvents equivalent
Type	System
Property	Block
Input port	Inport
Output port	Outport
Commodity	Entity
Source	Entity generator
Sink	Entity terminator
Merge	In switch
Decision	Out switch
Service	Entity server

Table 1: Term equivalence between Processing Networks and SimEvents

3.6.3.1 Commodity

SimEvents requires entities to have the same type along its connections, which is problematic if distinct commodity types are translated as distinct entity types. A workaround is to create entities based on commodities' signatures (using attributes names, multiplicity, and types), and have the corresponding type as attribute value if needed for comparing types. SimEvents entities also have a `decision` attribute used for `OutputSwitch`. The `OutputSwitch` block cannot select a path based on an expression, but can select a path based on an attribute value. In the translation to SimEvents, decision nodes relying on expressions are translated as a 0-time "calculation" `EntityServer` block that compute the expression and sets an attribute value corresponding to the index of an `OutputSwitch`'s output, and the subsequent `OutputSwitch` block selects the `Outport` block based on that value. Since an entity cannot be at multiple location at once, there cannot be concurrent setting and getting of the `decision` attribute.

3.6.3.2 Processing networks

A processing network is translated as a Simulink `System`, referred to as the *root* `System`.

3.6.3.3 AtomicProcessingNodes, without preemptive resume

For each node of processing network, a `System` is created along with a `Subsystem` block owned by the root `System`.

The content of each node's system is as follows:

- An `Inport` block is created. See DES representation in Subsection 4.4.3.17.
- An `EntityGenerator` block is created for each produced commodity. The entity type is given as per the previous paragraph, and the intergeneration time `dt` is given by the following formula: $dt = -(1/\lambda) * log(1 - rand)$, where λ is the production rate and $rand$ is a randomly generated double number between 0 and 1. See DES representation in Subsection 4.4.3.4.
- An `InSwitch` block is created, along with connections from the `Inport` block and from all the `EntityGenerator` blocks. See Subsection 4.4.3.18 for more information.
- A `Queue` block is created, along with a connection from the `InSwitch` block. The `Capacity` of the queue is set to the node's *storageCapacity* (`Inf` is unbounded). See DES representation in Subsection 4.4.3.5.
- An `EntityServer` block is created, along with a connection from the `Queue` block. The `Capacity` of the server is set to the node's *concurrentServingCapacity* and for each commodity flowing in the node, the service time is given by the following formula: $dt = -\mu * log(1 - rand);$, where μ is the mean service time for that commodity and $rand$ is a randomly generated double number between 0 and 1. See DES representation in Subsection 4.4.3.10.
- A calculation `EntityServer` block and an `OutSwitch` block are created, along with connections between them and from the `EntityServer` block to the calculation block. The `OutSwitch` block has as many output as there are commodity kinds flowing through the node, and each item is routed to an `Outport` block corresponding to its type (the proper port index is set as value of the decision attribute in the calculation block). See DES representation in Subsection 4.4.3.19.
- A calculation `EntityServer`, an `OutSwitch` block, and an `EntityTerminator` block are created for each commodity kind, and the calculation block block is connected to the `Outport` of the previous `OutSwitch` that corresponds to that commodity. Each new `OutSwitch` block has two `Outports`: one to the `EntityTerminator`, and one to the next `InSwitch` block. The switching (computed in the calculation block) is based on the probability that an item will be consumed or leave the network (in which case it goes to the `EntityTerminator`), as opposed to the probability that an item will go to another processing node (in which case it goes to the next `InSwitch` block). See DES representation in Subsection 4.4.3.19.
- An `InSwitch` block is created, along with connections from all the previous `OutSwitch` blocks. See DES representation in Subsection 4.4.3.18.
- An `Outport` block is created, along with a connection from the previous `InSwitch`. See DES representation in Subsection 4.4.3.17.

3.6.3.4 AtomicProcessingNodes, with preemptive resume

If the node has a *LIFO* discipline with *preemptiveResume* equal to true, some modifications are required:

- A timestamp `posixtime(datetime())` is set to an entity's `preemption` attribute when it arrived in the queue and this attribute is not 0.
- The `Queue` is changed to a priority queue with descending order, based on the `preemption` attribute.
- The `EntityServer` block is changed to permit preemption based on the `preemption` attribute and store the residual time in the `residualtime` attribute.
- The output of the `EntityServer` corresponding to the preempted element is connected to the `InSwitch` preceding the `Queue`.
- Upon completion of the service, the entity's `preemption` attribute is reset to 0 so that it is ready to go to another node.

3.6.3.5 Links between AtomicProcessingNodes

For every node that is the destination of a link between processing nodes, an `InSwitch` block is created in the root `System`.

The following actions are taken for every node that is the source of a link between processing nodes:

- A calculation `EntityServer` and an `OutSwitch` block are created, along with a connection from the node's `Outport`. The switching (computed in the calculation block) is based on the commodity type. See Subsection 4.4.3.19 for more information.
- For each commodity type, a calculation `EntityServer` and an `OutSwitch` block are created along with a connection to the corresponding previous `OutSwitch` block's `Outport`. There is an outgoing connection to the `InSwitch` block of each destination, and the switching is based on the probability of a commodity going to that destination (based on flow rates). See Subsection 4.4.3.19 for more information.

3.6.4 AnyLogic

Table 2 shows the term equivalence between PN/DES and AnyLogic.

PN / DES terms	AnyLogic equivalent
Type	Active object class
Property	embedded object
Input/Output port	Port
Commodity	Agent
Source	Source
Sink	Sink
Merge	N/A
Decision	SelectOutput
Service	Service

Table 2: Term equivalence between Processing Networks and AnyLogic

3.6.4.1 Commodity

In AnyLogic, Commodity kinds translate into Java classes that specialize *Agent*. These Agent classes can be abstract as in SysML. Also. SysML specializations can be translated into Java class extensions (unless there is multiple inheritance).

3.6.4.2 Processing networks

A processing network is translated as an active object `Class`, referred to as the *root* `Class`.

3.6.4.3 AtomicProcessingNodes, without preemptive resume

For each node of processing network, a `Class` is created along with an embedded object owned by the root `Class`.

The content of the node's `Class` is as follows:

- An (input) Port object is created. See Subsection 4.4.4.17 for more information.
- A `Source` object is created for each produced commodity. The agent type is given as per the previous paragraph, the *`arrivalType`* parameter is set to *`INTERARRIVAL_TIME`*, the *`interarrivalTime`* parameter is set to $exponential(\lambda)$, where λ is the production rate. See Subsection 4.4.4.4 for more information.
- A `Queue` object is created, along with connectors from the `Port` and from all the `Source` objects. If the queue capacity is unbounded, the *`maximumCapacity`* parameter is set to *`true`*. Otherwise, the *`capacity`* parameter is set to the node's *storageCapacity*. See Subsection 4.4.4.5 for more information.
- A `Delay` object is created, along with a connector from the `Queue` object. If the number of servers is unbounded, the *`maximumCapacity`* parameter is set to *`true`*. Otherwise, the *`capacity`* parameter is set to the node's *concurrentServingCapacity*. The *`delayTime`* parameter is an expression that, for each commodity type, returns $exponential(1/\mu)$, where μ is the mean service time for that commodity type. See Subsection 4.4.4.10 for more information.

- A `SelectOutput` object is created, along with a connector to it from the `Delay` object. This object routes flowing agents to separate outputs depending on their type. See Subsection 4.4.4.19 for more information.
- A `SelectOutput` object and a `Sink` object are created for each commodity kind, and the `SelectOutput` object is connected to the output `Port` of the previous `SelectOutput` that corresponds to that commodity. Each new `SelectOutput` has two outputs: one to the `Sink` object, and one to the next (output) `Port`. The switching is based on the probability that a commodity will be consumed or leave the network (in which case it goes to the `Sink` object), as opposed to the probability that an item will go to another processing node (in which case it goes to the next port). See Subsection 4.4.4.19 for more information.
- An (output) `Port` is created, along with a connector from all the previous `SelectOutput` outputs that do not go to the `Sink` object. See Subsection 4.4.4.17 for more information.

3.6.4.4 AtomicProcessingNodes, with preemptive resume

If the node has a *LIFO* discipline with *preemptiveResume* equal to true, some modifications are required:

- The `Queue` and the `Delay` object are replaced by a single `Service` object.
- A `ResourcePool` object is created with a number of resource corresponding to the node's *concurrentServingCapacity*
- A timestamp *`-System.currentTimeMillis()`* is set to an agent's *`preemption`* attribute when it arrived in the object.
- The parameters of the original `Queue` are the same as the parameters of the new `Service`, but the *`priority`* parameter is set to *`agent.preemption`*.
- The `Service` object is changed to permit preemption.
- Upon completion of the service, the agent's *`preemption`* attribute is set to 0.

3.6.4.5 Links between AtomicProcessingNodes

The following actions are taken for every node that is the source of a link between processing nodes:

- A `SelectOutput` object is created, along with a connector from the node's (output) `Port`. The switching is based on the commodity type. See Subsection 4.4.4.19 for more information.
- For each commodity type, a `SelectOutput` object is created along with a connector to the corresponding previous `SelectOutput` object's output `Port`. There is an outgoing connector to the (input) `Port` of each destination, and the switching is based on the probability of a commodity going to that destination (based on flow rates). See Subsection 4.4.4.19 for more information.

3.7 Translation and Discrete Event Simulation of Processing Network example

The two variations of processing network example presented in Subsection 3.3 can be translated to DES and simulated [4].

The DES results should be the similar on all analysis tools for any given example, but they are not likely to be the exact same due to the stochastic nature of the simulations. Moreover, since the requirements from QA are met, these results should converge to the QA results obtained in 3.5.

3.7.1 SimEvents representation of the single-commodity Processing Network example

After generating the SimEvents model and running the simulation for 10,000 seconds, the results are as shown in Figures 15, 16, 17, and 18. The values at end of simulation are (compared to expected queuing analysis values):

- Queue length in *apn1* is about 8.2 (8.1000 expected),
- *apn1* utilization ratio is slightly over 0.90 (0.9 expected),
- Queue length in *apn2* is about 7.8 (7.6737 expected),
- *apn2* utilization ratio is about 0.90 (0.9 expected).

These results are compatible with queuing analysis of this example (see Subsections 3.5.2 and 3.5.1).

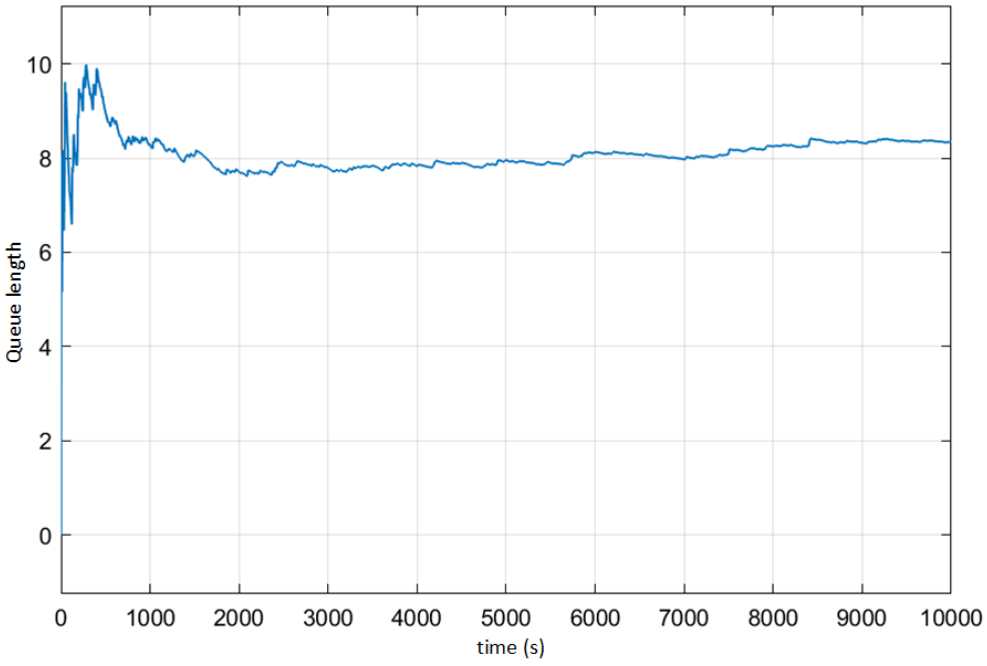

Figure 15: Discrete Event Simulation, single commodity, apn1 queue length, SimEvents

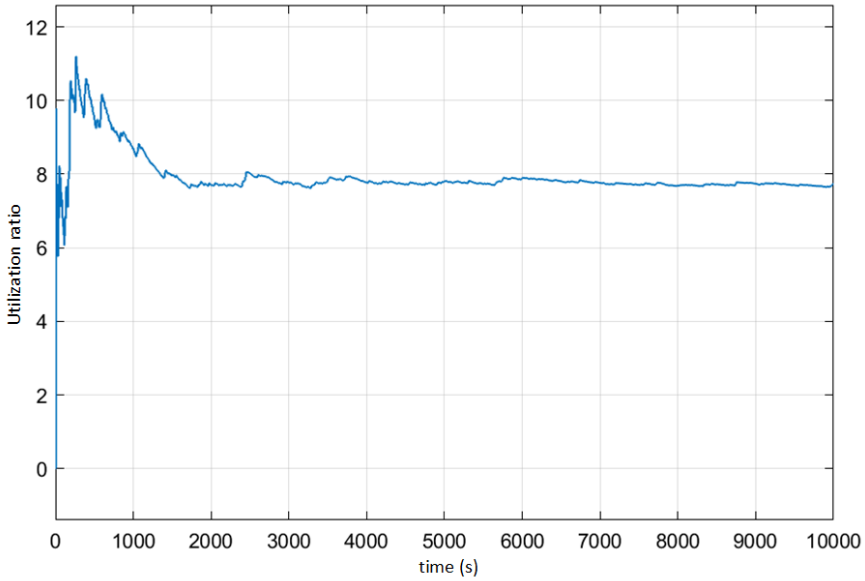

Figure 16: Discrete Event Simulation, single commodity, apn1 utilization ratio, SimEvents

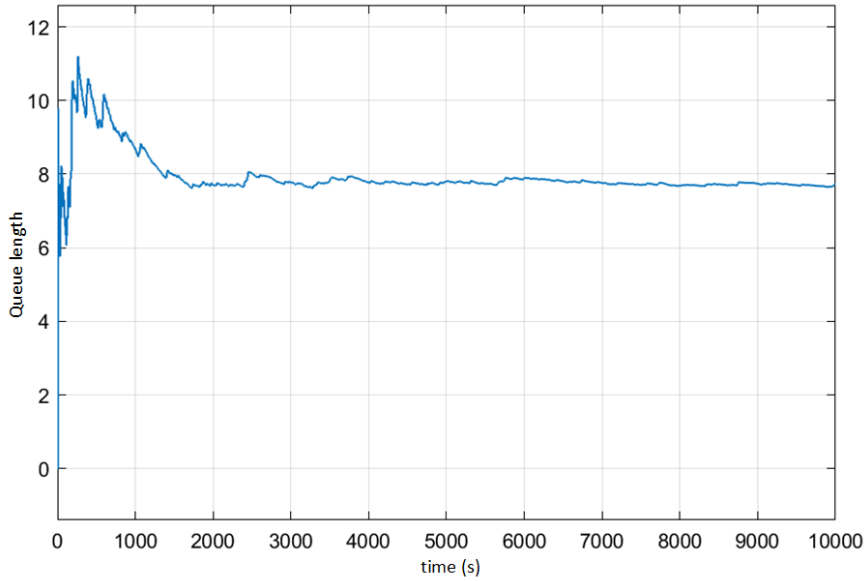

Figure 17: Discrete Event Simulation, single commodity, apn2 queue length, SimEvents

3.7.2 AnyLogic representation of the single-commodity Processing Network example

After generating the AnyLogic model and running the simulation, the following results are obtained for the queues. They are similar to those obtained in 3.7.1.

```
root.apn1.queue: Queue
  Capacity: maximum
  Timeout: disabled
  Preemption: disabled
  in:  62,496
  out:  62,496
  Length (av): 7.904

root.apn2.queue: Queue
  Capacity: maximum
  Timeout: disabled
  Preemption: disabled
  in:  62,495
  out:  62,492
  Length (av): 8.613
```

3.7.3 SimEvents representation of the multi-commodity Processing Network example

After generating the SimEvents model and running the simulation for 10,000 seconds, the results are as shown in Figures 19, 20, and 22. The values at end of simulation are (compared to expected queuing analysis values):

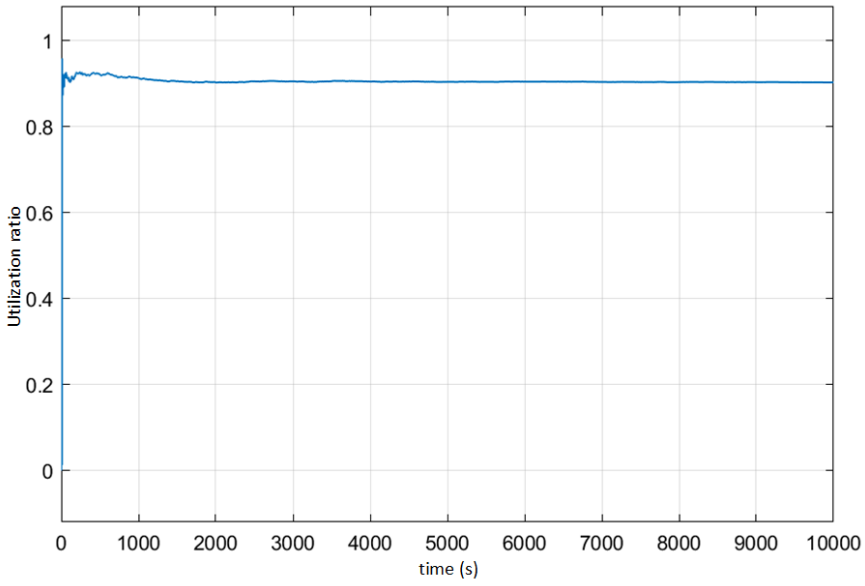

Figure 18: Discrete Event Simulation, single commodity, apn2 utilization ratio, SimEvents

- the queue length in *apn1* is about 8 (8.1 expected),
- the *apn1* utilization ratio is slightly over 0.90 (0.9 expected),
- the queue length in *apn2* is about 8 (8.1 expected),
- the *apn2* utilization ratio is about 0.90 (0.9 expected).

These results are compatible with queuing analysis of this example (see Subsections 3.5.4 and 3.5.3).

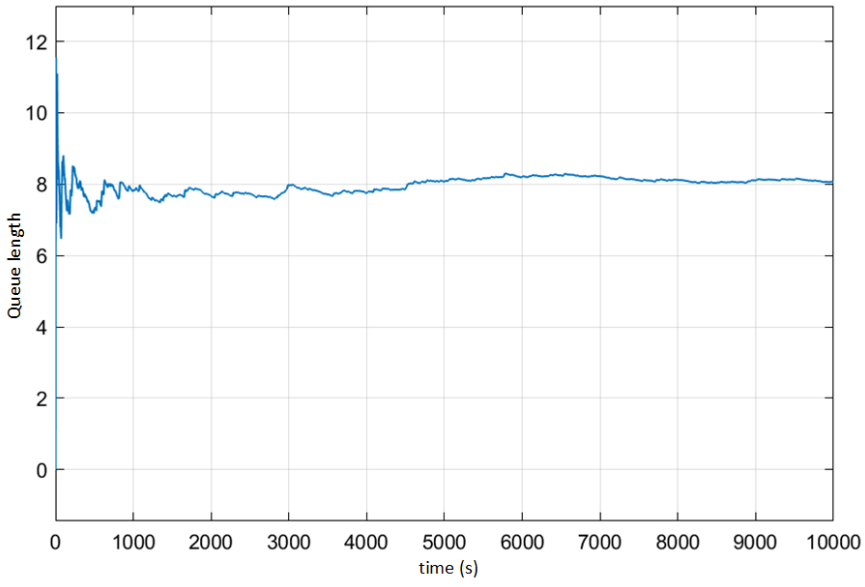

Figure 19: Discrete Event Simulation, multi commodity, apn1 queue length, SimEvents

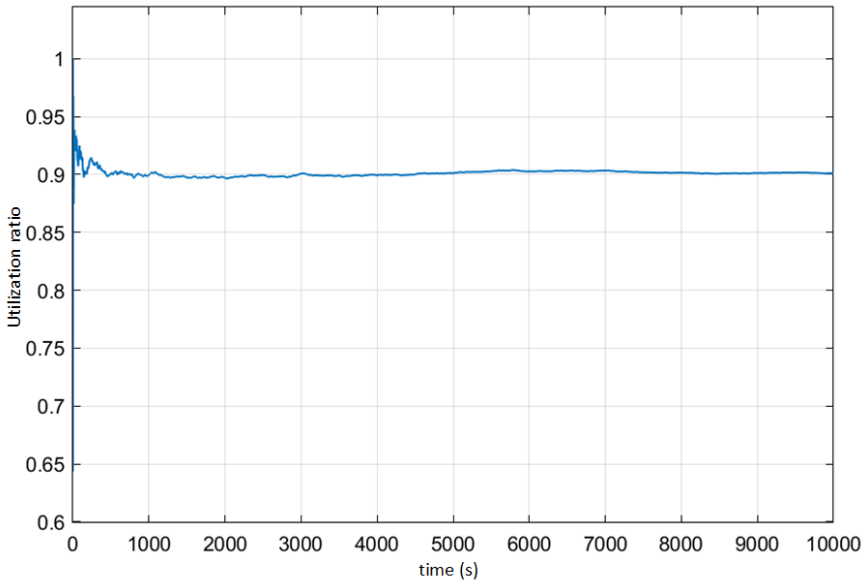

Figure 20: Discrete Event Simulation, multi commodity, apn1 utilization ratio, SimEvents

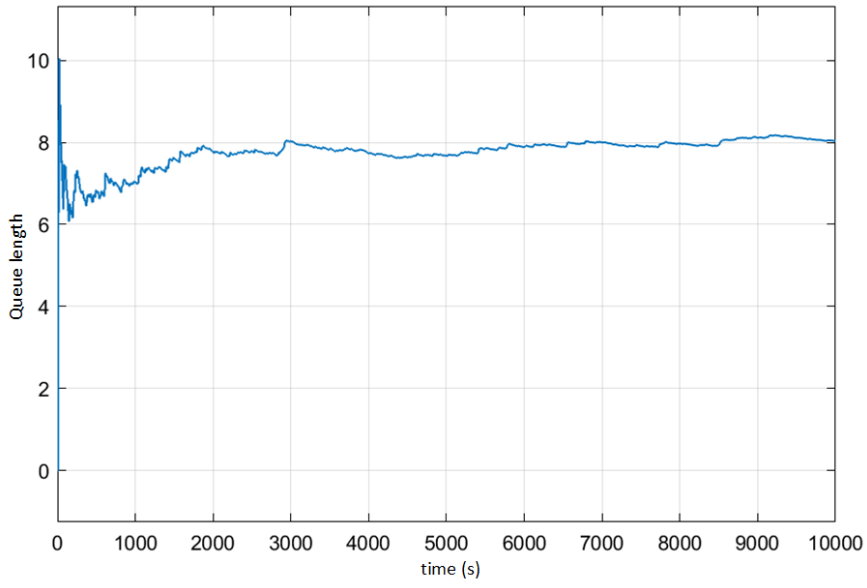

Figure 21: Discrete Event Simulation, multi commodity, apn2 queue length, SimEvents

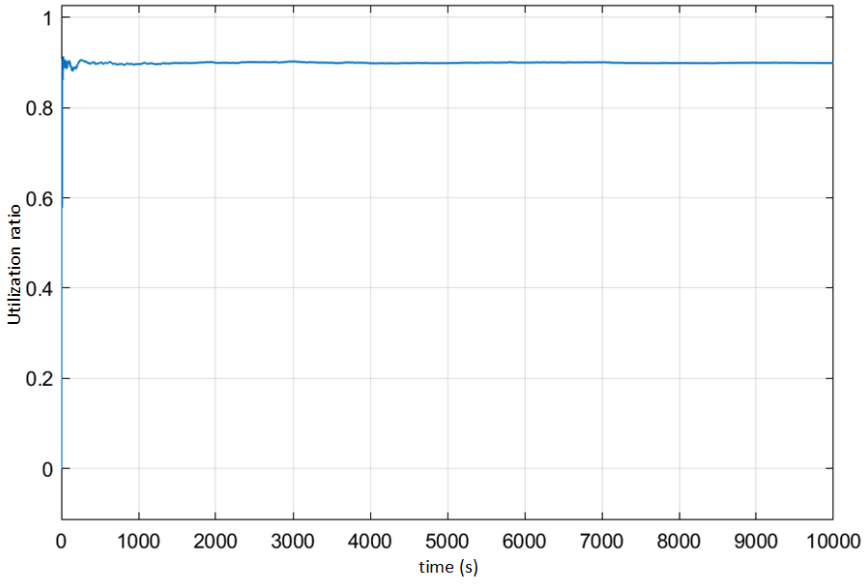

Figure 22: Discrete Event Simulation, multi commodity, apn2 utilization ratio, SimEvents

3.7.4 AnyLogic representation of the multi-commodity Processing Network example

After generating the AnyLogic model and running the simulation, the following results are obtained for the resource pools. They are similar to those obtained in 3.7.3.

```
root.apn1.resourcepool: ResourcePool
All units:  1
Active:     1
Idle units: 1
  Utilization: 0.904

root.apn2.resourcepool: ResourcePool
All units:  1
Active:     1
Idle units: 0
  Utilization: 0.892
```

4 Discrete Event Networks

This section defines Discrete Event Networks (DENs) and translations between extended SysML models and Discrete Event Simulation (DES) tools. DENs operate by steps, rather than continuously, reacting to discrete events, such as timers, with commodities moving between nodes at discrete times as well. This is similar enough to SysML activities that the extension is only a library of primitive activities (that do not use other activities) to reuse in activities representing DENs. These primitives often handle only one kind of commodity, unlike Commodity Flow Network (CFN) and Processing Network (PN) nodes, however the behavior of these primitives can be

customized by adding code to be executed, including to simulate modifications to commodities that add value to them. Subsection 4.1 reviews DEN and DES, while Subsections 4.2 and 4.4 define an extension of SysML for DENs and translation between extended SysML models and DES tools, respectively. See Appendix B for case and font conventions for model and analysis tool element names and Appendix C for an introduction to SysML.

4.1 Modeling and analysis

DENs in this specification are SysML activity models where the actions call other DEN activities, or primitive ones corresponding to those commonly supported by DES tools:

- *Produce*: Provide new commodities as output at specified inter-generation intervals.
- *Consume*: Accept commodities as input and dispose of them.
- *Queue*: Accept commodities as input, accumulate them, and provide them as output in a specified order when the subsequent action is ready to accept them.
- *Seize*: Provide input commodities as outputs when a resource from a resource pool becomes available, and remove that resource from the pool until release.
- *Release*: Provide input commodities as output, and return the resource previously seized to a resource pool.
- *Delay*: Provide input commodities as outputs after a specified duration.
- *Combine*: Provide new commodities as output, with the input commodities as attribute values.
- *Split*: Accept commodities as input and provide their attribute values as outputs.
- *Replicate*: Accept commodities as input and provide a copy of them as output.
- *Transform*: Accept commodities as input and provide new commodities as output.
- *Serve*: Accept commodities as input and provides them as output after a delay. Includes a resource pool to service commodities.
- *Process*: Act like a *Queue* followed by a *Serve*.

To align with DES tools, the activity parameters are all streaming and all activity edges are object flows. Actions typed by these activities run for the entire time of the simulation: they start executing when the simulation start, and they stop when the simulation stops. They can receive and emit tokens at any point during their execution, which is the meaning of streaming parameters.

DES in this specification is the execution of a DEN activities per SysML semantics, except:

- All actions begin and end executing at the beginning and end of the simulation, respectively.

- During simulation they typically receive and/or send multiple commodities over time.
- No object nodes may not hold more than one token at any point in time. *Queue*, *Delay*, *Serve*, and *Process* actions may hold more than one token any any point in time.
- In SysML activities, tokens on an output pin might not be accepted by its subsequent action and remain in the output pin, without affecting the action that provided the token. In DES, when an output is provided but not accepted as input to another element, the element providing the output will often be prevented from continuing its execution and/or accepting new inputs.

4.2 SysML extension

The SysML representation of DENs are SysML activities calling other activities as described in 4.1, some of which can be primitives defined in a library, see Subsection 4.2.1. This includes all information needed for analysis.

4.2.1 Library

Figure 23 shows the DEN library, composed of primitive activities corresponding to those commonly supported by DES tools, as described in Subsection 4.1.

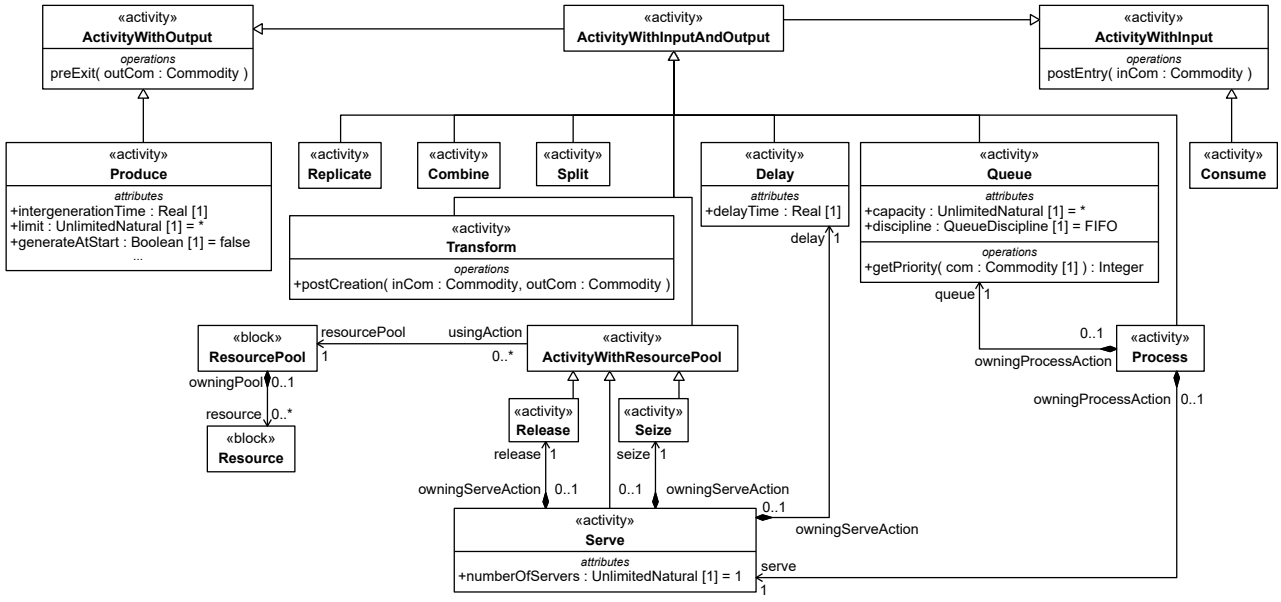

Figure 23: Discrete Event Network Library

ActivityWithInput is an abstract Activity that has one input parameter, and that can act upon token arrival through its *postEntry()* operation.

ActivityWithOutput is an abstract Activity that has one output parameter, and that can act upon token departure through its *preExit()* operation.

ActivityWithInputAndOutput is an abstract Activity that specialized both *ActivityWithInput* and *ActivityWithOutput*.

ActivityWithResourcePool is a specialization of *ActivityWithInputAndOutput* that also has a *resourcePool*.

The *Combine* activity is a specialization of *ActivityWithInputAndOutput* that creates a commodity of the same type as the output parameter, and assigns its attributes values to those of the input parameters. The number of inputs (input pins on the call behavior action) must match the number of attributes (inherited and owned) in the output commodity type.

The *Consume* activity is a specialization of *ActivityWithInput* that disposes of all input commodities.

The *Delay* activity is a specialization of *ActivityWithInputAndOutput* that takes (accepts) input commodities and outputs (offers) them after a specified delay. It has a *delayTime* property that indicate the time during which commodities are delayed.

The *Process* activity is equivalent to having a *Queue* followed by a *Serve*. It receives commodities in input, queue them up to the queue capacity, delays (serve) them, and outputs them.

The *Produce* activity generates commodities and outputs them. The time between the creations is determined by the *intergenerationTime* property. The activity also has a *limit* property that sets the maximum number of commodities created by the activity's executions, and a *generateAtStart* property indicating if a commodity is generated at the very beginning of the simulation.

The *Queue* activity is a specialization of *ActivityWithInputAndOutput* that takes input commodities and queues them until they can be output (accepted by the next block). The activity stops accepting incoming commodities if the capacity is reached. It has a *capacity* property indicating how many commodities can be stored in the queue at any point in time, and a *discipline* property indicating in which order commodities are sent out.

The *Release* activity is a specialization of *ActivityWithResourcePool* that takes a commodity as input, provides it as output while releasing a resource (from the provided resource pool) that had been seized by a *Seize* activity.

The *Replicate* activity is a specialization of *ActivityWithInput* that takes a commodity in input, duplicates it, and outputs the original object as well as the copy.

The *Resource* block represents a resource that can serve or process commodities.

The *ResourcePool* block represents a group of *resources* that are available to serve or process commodities. The values of that property change over time as resources start and stop processing commodities.

The *Seize* activity is a specialization of *ActivityWithResourcePool* that takes a commodity as input, removes a resource from the provided resource pool whenever one is available, and then outputs the commodity.

The *Serve* activity takes (accepts) input commodities and outputs (offers) them after a specified

service time. It has a capacity to indicate how many commodities can be serviced at the same time, and it can refer to a resource pool (unlike a delay). It is a combination of the *Seize*, *Delay*, and *Release* activities.

The *Split* activity is a specialization of *ActivityWithInput* that splits an input commodity into multiple commodities corresponding to the input commodity's attribute values. In the actions typed by it, the number of outputs (output pins on the call behavior action) must match the number of attributes (inherited and owned) in the input commodity type.

The *Transform* activity is a specialization of *ActivityWithInputAndOutput* that consumes a commodity as input and produces a new commodity as output. Its *postCreation* operation can be redefined to add an OpaqueBehavior method setting attributes of the output commodity based on the input commodity.

All concrete activities in this library are disjoint and exclusive, which means no action can be directly or indirectly typed by more than one of them.

4.2.2 Discrete event expression language

This subsection describes a tool-independent DENs expression language, to be used in opaque behaviors associated with operations, or in activity edge guards. Terminal and non-terminal symbols of the grammar are presented in Subsections 4.2.2.1 and 4.2.2.2, respectively. The grammar is written using the ANTLR4 language [7], which is very similar to EBNF.

4.2.2.1 Terminal symbols

The following code shows the terminal symbols of the grammar.

```
TILDE : '~';
NOT : '!';
PLUS : '+';
MINUS : '-';
MUL : '*';
DIV : '/';
LEFTDIV : '\\';
POW : '^';
TRANSPOSE : '\'';
EMUL : '.*';
EDIV : './';
ELEFTDIV : '.\\';
EPOW : '.^';
ETRANSPOSE : '.\'';

EXPR_LE : '<=';
EXPR_EQ : '==';
EXPR_NE : '!='|'~=';
EXPR_GE : '>=';
EXPR_AND : '&';
EXPR_OR : '|';
```

```
EXPR_AND_AND : '&&';
EXPR_OR_OR : '||';
EXPR_LT : '<';
EXPR_GT : '>';

EQ : '=';

LPAR : '(';
RPAR : ')';

LCUR : '{';
RCUR : '}';

LBRA : '[';
RBRA : ']';

DOT: '.';
COLON: ':';

GLOBAL : 'global';
PERSISTENT : 'persistent';

IF: 'if';
ELSE: 'else';
ELSEIF: 'elseif';
ENDIF: 'endif';

SWITCH: 'switch';
CASE: 'case';
OTHERWISE: 'otherwise';
ENDSWITCH : 'endswitch';

DO : 'do';
UNTIL: 'until';

WHILE : 'while';
ENDWHILE: 'endwhile';

FOR : 'for';
ENDFOR : 'endfor';

BREAK : 'break';
CONTINUE : 'continue';
FUNC_RET : 'return';

SPMD : 'spmd';

TRY: 'try';
CATCH: 'catch';
END_TRY_CATCH: 'end_try_catch';

FUNCTION: 'function';
ENDFUNCTION: 'endfunction';
```

```
fragment
D               : [0-9]
                ;

fragment
D_              : [0-9_]
                ;

S               : [ \t]
                ;

NL              : ('\n' | '\r' '\n'? )
                ;

IDENT   : [_$a-zA-Z][_$a-zA-Z0-9]*;

fragment
DECIMAL_DIGITS  : D D_*
                ;

fragment
EXPONENT        : [DdEe] [+-]? DECIMAL_DIGITS
                ;

fragment
REAL_DECIMAL    : (( DECIMAL_DIGITS DOT?)
                | (DECIMAL_DIGITS? DOT DECIMAL_DIGITS)) EXPONENT?
                ;

fragment
IMAG_UNIT    : [IiJj]
                ;

DECIMAL_NUMBER  : REAL_DECIMAL IMAG_UNIT?
                ;

SIZE_SUFFIX         : [su] ('8'|'16'|'32'|'64')
                    ;

fragment
BINARY_BITS         : '0' [bB][01][01_]*
                    ;

BINARY_NUMBER       : BINARY_BITS SIZE_SUFFIX?
                    ;

fragment
HEXADECIMAL_BITS    : '0' [xX][0-9a-fA-F][0-9a-fA-F_]*
                    ;

HEXADECIMAL_NUMBER  : HEXADECIMAL_BITS SIZE_SUFFIX?
                    ;
```

```
BLOCK_COMMENT     : ( '%{' S* NL .*? '%}' S* NL
                  | '#{' S* NL .*? '#}' S* NL)-> skip
                  ;

ELLIPSIS          : '...' ~[\r\n]* NL -> skip
                  ;

LINE_COMMENT      : ( '%' ~[\r\n]*
                  | '#' ~[\r\n]*)-> skip
                  ;

fragment
ESCAPE            : '\\' [abfnrtv"'\\]
                  | '\\' [0-7][0-7][0-7]
                  | '\\' 'x'+ [0-9a-fA-F]+
                  ;

SQ_STRING         : '\'' (~['\r\n])* '\''
                  ;

DQ_STRING         : '"' (~["\\] | ESCAPE | '\\' NL)* '"'
                  ;
```

Note that NL is not recognized when inside parenthesis, but it is recognized when inside squared or curly brackets.

4.2.2.2 Non-terminal symbols

The following code shows the non-terminal nodes of the grammar.

```
body_input      : list?
                ;

expression_input: expression
                ;

simple_list     : statement (sep_no_nl statement)* sep_no_nl?
                ;

list            : statement (sep statement)* sep?
                ;

fcn_list        : function (sep? function)* sep?
                ;

statement       : expression
                | command
                | word_list_cmd
                ;

word_list_cmd   : identifier word_list
                ;
```

```
word_list       : string+
                ;

identifier      : IDENT (DOT IDENT)*
                ;

string          : (DQ_STRING|SQ_STRING)+
                ;

constant        : DECIMAL_NUMBER
                | BINARY_NUMBER
                | HEXADECIMAL_NUMBER
                | string
                ;

matrix          : LBRA matrix_rows? RBRA
                ;

matrix_rows     : cell_or_matrix_row ((';'|NL) cell_or_matrix_row)*
                ;

cell            : LCUR cell_rows? RCUR
                ;

cell_rows       : cell_or_matrix_row ((';'|NL) cell_or_matrix_row)*
                ;

cell_or_matrix_row
                : ','? arg (','? arg)* ','?
                ;

primary_expr    : identifier identifier*
                | constant
                | matrix
                | cell
                | LPAR expression RPAR
                ;

arg_list        : arg (',' arg)*
                ;

arg             : expression
                ;

oper_expr       : primary_expr
                | oper_expr (LPAR arg_list? RPAR| LCUR arg_list? RCUR
                  | DOT LPAR expression RPAR)
                | oper_expr (trans_oper | pow_oper power_expr)
                |<assoc=right> unary_oper oper_expr
                | oper_expr multip_oper oper_expr
                | oper_expr addit_oper oper_expr
                ;
```

```
addit_oper      : MINUS
                | PLUS
                ;

multip_oper     : MUL
                | DIV
                | LEFTDIV
                | EMUL
                | EDIV
                | ELEFTDIV
                ;

unary_oper      : TILDE
                | NOT
                | PLUS
                | MINUS
                ;

pow_oper        : POW
                | EPOW
                ;

trans_oper      : TRANSPOSE
                | ETRANSPOSE
                ;

power_expr      : primary_expr
                | power_expr (LPAR arg_list? RPAR | LCUR arg_list? RCUR
                  | DOT LPAR expression RPAR)
                |<assoc=right> unary_oper power_expr
                ;

compare_oper    : EXPR_LT
                | EXPR_LE
                | EXPR_EQ
                | EXPR_NE
                | EXPR_GE
                | EXPR_GT
                ;

simple_expr     : oper_expr ( COLON oper_expr (COLON oper_expr)? )?
                | simple_expr compare_oper simple_expr
                | simple_expr EXPR_AND simple_expr
                | simple_expr EXPR_OR simple_expr
                | simple_expr EXPR_AND_AND simple_expr
                | simple_expr EXPR_OR_OR simple_expr
                ;

expression      : simple_expr
                |<assoc=right> expression assign_oper expression
                ;

assign_oper     : EQ
```

```
                ;

command         : select_command
                | loop_command
                | jump_command
                | except_command
                | function
                ;

decl_elt        : identifier (EQ expression)?
                ;

select_command  : if_command
                | switch_command
                ;

if_command      : IF expression sep? list? elseif_clause* else_clause?
                  ENDIF
                ;

elseif_clause   : ELSEIF sep? expression sep? list?
                ;

else_clause     : ELSE sep? list?
                ;

switch_command  : SWITCH expression sep? case_list? ENDSWITCH
                ;

case_list       : switch_case+ default_case?
                | default_case
                ;

switch_case     : CASE sep? expression sep? list?
                ;

default_case    : OTHERWISE sep? list?
                ;

loop_command    : WHILE expression sep? list?
                | DO sep? list? UNTIL expression
                | FOR simple_expr EQ expression sep? list? ENDFOR
                ;

jump_command    : BREAK
                | CONTINUE
                | FUNC_RET
                ;

except_command  : UNWIND_PROTECT sep? prt=list? UNWIND_PROTECT_CLEANUP
                  sep? cln=list? END_UNWIND_PROTECT
                | TRY sep? tr=list? CATCH sep? ct=list? END_TRY_CATCH
                ;
```

```
param_list      : LPAR (param_list_elt (',' param_list_elt)*)? RPAR
                ;

param_list_elt  : decl_elt
                ;

return_list     : LBRA (identifier (',' identifier)* )? RBRA
                | identifier
                ;

fcn_name        : identifier
                ;

function        : FUNCTION (return_list EQ)? fcn_name param_list? sep?
                  function_body ENDFUNCTION  // MATLAB: end
                ;

function_body   : list?
                ;
                ;

sep_no_nl       : (','| ';')+
                ;

nl              : NL+
                ;

sep             : (','|';'| NL)+
                ;
```

4.2.2.3 Implementation notes

The grammar presented in this Subsection is largely compatible with Octave, but requires transformations for SimEvents when using `end` commands and operators, as shown in Table 3.

Octave keyword or operator	MATLAB equivalent
`endif`	`end`
`endswitch`	`end`
`endwhile`	`end`
`endfor`	`end`
`end_try_catch`	`end`
`endclassdef`	`end`
`endproperties`	`end`
`endmethods`	`end`
`!=`	`~=`
`!`	`~`

Table 3: Octave keywords and operators equivalent in MATLAB

The following is a non-exhaustive list of translations from the expression language to Java:

- The type of the values being assigned to expression language variables translate to the type of Java variables, which are declared with their type at the beginning of the scope in which they are used for the first time. The types can be boolean, integer scalar/vector/matrix (int, int[], or int[][]), real scalar/vector/matrix (double, double[], or double[][]), string, object, or cell array.
- Expression language ranges translate to Java functions that return an array of integer or real (`IntStream.iterate` and `DoubleStream.iterate` respectively).
- The expression language `rand` function translates to `Math.random()`.
- The expression language `length` function on an array translates to the `.length` property call of the array object.
- Indices for vectors and matrices in the expression language are decremented by 1 when translating to Java. For example, matrix value retrievals in the expression language (such as m(2,1)) translate to the equivalent code in Java (such as m[1][0]).

4.3 Discrete Event Network example

The example in Figure 24 is adapted from a SimEvents model in [8]. It includes physical and informational commodities (orders). The *ProcessFlowRateModel* activity contains 9 call behavior actions typed by the following 9 activities (since there is exactly 1 action per activity, the activity name is used instead of the action name for brevity):

- *LTLShip* produces order messages for Less-than-Truckload (LTL) goods (pallets and mixed pallets), and consumes the actual goods in return. The order messages are sent to *LTLOA*, from which it receives the LTL goods.
- *LTLOA* (LTL order assembly) receives LTL order messages, and redirects them to *MixedPalletAssembly* or *PalletStore*. In return, *LTLOA* receives mixed pallets and pallets respectively.
- *ParcelShip* produces order messages for Parcel goods (cartons, mixed cartons, and eaches), and consumes the actual goods in return. The order messages are sent to *ParcelOA*, from which it receives the Parcel goods.
- *ParcelOA* (Parcel order assembly) receives Parcel order messages, and redirects them to *PalletStore*, *EachStore*, or *MixedCartonAssembly*. In return, *ParcelOA* receives cartons, eaches, and mixed cartons respectively.
- *MixedPalletAssembly* receives mixed pallet orders from *LTLOA*, and sends out carton messages to *PalletStore* and mixed carton messages to *MixedCartonAssembly*. It receives cartons and mixed cartons in return, respectively, and sends mixed pallets back to *LTLOA*.
- *MixedCartonAssembly* receives mixed cartons orders from *MixedPalletAssembly* and *ParcelOA*, and sends out each messages to *EachStore*. It receives eaches in return and sends mixed cartons back to *MixedPalletAssembly* and *ParcelOA*.

- *PalletStore* receives pallet orders from *LTLOA*, and sends pallets in return. It also receives cartons messages from *ParcelOA*, *MixedPalletAssembly*, and *EachStore* and sends out cartons in return. Lastly, it receives pallets from *OriginalPalletStock*.
- *EachStore* receives each orders from *MixedCartonAssembly* and *ParcelOA*, and sends eaches to these activities in return. It also sends cartons messages to *ParcelOA* and receives cartons in return.
- *OriginalPalletStock* sends out pallet goods to *PalletStore*.

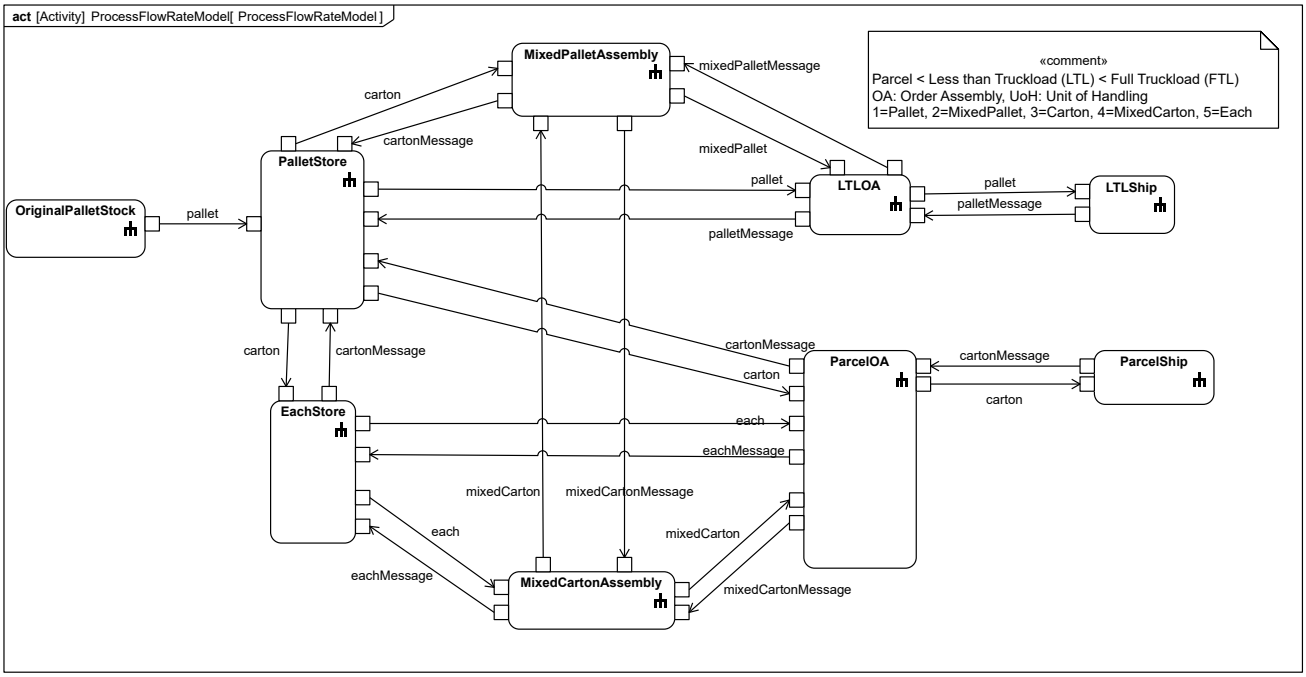

Figure 24: Discrete Event Simulation, activity, process flow rate model

All activities except *OriginalPalletStock* record inflows and outflows of goods and deal with messages and goods in a different way:

- *LTLShip* creates outgoing pallet order messages with a random number of cartons, and destroys incoming pallets (they are considered to be "shipped" and are no longer in the network).
- *LTLOA* converts 90% of outgoing pallet order messages into mixed pallet order messages, and it converts incoming mixed pallets into pallets (should be generated in LTL ship?).
- *ParcelShip* creates outgoing carton order messages, and destroys incoming cartons (they are considered to be shipped and are no longer in the network).
- *ParcelOA* converts 30% of carton order messages into mixed carton order messages, and 10% of carton order messages into each order messages. In return, it converts incoming eaches and mixed cartons into cartons.

- *MixedPalletAssembly* converts incoming mixed pallet order messages into outgoing carton order messages, and converts incoming cartons into outgoing mixed pallets (23-to-1 quantity ratio). 75% of carton order messages are sent to *PalletStore*, while the rest is sent to *MixedCartonAssembly*.
- *MixedCartonAssembly* converts incoming mixed cartons orders into each order messages. In return, it converts eaches into cartons (10-to-1 quantity ratio) and sends 13% of them to *MixedPalletAssembly* and the rest to *ParcelOA*.
- *PalletStore* converts pallet order messages into pallets, and carton order messages into cartons. 11% of these cartons are directed to *MixedPalletAssembly*, 51% are directed to *ParcelOA*, and the rest is directed to *EachStore*.
- *EachStore* converts each order messages into eaches. 22% of these eaches are directed to *ParcelOA*, and the rest is directed to *MixedCartonAssembly*.

Input and output pins in all these activities are of two kinds: *Flow* and *Message*, as shown Figure 25). *Message* has three attributes:

- *quantity* of multiplicity 1 and type Integer;
- *UoH* (unit of handling), of multiplicity 1 and type Integer. It has 5 different values: 1 for pallets, 2 for mixed pallets, 3 for cartons, 4 for mixed cartons, and 5 for eaches;
- *cartonQuantity*, determines the number of cartons used for pallets (unused).

Flow has two attributes: *quantity* and *UoH*, with the same meaning, type, and multiplicity as *Message*.

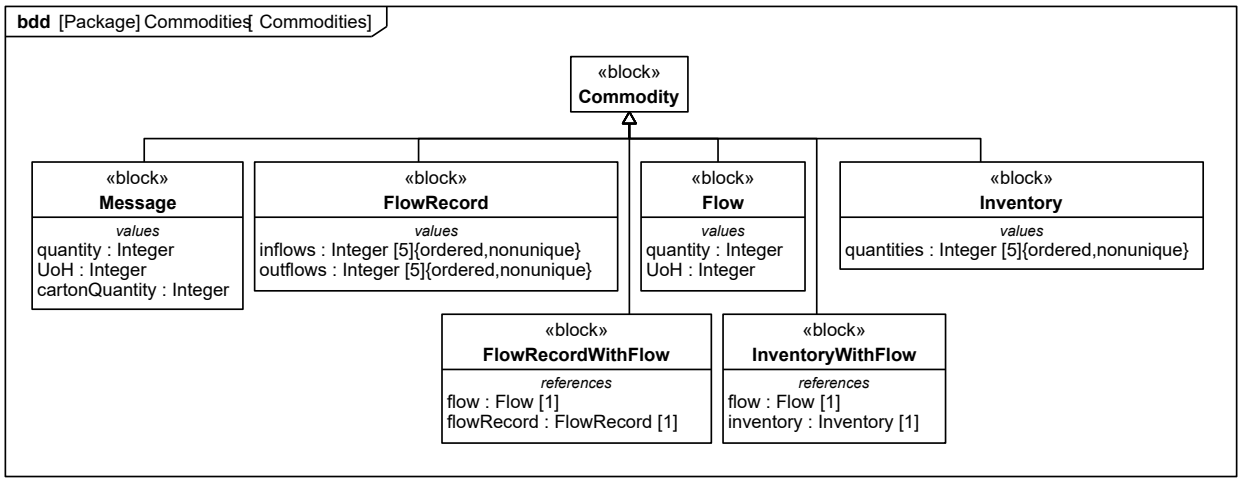

Figure 25: Discrete Event Simulation, activity, commodities

Within activities, *FlowRecord* is used to record the incoming and outgoing flows. It has two attributes of type Integer and multiplicity 5 (one for each UoH): *inflows* and *outflows*. *Inventory* is

used to record the current inventory in the activity. It has one attribute of multiplicity 5 and of type Integer: *quantities*. Two additional types are used to combine a flow records and inventories with flows: *FlowRecordWithFlow* has an attribute typed by *FlowRecord* and one typed by *Flow*, while *InventoryWithFlow* has an attribute typed by *Inventory* and one typed by *Flow*.

There are 6 specializations of DES *Source* shown in Figure 26: *DummyMessageSource*, *PalletMessageSource*, *CartonMessageSource*, *PalletSource*, *InventorySource*, *FlowRecordSource*. The last 5 sources redefine the *preExit()* operation to add an OpaqueBehavior method that sets the initial values of the created objects.

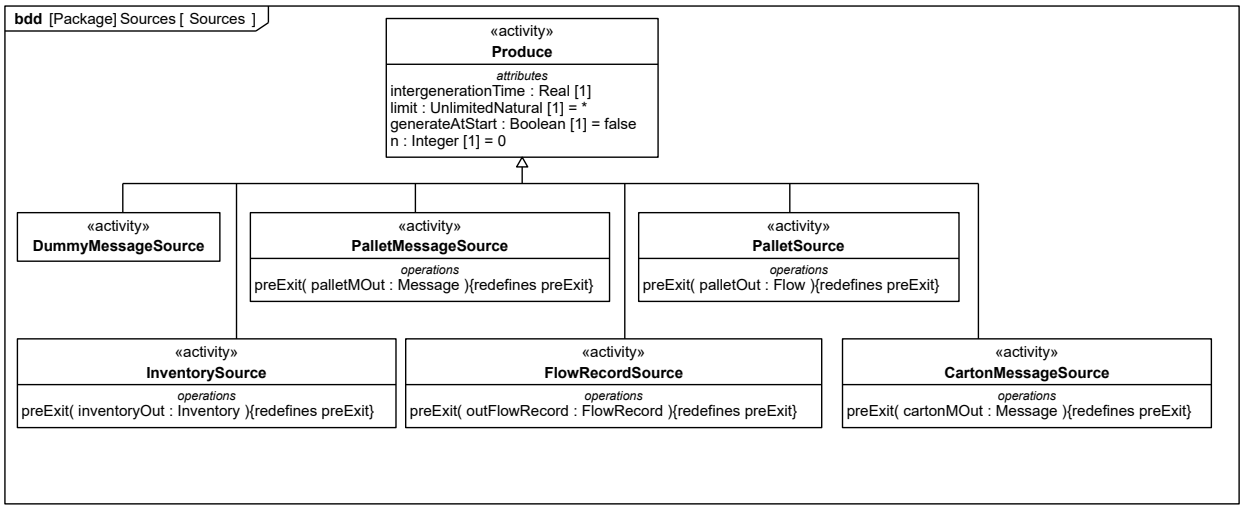

Figure 26: Discrete Event Simulation, activity, sources

There are 11 specializations of DES *Transform* shown in Figure 27, which are used to convert objects from one type to another : *EachToMixedCarton*, *EachToCarton*, *MixedCartonToCarton*, *MixedPalletToPallet*, *FlowToMixedPallet*, *MessageToFlow*, *PalletMessageToMixedPalletMessage*, *CartonMessageToMixedCartonMessage*, *PalletMessageToCartonMessage*, *CartonMessageToEachMessage1*, and *CartonMessageToEachMessage*. All these sources redefine the *preExit()* operation to add an OpaqueBehavior method that sets the initial values of the created objects based on the input object.

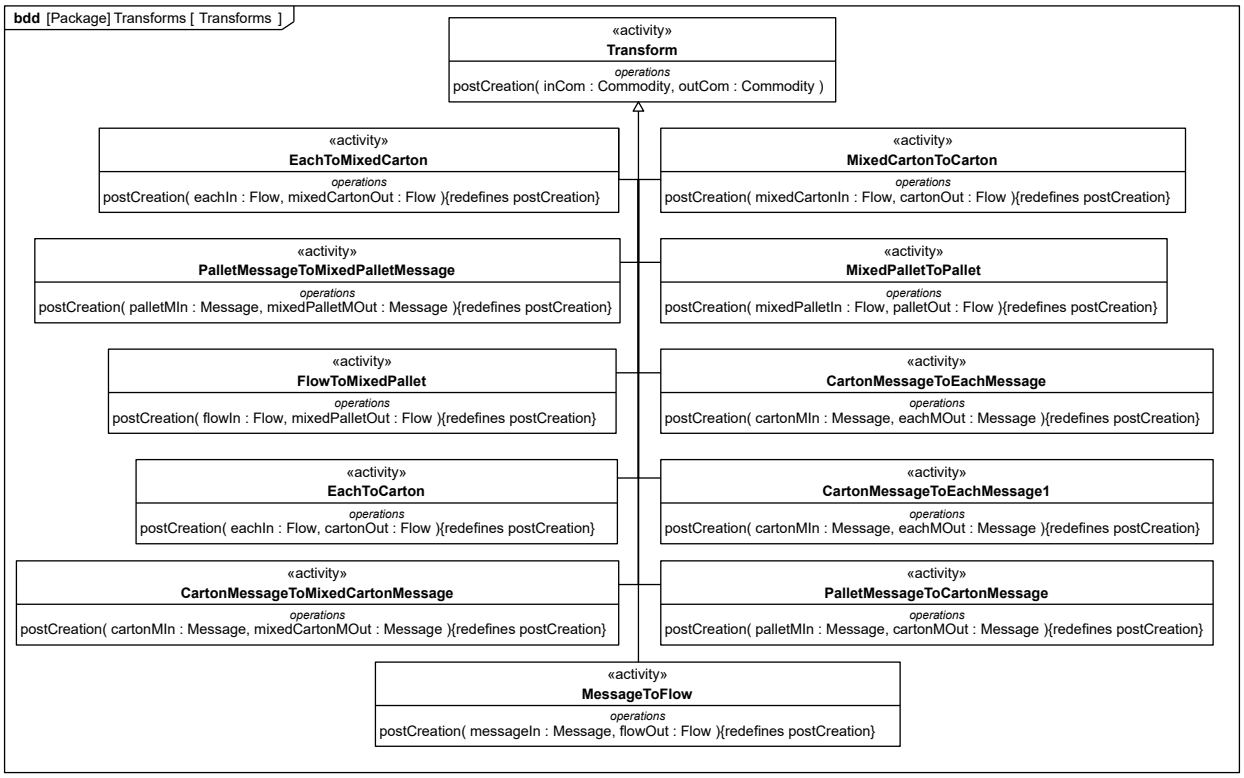

Figure 27: Discrete Event Simulation, activity, transforms

There are 4 specializations of *Server* shown in Figure 28 to make calculations on *FlowRecord* and *Inventory* objects . These specializations redefine the *preExit()* operation to add an opaque expression specifying the calculation. *InflowInventoryCalculation* and *OutflowInventoryCalculation* remove and add respectively the quantity of the *Flow* to the *quantities* attribute of the *Inventory*. *InflowCalculation* and *OutflowCalculation* add the quantity of the *Flow* to the *inflows* and *outflows* attributes of the *FlowRecord* object, respectively.

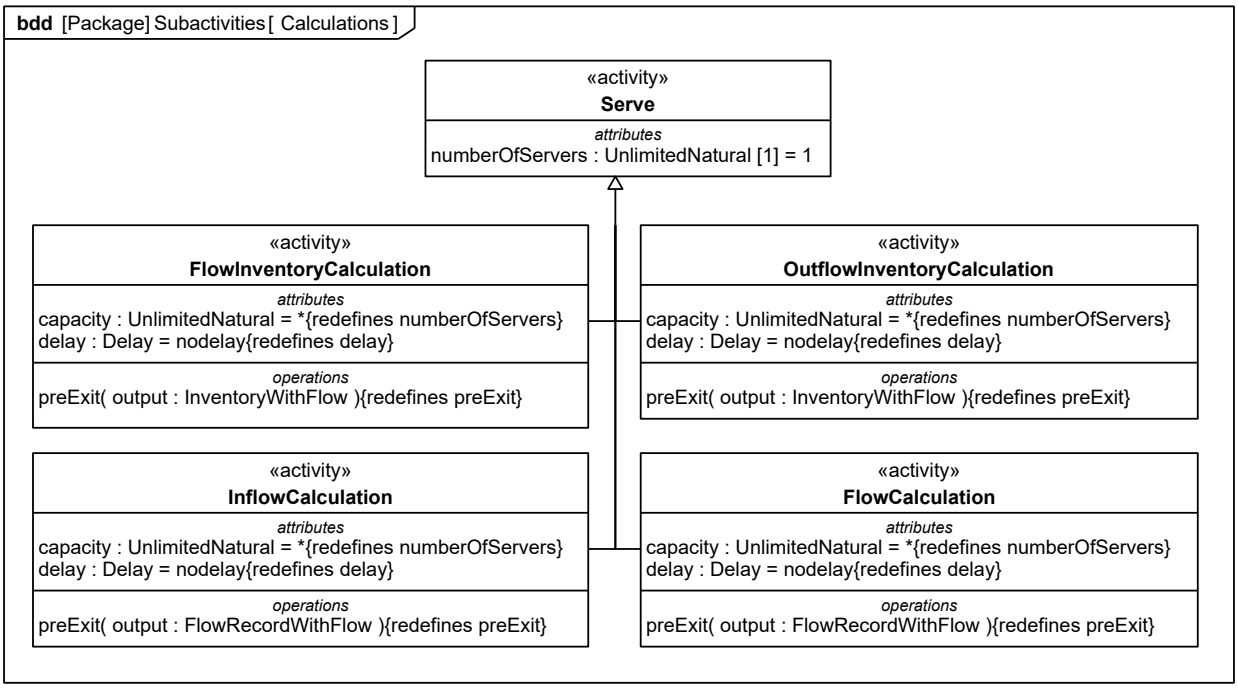

Figure 28: Discrete Event Simulation, activity, calculations

4.4 Translating between Discrete Event Networks and Discrete Event Simulation models

The primitives of DENs correspond to DES tool elements, so as a result, DENs can be translated as DES models.

4.4.1 Additional SysML model constraints for translation

The model must contain only elements covered in this translation subsection. That means activities may only contain:

- CallBehaviorActions with the activities from the DEN library as behavior (with concrete commodity types for output pins of *Produce* and *Transform* call), or other activities that meet the requirements.
- ActivityParameterNodes and Parameters typed by Commodity or one of its specializations.
- MergeNodes and DecisionNodes.
- OpaqueActions with an input and output pin typed by Commodity or one of its specializations, and the `syslma` language.
- Input pins, output pins, and activity parameters with multiplicity 0..*.

- Input and output pins that are streaming..
- CallBehaviorActions for *Produce* and *Transform* with output pins typed by concrete commodity types.
- ObjectFlows between ActivityParameterNodes, pins, and MergeNodes and DecisionNodes.

Activities and CallBehaviorActions enable DENs to call other DENs, making the flattening process in translating CFN and PN unnecessary for DENs.

4.4.2 Discrete Event Networks and Discrete Event Simulation models

This subsection presents how the DEN terms correspond to DES terms.

Property values of DEN activities and their specializations are selected using the method described in Subsection 2.4.2.2. Adjunct properties for actions are needed to specify initial values.

4.4.3 SimEvents translation

SimEvents uses *`systems`* as equivalent to UML classes or SysML blocks, and *`blocks`* as equivalent to UML/SysML properties. Subsystem blocks are blocks that refer to user-defined systems, while other blocks are defined in the Simulink library. The DEN activity nodes generally translate into blocks from the SimEvent library.

4.4.3.1 Model

SimEvents models are serialized using the OpenXML format. This subsection shows how the translated content is represented in this format, referring to the eXtensible Markup Language (XML) elements and attributes found in the files, which might differ from the terms in SimEvents.

The following code shows the root element of the SimEvent model file.

```
<ModelInformation Version="1.0">
  <Model>
    <P Name="ModelUUID">...</P>
    <P Name="SLXCompressionType">Normal</P>
    <P Name="LastSavedArchitecture">win64</P>
    <System Ref="system_root"/>
  </Model>
</ModelInformation>
```

4.4.3.2 Activities and blocks

`Systems` are user-defined elements that may correspond to SysML blocks or activities. The following code shows a `System` element in XML.

```
<System>
  <P Name="Location">[0,0,0,0]</P>
  <P Name="Open">on</P>
  <P Name="SetExecutionDomain">off</P>
  <P Name="ExecutionDomainType">Deduce</P>
  <P Name="ReportName">simulink-default.rpt</P>
  <P Name="SIDHighWatermark">13</P>
  <P Name="SimulinkSubDomain">Simulink</P>
  ...
</System>
%\end{lstlisting}
```

4.4.3.3 Commodities

CFN commodities translate into structured entities in SimEvents, but this is not a 1-to-1 translation. Because two distinct Entity types are always incompatible and SimEvents require its connections to be between compatible types, Entity types are inferred based on attribute names, types, and multiplicities so that two commodity types with the same structure correspond to the same SimEvents Entity type.

Entity types are defined in the Source blocks that create instances, see next.

4.4.3.4 Produce CallBehaviorAction

A *CallBehaviorAction* typed by *Produce* or one of its specializations is translated as an `EntityGenerator` block in SimEvents. The name of the call behavior action is given to the block.

The behavior of the *preExit()* operation of the *Produce* is translated into code for the *`GenerateAction`* parameter, replacing the reference to the *CallBehaviorAction*'s input pin name by *`entity`*.

The inter-generation time of the *Produce* is translated as follows:

1. If the *Produce* action has a *limit* of 0, the *`TimeSource`* parameter of the `EntityGenerator` block is set to *`Dialog`*, the *`Period`* parameter is set to *`inf`*, and the *`GenerateEntityAtSimulationStart`* parameter is set to *`on`*;
2. then if the *Produce* action has a *limit* of 1, the *`TimeSource`* parameter is set to *`Dialog`*, the *`Period`* parameter is set to *`inf`*, and the *`GenerateEntityAtSimulationStart`* parameter is set to *`off`*;
3. otherwise
   (a) if the *interarrivalTime* property or corresponding slot is not stereotyped by a specialization of EventDistribution, the *`TimeSource`* parameter is set to *`Dialog`*, and the *`Period`* parameter is set to *`1/meantime`*;
   (b) if the *interarrivalTime* property or corresponding slot is stereotyped by Beta with parameters p and q, the *`TimeSource`* parameter is set to *`MATLAB action`*, and the *`Intergeneration-TimeAction`* parameter is set to *`dt = betarnd(p, q);`*;

(c) if the *interarrivalTime* property or corresponding slot is stereotyped by Erlang with parameters k and lambda, the `TimeSource` parameter is set to `MATLAB action`, and the `IntergenerationTimeAction` parameter is set to `dt = (-lambda) * sum(log(1-rand(1, k)));` . If *inverse* is true, `1/lambda` is used instead of `lambda`;

(d) if the *interarrivalTime* property or corresponding slot is stereotyped by Exponential with parameter lambda, the `TimeSource` parameter is set to `MATLAB action`, and the `IntergenerationTimeAction` parameter is set to `dt = (-lambda) * log(1 - rand);` . If *inverse* is true, `1/lambda` is used instead of `lambda`;

(e) if the *interarrivalTime* property or corresponding slot is stereotyped by Gamma with parameter alpha and beta, the `TimeSource` parameter is set to `MATLAB action`, and the `IntergenerationTimeAction` parameter is set to `dt = gamrnd(alpha, beta);` ;

(f) if the *interarrivalTime* property or corresponding slot is stereotyped by LogNormal with parameters mu and sigma, the `TimeSource` parameter is set to `MATLAB action`, and the `IntergenerationTimeAction` parameter is set to `dt = lognrnd(mu, sigma);` ;

(g) if the *interarrivalTime* property or corresponding slot is stereotyped by Pareto with parameters alpha and min, the `TimeSource` parameter is set to `MATLAB action`, and the `IntergenerationTimeAction` parameter is set to `dt = gprnd( 1 / alpha, min * alpha, min * alpha);` ;

(h) if the *interarrivalTime* property or corresponding slot is stereotyped by Rayleigh with parameter sigma, the `TimeSource` parameter is set to `MATLAB action`, and the `IntergenerationTimeAction` parameter is set to `dt = raylrnd(sigma);` ;

(i) if the *interarrivalTime* property or corresponding slot is stereotyped by Triangular with parameters min, max and mode, the `TimeSource` parameter is set to `MATLAB action`, and the `IntergenerationTimeAction` parameter is set to `dt = random(makedist('Triangular', 'A', min, 'B', mode, 'C', max));` ;

(j) if the *interarrivalTime* property or corresponding slot is stereotyped by Uniform with parameters min and max, the `TimeSource` parameter is set to `MATLAB action`, and the `IntergenerationTimeAction` parameter is set to `dt = unifrnd(min, max);` ;

(k) if the *interarrivalTime* property or corresponding slot is stereotyped by Weibull with parameters alpha and beta, the `TimeSource` parameter is set to `MATLAB action`, and the `IntergenerationTimeAction` parameter is set to `dt = wblrnd(alpha, beta);` .

The parameter `EntityType` is given the `Structured` value, and the parameter `EntityTypeName` is given the type name as value.

The attributes of the type of the *CallBehaviorAction*'s output pin are listed in the `AttributeName` parameter, and their initial values are listed in the `AttributeInitialValue` parameter. These lists are separated by a vertical bar character ("|").

On top of the type attributes, additional attributes are added:

- `priority`: contains a number representing the priority of the corresponding commodity, to be used by the upcoming priority queue;

- *decision*: contains the calculated value to be used by an upcoming `OutputSwitch`, since the output switch itself cannot do calculations;
- *type*: contains a integer identifier , unique for the corresponding commodity;
- *preempted*: contains an integer used for sorting elements by arrival in the queue, in case of preemption;
- *residualtime*: contains the time left to serve a commodity, when that commodity has been preempted.

The following code shows the content for a *Produce* action in XML.

```
<Block BlockType="EntityGenerator" Name="bsource" SID="1">
  <P Name="Ports">[0,1]</P>
  ...
  <P Name="OutputPortMessageModes">m</P>
  <P Name="OutputPortMap">o3</P>
  <P Name="InputPortMap"/>
  <P Name="TimeSource">MATLAB action</P>
  <P Name="IntergenerationTimeAction">persistent rngInit;
if isempty(rngInit)
rng(1270254173);
rngInit=true;
end
dt = -(1/2.0) * log(1-rand);</P>
  <P Name="GenerateEntityAtSimulationStart">on</P>
  <P Name="GenerateAction">entity.v = 1.5;</P>
  <P Name="AttributeName">decision|type|v</P>
  <P Name="AttributeInitialValue">0|1920375797|0.0</P>
  <P Name="EntityType">Structured</P>
  <P Name="EntityTypeName">E__v1Integer</P>
</Block>
```

4.4.3.5 Queue CallBehaviorAction

A *CallBehaviorAction* typed by *Queue* or one of its specializations is translated as a `Queue` block. The name of the call behavior action is given to the block.

The capacity of the *Queue* activity is provided as the value of the *Capacity* parameter if this capacity is not unlimited. If the capacity is unlimited, the parameter is given the value *inf*. The queue discipline of the *Queue* activity is provided as value of the *QueueType* parameter, such that the values are *FIFO*, *LIFO*, or *Priority*. When the discipline is *Priority*, the *PrioritySource* parameter is set to *priority*, the *SortingDirection* parameter is set to *Descending*, and the *EntryAction* parameter is set to *entity.priority=* followed by the content of the *getPriority()* operation from the Activity typing the *CallBehaviorAction*.

The behavior of the *postEntry()* operation of the *Queue* is translated into code for the *EntryAction* parameter, replacing the reference to the *CallBehaviorAction*'s input pin name by *entity*.

If the behavior has a *preExit()* operation, a "zero-time" `EntityServer` block is introduced after the `Queue` and the operation is translated into code in the *`EntryAction`* parameter of the new `EntityServer`. In that code, the *CallBehaviorAction*'s input pin name is replaced by *`entity`*. The `Queue`'s output is connected to the `EntityServer`'s input, and the outgoing connections from the `Queue` become connections from the `EntityServer`.

The *`InputPortMap`* parameter is given the value *`u0`*, and the *`OutputPortMap`* is given the value *`o5`*. The following code shows the content for a *Queue* action in XML.

```
<Block BlockType="Queue" Name="aqueue" SID="4">
  <P Name="Ports">[1,1]</P>
  ...
  <P Name="InputPortMessageModes">m</P>
  <P Name="InputPortMap">u0</P>
  <P Name="OutputPortMessageModes">m</P>
  <P Name="OutputPortMap">o5</P>
  <P Name="Capacity">50</P>
</Block>
```

4.4.3.6 Seize CallBehaviorAction

A CallBehaviorAction typed by *Seize* or one of its specializations is translated as a `EntityResourceAcquirer` block. The name of the call behavior action is given to the block. The name of the resource pool of the *Seize* activity is provided as value of the *`ResourceName`* parameter of the block.

The behavior of the *postEntry()* operation of the *Seize* is translated into code for the *`EntryAction`* parameter, replacing the reference to the *CallBehaviorAction*'s input pin name by *`entity`*.

If the behavior has a *preExit()* operation, a zero-time `EntityServer` block is introduced after the `EntityResourceAcquirer` and the operation is translated into code in the *`EntryAction`* parameter of the new `EntityServer`. In that code, the *CallBehaviorAction*'s input pin name is replaced by *`entity`*. The `EntityResourceAcquirer`'s output is connected to the `EntityServer`'s input, and the outgoing connections from the `EntityResourceAcquirer` become connections from the `EntityServer`.

The *`InputPortMap`* parameter is given the value *`u0`*, and the *`OutputPortMap`* is given the value *`o3`*. The following code shows the content for a *Seize* action in XML.

```
<Block BlockType="EntityResourceAcquirer" Name="bseize" SID="7">
  <P Name="Ports">[1,1]</P>
  ...
  <P Name="InputPortMap">u0</P>
  <P Name="OutputPortMap">o3</P>
  <P Name="ResourceName">rp</P>
</Block>
```

4.4.3.7 Delay CallBehaviorAction

A CallBehaviorAction typed by *Delay* or one of its specializations is translated an `EntityServer`. The name of the call behavior action is given to the block.

The delay time distribution of the *Delay* is translated like the inter-generation time of the *Produce* activity, see Subsection 4.4.3.4.

The behavior of the *postEntry()* operation of the *Delay* is translated into code for the `EntryAction` parameter, replacing the reference to the *CallBehaviorAction*'s input pin name by `entity`.

The behavior of the *preExit()* operation of the *Delay* is translated into code for the `ServiceComplete-Action` parameter, replacing the reference to the *CallBehaviorAction*'s input pin name by `entity`.

The block also has the following parameters values: `Capacity` set to `Inf`, `InputPortMap` is set to `u2`, and `OutputPortMap` is set to `o9`. The following code shows the content for a *Delay* action in XML.

```
<Block BlockType="EntityServer" Name="bdelay" SID="8">
  <P Name="Ports">[1,1]</P>
  ...
  <P Name="InputPortMessageModes">m</P>
  <P Name="InputPortMap">u2</P>
  <P Name="OutputPortMessageModes">m</P>
  <P Name="OutputPortMap">o9</P>
  <P Name="Capacity">Inf</P>
  <P Name="ServiceTimeSource">MATLAB action</P>
  <P Name="ServiceCompleteAction">entity.v = entity.v * 2;</P>
  <P Name="ServiceTimeAction">persistent rngInit;
if isempty(rngInit)
rng(470516241);
rngInit=true;
end
dt = -1.2 * log(1-rand);</P>
</Block>
```

4.4.3.8 Release CallBehaviorAction

A CallBehaviorAction typed by *Release* or one of its specializations is translated as a `EntityResourceReleaser` block. The name of the call behavior action is given to the block. The name of the resource pool of the *Release* activity is provided as value of the `ResourceName` parameter of the block.

The behavior of the *postEntry()* operation of the *Release* is translated into code for the `EntryAction` parameter, replacing the reference to the *CallBehaviorAction*'s input pin name by `entity`.

If the behavior has a *preExit()* operation, a zero-time `EntityServer` block is introduced after the `EntityResourceReleaser` and the operation is translated into code in the `EntryAction` parameter of the new `EntityServer`. In that code, the *CallBehaviorAction*'s input pin name is replaced by `entity`. The `EntityResourceReleaser`'s output is connected to the `EntityServer`'s input, and the outgoing connections from the `EntityResourceReleaser` become connections from the `EntityServer`.

The block also has the following parameters values: `ResourceToRelease` set to `Selected`, `Resource-Amount` set to `1`, `InputPortMap` is set to `u0`, and `OutputPortMap` is set to `o0`. The following code shows the content for a *Release* action in XML.

```
<Block BlockType="EntityResourceReleaser" Name="brelease" SID="9">
  <P Name="Ports">[1,1]</P>
  ...
  <P Name="InputPortMap">u0</P>
  <P Name="OutputPortMap">o0</P>
  <P Name="ResourceToRelease">Selected</P>
  <P Name="ResourceName">rp</P>
  <P Name="ResourceAmount">1</P>
</Block>
```

4.4.3.9 Consume CallBehaviorAction

A CallBehaviorAction typed by *Consume* or one of its specializations is translated as an `EntityTerminator` block.

The behavior of the *postEntry()* operation of the *Consume* is translated into code for the `EntryAction` parameter, replacing the reference to the *CallBehaviorAction*'s input pin name by `entity`. The name of the call behavior action is given to the block.

The following code shows the content for a *Consume* action in XML.

```
<Block BlockType="EntityTerminator" Name="sink" SID="16">
  <P Name="Ports">[1,0]</P>
  ...
  <P Name="InputPortMap">u0</P>
  <P Name="OutputPortMap"/>
</Block>
```

4.4.3.10 Serve CallBehaviorAction

A CallBehaviorAction typed by *Serve* or one of its specializations is translated as an `EntityServer` block. The name of the call behavior action is given to the block.

The service time distribution of the *Serve* is translated like the inter-generation time of the *Produce* activity, see Subsection 4.4.3.4.

If the *Serve* activity has a resource pool, the number of resources is provided as value of the `Capacity` parameter. Otherwise, that value is set to `Inf`.

The behavior of the *postEntry()* operation of the *Serve* is translated into code for the `EntryAction` parameter, replacing the reference to the *CallBehaviorAction*'s input pin name by `entity`.

The behavior of the *preExit()* operation of the *Serve* is translated into code for the `ServiceComplete-Action` parameter, replacing the reference to the *CallBehaviorAction*'s input pin name by `entity`.

The block also has the following parameters values: `ServiceTimeSource` set to `MATLAB action`, `InputPortMap` is set to `u2`, and `OutputPortMap` is set to `o9`. The value of the service time is provided by the `ServiceTimeAction` parameter, and it is determined like the inter-generation time of the `Produce` activity, see Subsection 4.4.3.4. The following code shows the content for a *Serve* action in XML.

```
<Block BlockType="EntityServer" Name="aserver" SID="5">
  <P Name="Ports">[1,1]</P>
  ...
  <P Name="InputPortMessageModes">m</P>
  <P Name="InputPortMap">u2</P>
  <P Name="OutputPortMessageModes">m</P>
  <P Name="OutputPortMap">o9</P>
  <P Name="Capacity">2</P>
  <P Name="ServiceTimeSource">MATLAB action</P>
  <P Name="ServiceCompleteAction">entity.b.v = entity.b.v + 2;
entity.c.v = entity.c.v + 3;</P>
  <P Name="ServiceTimeAction">persistent rngInit;
if isempty(rngInit)
rng(2011968972);
rngInit=true;
end
dt = -1.5 * log(1-rand);</P>
</Block>
```

4.4.3.11 Process CallBehaviorAction

A CallBehaviorAction typed by *Process* or one of its specializations is translated as a pattern made of:

1. a `Queue` block,
2. an `EntityServer` block,
3. a line connecting the output of the queue to the input of the server

The name of the call behavior action is given to the blocks along with a suffix to distinguish them (such as ”_queue”, ”_server”).

The processing time distribution of the *Process* action is translated like the inter-generation time for the *Produce* activity, see Subsection 4.4.3.4.

If the *Process* activity has a resource pool, the name of that resource pool is provided as value of the *`Capacity`* parameter of the service block. Otherwise, that value is set to *`Inf`*.

The capacity of the *Process* activity is provided as the value of the *`Capacity`* parameter of the queue block if this capacity is not unlimited. Otherwise, that value is set to *`Inf`*.

The queue discipline of the *Process* activity is provided as value of the *`QueueType`* parameter, such that the values are *`FIFO`*, *`LIFO`*, or *`Priority`*. When the discipline is *`Priority`*, the *`PrioritySource`* parameter is set to *`priority`*, the *`SortingDirection`* parameter is set to *`Descending`*, and the *`EntryAction`* parameter is set to *`entity.priority=`* followed by the content of the *getPriority()* operation from the Activity typing the *CallBehaviorAction*.

The behavior of the *postEntry()* operation of the *Process* is translated into code for the *`EntryAction`* parameter of the `Queue` block, replacing the reference to the *CallBehaviorAction*’s input pin name by *`entity`*.

The behavior of the *preExit()* operation of the *Process* is translated into code for the `ServiceCompleteAction` parameter of the *EntityServer* block, replacing the reference to the *CallBehaviorAction*'s input pin name by `entity`.

The server block also has the following parameters values: `ServiceTimeSource` set to `MATLAB action`, `InputPortMap` is set to `u2`, and `OutputPortMap` is set to `o9`.

The queue block has the following parameter values: `InputPortMap` set to `u0`, and `OutputPortMap` set to `o5`. The following code shows the content for a *Process* action in XML.

```
<Block BlockType="Queue" Name="cprocessing_queue" SID="10">
  <P Name="Ports">[1,1]</P>
  ...
  <P Name="InputPortMessageModes">m</P>
  <P Name="InputPortMap">u0</P>
  <P Name="OutputPortMessageModes">m</P>
  <P Name="OutputPortMap">o5</P>
  <P Name="Capacity">Inf</P>
</Block>
<Block BlockType="EntityServer" Name="cprocessing_server" SID="11">
  <P Name="Ports">[1,1]</P>
  ...
  <P Name="InputPortMessageModes">m</P>
  <P Name="InputPortMap">u2</P>
  <P Name="OutputPortMessageModes">m</P>
  <P Name="OutputPortMap">o9</P>
  <P Name="Capacity">2</P>
  <P Name="ServiceTimeSource">MATLAB action</P>
  <P Name="ServiceCompleteAction">entity.v = entity.v * 3;</P>
  <P Name="ServiceTimeAction">persistent rngInit;
if isempty(rngInit)
rng(822703292);
rngInit=true;
end
dt = -1.25 * log(1-rand);</P>
</Block>
<Line>
  ...
  <P Name="Src">10#out:1</P>
  <P Name="Dst">11#in:1</P>
</Line>
```

4.4.3.12 Replicate CallBehaviorAction

A CallBehaviorAction typed by *Replicate* or one of its specializations is translated as an `EntityReplicator` block. The name of the call behavior action is given to the block.

If the behavior has a *postEntry()* operation, a zero-time `EntityServer` block is introduced before the `EntityReplicator` and the operation is translated into code in the `EntryAction` parameter of the new `EntityServer`. In that code, the *CallBehaviorAction*'s input pin name is replaced by `entity`. The `EntityServer`'s output is connected to the `EntityReplicator`'s input, and the incoming connections from the `EntityReplicator` become connections to the `EntityServer`.

The replicator block has the following parameter values: *InputPortMessageModes* set to *m*, *OutputPortMessageModes* set to *m,m*, *InputPortMap* set to *u0*, and *OutputPortMap* set to *o0,o1*. The following code shows the content for a *Replicate* action in XML.

```
<Block BlockType="EntityReplicator" Name="breplicate" SID="13">
  <P Name="Ports">[1,2]</P>
  ...
  <P Name="InputPortMessageModes">m</P>
  <P Name="OutputPortMessageModes">m,m</P>
  <P Name="InputPortMap">u0</P>
  <P Name="OutputPortMap">o0,o1</P>
</Block>
```

4.4.3.13 Combine CallBehaviorAction

A CallBehaviorAction typed by *Combine* or one of its specializations is translated as a `CompositeEntityCreator` block. The name of the call behavior action is given to the block.

A block has two input values that are set as attribute value of a newly created entity. The number of input values must match the number of attributes (owned and inherited) in the output type.

The type of the newly created entity is given as value of the *EntityTypeName* parameter.

The list of the attribute names (separated by "|") is given as value of the parameter *InputEntityName*.

If the behavior has a *preExit()* operation, a zero-time `EntityServer` block is introduced after the `CompositeEntityCreator` and the operation is translated into code in the *EntryAction* parameter of the new `EntityServer`. In that code, the *CallBehaviorAction*'s input pin name is replaced by *entity*. The `CompositeEntityCreator`'s output is connected to the `EntityServer`'s input, and the outgoing connections from the `CompositeEntityCreator` become connections from the `EntityServer`.

The composite entity creator block also has the following parameter values: *InputPortMessageModes* set to *m,m*, *OutputPortMessageModes* set to *m*, *InputPortMap* set to *u0,u1*, and *OutputPortMap* set to *o0*. The following code shows the content for a *Combine* action in XML.

```
<Block BlockType="CompositeEntityCreator" Name="acombiner" SID="3">
  <P Name="Ports">[2,1]</P>
  ...
  <P Name="InputPortMessageModes">m,m</P>
  <P Name="OutputPortMessageModes">m</P>
  <P Name="InputPortMap">u0,u1</P>
  <P Name="OutputPortMap">o0</P>
  <P Name="EntityTypeName">A</P>
  <P Name="InputEntityName">b|c</P>
</Block>
```

4.4.3.14 Split CallBehaviorAction

A CallBehaviorAction typed by *Split* or one of its specializations is translated a `CompositeEntitySplitter` block. The name of the call behavior action is given to the block.

If the behavior has a *postEntry()* operation, a zero-time `EntityServer` block is introduced before the `CompositeEntitySplitter` and the operation is translated into code in the *`EntryAction`* parameter of the new `EntityServer`. In that code, the *CallBehaviorAction*'s input pin name is replaced by *`entity`*. The `EntityServer`'s output is connected to the `CompositeEntitySplitter`'s input, and the outgoing connections from the `CompositeEntitySplitter` become connections to the `EntityServer`.

The composite entity splitter block has the following parameter values: *`InputPortMessageModes`* set to *`m`*, *`OutputPortMessageModes`* set to *`m,m`*, *`InputPortMap`* set to *`u0`*, and *`OutputPortMap`* set to *`o0,o1`*. The following code shows the content for a *Split* action in XML.

```
<Block BlockType="CompositeEntitySplitter" Name="asplitter" SID="6">
  <P Name="Ports">[1,2]</P>
  ...
  <P Name="InputPortMessageModes">m</P>
  <P Name="OutputPortMessageModes">m,m</P>
  <P Name="InputPortMap">u0</P>
  <P Name="OutputPortMap">o0,o1</P>
</Block>
```

4.4.3.15 Transform CallBehaviorAction

A CallBehaviorAction typed by *Transform* or one of its specializations is translated in the following pattern:

- an `EntityReplicator` block,
- an `EntitySource` block,
- a `CompositeEntityCreator` block,
- an `EntityServer` block,
- a `CompositeEntitySplitter` block,
- an `EntityTerminator` block,
- a line between the first output of the replicator and the source input,
- a line between the second output of the replicator and the first input of the composite entity creator,
- a line between the source output and the second input of the composite entity creator,
- a line between the composite entity creator output and the server input,
- a line between the server output and the composite entity splitter input,
- a line between the first composite entity splitter output and the sink input.

This pattern is to replicate the input entity, use the duplicate entity as a signal to create the new entity, then combine the input entity and the output entity, use a server to execute the actions defined in the *Transform*'s *preExit()* operation (such as setting values), then split the combined object and dispose of the input. The name of the call behavior action is given to each blocks along with a unique suffix.

If the behavior has a *postEntry()* operation, a "zero-time" `EntityServer` block is introduced after the `EntityResourceAcquirer` and the operation is translated into code in the *`EntryAction`* parameter of the new `EntityServer`. In that code, the *CallBehaviorAction*'s input pin name is replaced by *`entity`*. The `EntityResourceAcquirer`'s output is connected to the `EntityServer`'s input, and the outgoing connections from the `EntityResourceAcquirer` become connections from the `EntityServer`.

If the behavior has a *preExit()* operation, a zero-time `EntityServer` block is introduced after the `EntityResourceAcquirer` and the operation is translated into code in the *`EntryAction`* parameter of the new `EntityServer`. In that code, the *CallBehaviorAction*'s input pin name is replaced by *`entity`*. The `EntityResourceAcquirer`'s output is connected to the `EntityServer`'s input, and the outgoing connections from the `EntityResourceAcquirer` become connections from the `EntityServer`.

If the behavior has a *preExit()* operation, a zero-time `EntityServer` block is introduced after the `EntityResourceAcquirer` and the operation is translated into code in the *`EntryAction`* parameter of the new `EntityServer`. In that code, the *CallBehaviorAction*'s input pin name is replaced by *`entity`*. The `EntityResourceAcquirer`'s output is connected to the `EntityServer`'s input, and the outgoing connections from the `EntityResourceAcquirer` become connections from the `EntityServer`.

The parameters for these blocks are the same as shown in the previous paragraphs, with the following exceptions:

- the source block has its *`InputPortMessageModes`* parameter set to *`m`*, its *`InputPortMap`* parameter set to *`u0`*, and its *`GenerationMethod`* set to *`Event-based`*;
- the composite entity creator block is given the name of the CallBehaviorAction's input pin and output pin as the two attribute of the newly created entity;
- the *Transform*'s *preExit()* operation code is given to the server's exit action, prefixing the input/output names by *`entity.`* since the objects are accessed from the composite entities in SimEvents.

The following code shows the content for a *Transform* action in XML.

```
<Block BlockType="EntityReplicator" Name="c_to_b_replicate" SID="12">
  <P Name="Ports">[1,2]</P>
  ...
  <P Name="InputPortMessageModes">m</P>
  <P Name="OutputPortMessageModes">m,m</P>
  <P Name="InputPortMap">u0</P>
  <P Name="OutputPortMap">o0,o1</P>
</Block>
<Block BlockType="EntityGenerator" Name="c_to_b_source" SID="13">
  <P Name="Ports">[1,1]</P>
  ...
```

```
  <P Name="InputPortMessageModes">m</P>
  <P Name="OutputPortMessageModes">m</P>
  <P Name="InputPortMap">u0</P>
  <P Name="OutputPortMap">o3</P>
  <P Name="GenerationMethod">Event-based</P>
  <P Name="AttributeName">decision|type|v</P>
  <P Name="AttributeInitialValue">0|1990004466|0.0</P>
  <P Name="EntityType">Structured</P>
  <P Name="EntityTypeName">E__v1Integer</P>
</Block>
<Block BlockType="CompositeEntityCreator" Name="c_to_b_combiner"
    SID="14">
  <P Name="Ports">[2,1]</P>
  ...
  <P Name="InputPortMessageModes">m,m</P>
  <P Name="OutputPortMessageModes">m</P>
  <P Name="InputPortMap">u0,u1</P>
  <P Name="OutputPortMap">o0</P>
  <P Name="EntityTypeName">Entityc_to_b_combiner</P>
  <P Name="InputEntityName">bOut|cIn</P>
</Block>
<Block BlockType="EntityServer" Name="c_to_b_server" SID="15">
  <P Name="Ports">[1,1]</P>
  ...
  <P Name="InputPortMessageModes">m</P>
  <P Name="InputPortMap">u2</P>
  <P Name="OutputPortMessageModes">m</P>
  <P Name="OutputPortMap">o9</P>
  <P Name="Capacity">Inf</P>
  <P Name="ServiceTimeSource">MATLAB action</P>
  <P Name="ServiceCompleteAction">entity.bOut = entity.cIn + 1;</P>
  <P Name="ServiceTimeAction">dt = 0.0;</P>
</Block>
<Block BlockType="CompositeEntitySplitter" Name="c_to_b_splitter"
    SID="16">
  <P Name="Ports">[1,2]</P>
  ...
  <P Name="InputPortMessageModes">m</P>
  <P Name="OutputPortMessageModes">m,m</P>
  <P Name="InputPortMap">u0</P>
  <P Name="OutputPortMap">o0,o1</P>
</Block>
<Block BlockType="EntityTerminator" Name="c_to_b_sink" SID="17">
  <P Name="Ports">[1,0]</P>
  ...
  <P Name="InputPortMap">u0</P>
  <P Name="OutputPortMap"/>
</Block>
<Line>
  ...
  <P Name="Src">12#out:1</P>
  <P Name="Dst">13#in:1</P>
</Line>
```

```
<Line>
  ...
  <P Name="Src">13#out:1</P>
  <P Name="Dst">14#in:1</P>
</Line>
<Line>
  ...
  <P Name="Src">12#out:2</P>
  <P Name="Dst">14#in:2</P>
</Line>
<Line>
  ...
  <P Name="Src">14#out:1</P>
  <P Name="Dst">15#in:1</P>
</Line>
<Line>
  ...
  <P Name="Src">15#out:1</P>
  <P Name="Dst">16#in:1</P>
</Line>
<Line>
  ...
  <P Name="Src">16#out:2</P>
  <P Name="Dst">17#in:1</P>
</Line>
```

4.4.3.16 Other CallBehaviorActions

Other *CallBehaviorAction*s are translated as subsystem blocks, referring to the system corresponding to the *CallBehaviorAction*'s behavior. The following code shows the content for other *CallBehaviorAction*s in XML.

```
<Block BlockType="SubSystem" Name="Complete_callbehavioraction42"
    SID="24">
  <P Name="Ports">[1,1]</P>
  ...
  <P Name="RequestExecContextInheritance">off</P>
  <P Name="ContentPreviewEnabled">on</P>
  <System Ref="CompositeActivity"/>
</Block>
```

4.4.3.17 ActivityParameterNode and Parameter

An input activity parameter is translated as an `Inport` block, and an output activity parameter is translated as an `Outport` block. In the activity, an *ActivityParameterNode* corresponds to the `Inport` or `Outport` block related to its parameter (effectively equivalent to merging an activity parameter node and its parameter). The following code shows the content for *Parameter*s in XML.

```
<Block BlockType="Inport" Name="inVC" SID="27">
  ...
  <P Name="Port">1</P>
```

```
</Block>
<Block BlockType="Outport" Name="outVC" SID="28">
  ,,,
  <P Name="Port">1</P>
</Block>
```

4.4.3.18 MergeNode

A *MergeNode* is translated as an `EntityInputSwitch` block.

Since an inport can have only one incoming line, there has to be one inport per incoming line. The number of ports is given as value of the parameter *`NumberInputPorts`*. The *`InputPortMessageModes`* parameter must then have as value that number of *`m`* separated by "`,`", and the *`InputPortMap`* must have as value a list of as many *`u1`*, *`u2`*, and so on separated by `,`.

The block is also has its parameter *`OutputPortMessageModes`* set to *`m`*, and its parameter *`OutputPortMap`* set to *`o0`*. The following code shows the content for a *Merge* node in XML.

```
<Block BlockType="EntityInputSwitch" Name="Complete_mergenode37"
    SID="19">
  <P Name="Ports">[3,1]</P>
  ...
  <P Name="NumberInputPorts">3</P>
  <P Name="InputPortMessageModes">m,m,m</P>
  <P Name="InputPortMap">u1,u2,u3</P>
  <P Name="OutputPortMessageModes">m</P>
  <P Name="OutputPortMap">o0</P>
</Block>
```

4.4.3.19 DecisionNode

A *DecisionNode* is translated as an `EntityOutputSwitch` block, which has one outport per outgoing line. The number of ports is given as value of the parameter *`NumberOutputPorts`*. The *`OutputPortMessageModes`* parameter must have as value that number of *`m`* separated by "`,`", and the *`OutputPortMap`* must have as value a list of as many *`o0`*, *`o1`*, and so on separated by "`,`". The block is also has its parameter *`InputPortMessageModes`* set to *`m`*, and its parameter *`InputPortMap`* set to *`u1`*.

The output port can be selected based on three criteria:

- *Conditions* are for when the outgoing edges from the *DecisionNode* have a guard with an *OpaqueExpression*.
- *probabilities* are for when the outgoing edges from the *DecisionNode* have a guard with a *LiteralReal*.
- *Port selection code* is for when the decision node has a *`decisionInput`* behavior.

In all three cases, the `EntityOutputSwitch` block has its parameter *`SwitchingCriterion`* set to the value *`From attribute`*, and an `EntityServer` block of infinite capacity is introduced, linked to the `EntityOutputSwitch` block's input.

The switch block has its parameter *`SwitchAttributeName`* set to *`decision`*. That server block is given an exit action that assigns the proper outport number to the *`decision`* attribute of the entity:

- For *conditions*, the output port number is selected based on the guards of their corresponding outgoing edge.
- For *probabilities*, the output port number is selected based on a random number and the cumulative probabilities of the outgoing edges.
- For *port selection code*, the output port number is selected using the *`decisionInput`* of the *DecisionNode*.

The following code shows the content for a *Decision* node in XML.

```
<Block BlockType="EntityServer" Name="Complete_decisionnode39_decision"
    SID="21">
  <P Name="Ports">[1,1]</P>
  ...
  <P Name="InputPortMessageModes">m</P>
  <P Name="InputPortMap">u2</P>
  <P Name="OutputPortMessageModes">m</P>
  <P Name="OutputPortMap">o9</P>
  <P Name="Capacity">Inf</P>
  <P Name="ServiceTimeSource">MATLAB action</P>
  <P Name="ServiceCompleteAction">V=[1 2];
P=[0.5 0.5];
persistent rngInit;
if isempty(rngInit)
rng(1485106560);
rngInit=true;
end
r = rand;
idx = find(r &lt; cumsum(P), 1);
entity.decision = idx(1);</P>
  <P Name="ServiceTimeAction">dt = 0.0;</P>
</Block>
<Block BlockType="EntityOutputSwitch" Name="Complete_decisionnode39"
    SID="20">
  <P Name="Ports">[1,2]</P>
  <P Name="Position">[2845,73,2860,139]</P>
  <P Name="ZOrder">20</P>
  <P Name="OutputPortMessageModes">m,m</P>
  <P Name="OutputPortMap">o0,o1</P>
  <P Name="InputPortMessageModes">m</P>
  <P Name="InputPortMap">u1</P>
  <P Name="NumberOutputPorts">2</P>
  <P Name="SwitchingCriterion">From attribute</P>
  <P Name="SwitchAttributeName">decision</P>
```

```
</Block>
<Line>
  ...
  <P Name="Src">21#out:1</P>
  <P Name="Dst">20#in:1</P>
</Line>
```

4.4.3.20 Opaque Action

An *OpaqueAction* is translated as an `EntityServer` block with infinite capacity, see previous paragraph. The code corresponding to the node's behavior is given to the server's *`ServiceCompleteAction`* parameter. The OpaqueAction must have an input and an output pin, and their names are replaced by *`entity`* in the server's action. The name of the opaque action is given to the block.

The following code shows the content for an *OpaqueAction* in XML.

```
<Block BlockType="EntityServer" Name="vcaction" SID="26">
  <P Name="Ports">[1,1]</P>
  ...
  <P Name="InputPortMessageModes">m</P>
  <P Name="InputPortMap">u2</P>
  <P Name="OutputPortMessageModes">m</P>
  <P Name="OutputPortMap">o9</P>
  <P Name="Capacity">Inf</P>
  <P Name="ServiceTimeSource">MATLAB action</P>
  <P Name="ServiceCompleteAction">entity.v = entity.v - 1;</P>
  <P Name="ServiceTimeAction">dt = 0.0;</P>
</Block>
```

4.4.3.21 ObjectFlow

Object flows are translated as lines. These have a source and a target, each of which must be an outport from a block and an inport from a block, respectively. The following code shows the content for *ObjectFlow*s in XML.

```
<Line>
  ...
  <P Name="Src">11#out:1</P>
  <P Name="Dst">13#in:1</P>
</Line>
<Line>
  ...
  <P Name="Src">12#out:1</P>
  <P Name="Dst">10#in:1</P>
</Line>
```

4.4.4 AnyLogic translation

This subsection describes the translation of SysML activity models relying on the DEN library into a files that can be read by AnyLogic. These file are in XML. Note that not all the content required to create a valid file is shown, but only the parts that are translated from SysML.

4.4.4.1 Model

AnyLogic models are serialized in XML. This subsection shows how the translated content is represented in this format, referring to the XML elements and attributes found in the files, which might differ from the terms in AnyLogic.

In the hierarchy of XML elements, a model is part of a workspace, and it notably contains a set of "active object classes", which are equivalent to UML classes. The following code shows the root element of the SimEvent model file.

```
<AnyLogicWorkspace ...>
  <Model>
    <Id>2147483648</Id>
    <Name><![CDATA[CompleteModel]]></Name>
    <EngineVersion>6</EngineVersion>
    <JavaPackageName><![CDATA[CompleteModel]]></JavaPackageName>
    <ModelTimeUnit><![CDATA[Second]]></ModelTimeUnit>
    <ActiveObjectClasses>
     ...
    </ActiveObjectClasses>
    ...
  </Model>
</AnyLogicWorkspace>
```

4.4.4.2 Activities and blocks

User-defined SysML activities and blocks (including commodities) are generally translated into active object classes.

The DEN activities and their specializations correspond to active object classes from AnyLogic's ProcessModeling Library, so the content of these activities is not translated. These active object classes rely on template parameters to indicate the type of the objects flowing, and embedded objects typed by those classes indicate the actual template parameter substitutions occurring.

Active object classes have a identifier and a name. They can have additional Java code that is included by AnyLogic when simulating the model. This code is used in particular to add custom methods and attributes for commodities. Active object classes can have a set of connectors and embedded objects, which are equivalent to SysML properties.

The following code shows the active object class for an *Activity* in XML.

Relevant parts of the active object class definition are as follows:

```
<ActiveObjectClass>
  <Id>2147483649</Id>
  <Name><![CDATA[Complete]]></Name>
  <AdditionalClassCode>...</AdditionalClassCode>
  ...
  <Connectors>...</Connectors>
  ...
  <EmbeddedObjects>...</EmbeddedObjects>
  ...
</ActiveObjectClass>
```

4.4.4.3 Commodities

Commodities are also translated as active object classes, with constructors and properties given as `AdditionalCode`.

The following code shows the active object class for a commodity *A* that has a property *b* of type *B* and a property *c* of type *C*.

```
<ActiveObjectClass>
  <Id>2147483687</Id>
  <Name><![CDATA[A]]></Name>
  <AdditionalClassCode><![CDATA[public A(B b, C c)
{
this();
this.b = b;
this.c = c;
}
public B b = null;
public C c = null;]]></AdditionalClassCode>
  ...
</ActiveObjectClass>
```

4.4.4.4 Produce CallBehaviorAction

A *CallBehaviorAction* typed by *Produce* or one of its specializations is translated as an embedded object typed by the `Source` active object class, from the Process Modeling Library. The name of the call behavior action is given to the embedded object.

The behavior of the *preExit()* operation of the *Produce* is translated into code for the *`onAtExit`*, replacing the reference to the *CallBehaviorAction*'s output pin name by *`agent`*.

The interarrival time of the *Produce* is translated as follows:

1. if the *Produce* action has a *limit* of 0, the *`interarrivalTime`* parameter is given *`1`* as value and the *`maxArrivals`* parameter is given the *`0`* value;
2. if the *Produce* action has a *limit* of 1, the *`interarrivalTime`* parameter is given *`1`* as value, the *`firstArrivalMode`* parameter is given the expression *`self.AT_START`* as value, and the *`maxArrivals`* parameter is given the *`1`* value;

3. else

   (a) if the *interarrivalTime* property or corresponding slot is not stereotyped by a specialization of EventDistribution, the `interarrivalTime` parameter is given the expression `1/meantime` as value;

   (b) if the *interarrivalTime* property or corresponding slot is stereotyped by Beta with parameters p and q, the `interarrivalTime` parameter is given the expression `beta(p, q)` as value;

   (c) if the *interarrivalTime* property or corresponding slot is stereotyped by Erlang with parameters k and lambda, the `interarrivalTime` parameter is given the expression `erlang(lambda, k, min)` as value;

   (d) if the *interarrivalTime* property or corresponding slot is stereotyped by Exponential with parameter lambda, the `interarrivalTime` parameter is given the expression `exponential(lambda, min)` as value. If *inverse* is true, `1/lambda` is used instead of `lambda`;

   (e) if the *interarrivalTime* property or corresponding slot is stereotyped by Gamma with parameter alpha and beta, the `interarrivalTime` parameter is given the expression `gamma(alpha, beta, min)` as value;

   (f) if the *interarrivalTime* property or corresponding slot is stereotyped by LogNormal with parameters mu and sigma, the `interarrivalTime` parameter is given the expression `lognormal(mu, sigma)` as value;

   (g) if the *interarrivalTime* property or corresponding slot is stereotyped by Pareto with parameters alpha and min, the `interarrivalTime` parameter is given the expression `pareto(alpha, min)` as value;

   (h) if the *interarrivalTime* property or corresponding slot is stereotyped by Rayleigh with parameter sigma, the `interarrivalTime` parameter is given the expression `rayleigh(sigma, min)` as value;

   (i) if the *interarrivalTime* property or corresponding slot is stereotyped by Triangular with parameters min, max and mode, the `interarrivalTime` parameter is given the expression `triangular(min, max, mode)` as value;

   (j) if the *interarrivalTime* property or corresponding slot is stereotyped by Uniform with parameters min and max, the `interarrivalTime` parameter is given the expression `uniform(min, max)` as value;

   (k) if the *interarrivalTime* property or corresponding slot is stereotyped by Weibull with parameters alpha and beta, the `interarrivalTime` parameter is given the expression `weibull(alpha, beta, min)` as value.

Finally, the agent corresponding to the type of the *CallBehaviorAction*'s output pin is used as template parameter value for the source.

The embedded object also has the following parameters values: `pushProtocol` set to `false`, and `discardHangingEntities` set to `false`, to obtain the same behavior as activity pins. If the `pushProtocol` is set to `true` and the created object is not accepted by a subsequent block, the simulation throws an

exception. When *pushProtocol* is set to *false*, the created object either wait (*discardHangingEntities* is *false*) or is destroyed (*discardHangingEntities* is *true*).

The following code shows the content for a *Produce* action in XML.

```
<EmbeddedObject>
  <Id>2147483654</Id>
  <Name><![CDATA[bsource]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Source]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Source]]></ClassName>
      <ItemName><![CDATA[1412336242928]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[B]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <Parameters>
    <Parameter>
      <Name><![CDATA[arrivalType]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[self.INTERARRIVAL_TIME]]></Code>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[interarrivalTime]]></Name>
      <Value Class="CodeUnitValue">
        <Code><![CDATA[exponential(2.0 )]]></Code>
        <Unit Class="TimeUnits"><![CDATA[SECOND]]></Unit>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[newEntity]]></Name>
      <Value Class="EntityCodeValue">
        <IsAgentEntity>true</IsAgentEntity>
        <EntityEmbeddedObject>
          <ActiveObjectClass>
            <PackageName><![CDATA[CompleteModel]]></PackageName>
            <ClassName><![CDATA[B]]></ClassName>
          </ActiveObjectClass>
          <GenericParameterSubstitute>
            <GenericParameterSubstituteReference>
              <PackageName><![CDATA[CompleteModel]]></PackageName>
```

```
              <ClassName><![CDATA[B]]></ClassName>
              <ItemName><![CDATA[i2147483698]]></ItemName>
            </GenericParameterSubstituteReference>
          </GenericParameterSubstitute>
          <Parameters/>
          ...
        </EntityEmbeddedObject>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[pushProtocol]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[false]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[discardHangingEntities]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[false]]></Code>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[onAtExit]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[agent.v = 1.5;]]></Code>
      </Value>
    </Parameter>
    ...
  </Parameters>
  ...
</EmbeddedObject>
```

4.4.4.5 Queue CallBehaviorAction

A *CallBehaviorAction* typed by *Queue* or one of its specializations is translated as an embedded object typed by the `Queue` active object class, from the Process Modeling Library. The name of the call behavior action is given to the embedded object.

The capacity of the *Queue* activity is provided as the value of the *`capacity`* parameter if this capacity is not unlimited. If the capacity is unlimited, the *`maximumCapacity`* parameter is given the *`true`* value.

If the queue has a FIFO strategy, the *`queuing`* parameter is given the expression *`self.QUEUING_FIFO`* as value. If the queue has a LIFO strategy, the *`queuing`* parameter is given the the expression *`self.QUEUING_LIFO`* as value.

Finally, the agent corresponding to the type of the *CallBehaviorAction*'s input pin is used as template parameter value of the queue embedded object.

The behavior of the *postEntry()* operation of the *Queue* is translated into code for the *`onEnter`* action, replacing the reference to the *CallBehaviorAction*'s input pin name by *`agent`*.

The behavior of the *preExit()* operation of the *Queue* is translated into code for the `onAtExit` action, replacing the reference to the *CallBehaviorAction*'s output pin name by `agent`.

The following code shows the content for a *Queue* action in XML.

```
<EmbeddedObject>
  <Id>2147483657</Id>
  <Name><![CDATA[aqueue]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Queue]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Queue]]></ClassName>
      <ItemName><![CDATA[1412336242932]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[Commodity]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <Parameters>
    <Parameter>
      <Name><![CDATA[capacity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[50]]></Code>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[queuing]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[self.QUEUING_FIFO]]></Code>
      </Value>
    </Parameter>
    ...
  </Parameters>
  ...
</EmbeddedObject>
```

4.4.4.6 Seize CallBehaviorAction

A *CallBehaviorAction* typed by *Seize* or one of its specializations is translated as an embedded object typed by the `Seize` active object class, from the Process Modeling Library. The name of the call behavior action is given to the embedded object. The name of the resource pool of the *Seize* activity is provided as value of the `resourcePool` parameter of the embedded object.

The agent corresponding to the type of the *CallBehaviorAction*'s input pin is used as template pa-

rameter value of the seize embedded object.

The embedded object also has the following parameters values: `seizeFromOnePool` set to `true`, `resourceQuantiy` set to `1`, and `maximumCapacity` set to `true`.

The behavior of the *postEntry()* operation of the *Seize* is translated into code for the `onEnter` action, replacing the reference to the *CallBehaviorAction*'s input pin name by `agent`.

The behavior of the *preExit()* operation of the *Seize* is translated into code for the `onSeizeUnit` action, replacing the reference to the *CallBehaviorAction*'s output pin name by `agent`.

The following code shows the content for a *Seize* action in XML.

```
<EmbeddedObject>
  <Id>2147483664</Id>
  <Name><![CDATA[bseize]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Seize]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Seize]]></ClassName>
      <ItemName><![CDATA[1412336243147]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[Commodity]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <Parameters>
    <Parameter>
      <Name><![CDATA[seizeFromOnePool]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[true]]></Code>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[resourcePool]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[rp]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[resourceQuantity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[1]]></Code>
      </Value>
    </Parameter>
```

```
    ...
    <Parameter>
      <Name><![CDATA[maximumCapacity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[true]]></Code>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[pushProtocol]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[false]]></Code>
      </Value>
    </Parameter>
    ...
  </Parameters>
  ...
</EmbeddedObject>
```

4.4.4.7 Delay CallBehaviorAction

A *CallBehaviorAction* typed by *Delay* or one of its specializations is translated as an embedded object typed by the `Delay` active object class, from the Process Modeling Library. The name of the call behavior action is given to the embedded object.

The delay time distribution of the *Delay* is translated like the inter-generation time of the *Produce* activity, see Subsection 4.4.4.4.

The behavior of the *postEntry()* operation of the *Delay* is translated into code for the *`onEnter`* action, replacing the reference to the *CallBehaviorAction*'s input pin name by *`agent`*.

The behavior of the *preExit()* operation of the *Delay* is translated into code for the *`onAtExit`* action, replacing the reference to the *CallBehaviorAction*'s output pin name by *`agent`*.

Finally, the agent corresponding to the type of the *CallBehaviorAction*'s input pin is used as template parameter value of the delay embedded object.

The embedded object also has the following parameters values: *`maximumCapacity`* set to *`true`*, and *`pushProtocol`* set to *`false`*.

The following code shows the content for a *Delay* action in XML.

```
<EmbeddedObject>
  <Id>2147483665</Id>
  <Name><![CDATA[bdelay]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Delay]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
```

```
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Delay]]></ClassName>
      <ItemName><![CDATA[1412336242930]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[B]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <Parameters>
    ...
    <Parameter>
      <Name><![CDATA[delayTime]]></Name>
      <Value Class="CodeUnitValue">
        <Code><![CDATA[exponential( 1/1.2 )]]></Code>
        <Unit Class="TimeUnits"><![CDATA[SECOND]]></Unit>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[capacity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[1]]></Code>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[pushProtocol]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[false]]></Code>
      </Value>
    </Parameter>
    ...
  </Parameters>
  ...
</EmbeddedObject>
```

4.4.4.8 Release CallBehaviorAction

A *CallBehaviorAction* typed by *Release* or one of its specializations is translated as an embedded object typed by the `Release` active object class, from the Process Modeling Library. The name of the call behavior action is given to the embedded object. The name of the resource pool of the *Release* activity is provided as value of the `resourcePool` parameter of the embedded object.

The behavior of the *postEntry()* operation of the *Release* is translated into code for the `onEnter` action, replacing the reference to the *CallBehaviorAction*'s input pin name by `agent`.

The behavior of the *preExit()* operation of the *Release* is translated into code for the `onRelease` action, replacing the reference to the *CallBehaviorAction*'s output pin name by `agent`.

Finally, the agent corresponding to the type of the *CallBehaviorAction*'s input pin is used as template parameter value of the release embedded object.

The following code shows the content for a *Release* action in XML.

```
<EmbeddedObject>
  <Id>2147483666</Id>
  <Name><![CDATA[brelease]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Release]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Release]]></ClassName>
      <ItemName><![CDATA[1412336243154]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[Commodity]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <Parameters>
    <Parameter>
      <Name><![CDATA[releaseMode]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[self.SPECIFIED_QUANTITY]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[resourcePool]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[rp]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[quantity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[1]]></Code>
      </Value>
    </Parameter>
    ...
  </Parameters>
  ...
</EmbeddedObject>
```

4.4.4.9 Consume CallBehaviorAction

A *CallBehaviorAction* typed by *Consume* or one of its specializations is translated as an embedded object typed by the `Sink` active object class, from the Process Modeling Library. The name of the call behavior action is given to the embedded object.

The behavior of the *postEntry()* operation of the *Queue* is translated into code for the `onEnter` action,

replacing the reference to the *CallBehaviorAction*'s input pin name by `agent`.

Finally, the agent corresponding to the type of the *CallBehaviorAction*'s input pin is used as template parameter value of the sink embedded object.

The following code shows the content for a *Consume* action in XML.

```
<EmbeddedObject>
  <Id>2147483675</Id>
  <Name><![CDATA[sink]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Sink]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Sink]]></ClassName>
      <ItemName><![CDATA[1412336242929]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[Commodity]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  ...
</EmbeddedObject>
```

4.4.4.10 Serve CallBehaviorAction

A *CallBehaviorAction* typed by *Serve* or one of its specializations and with an unlimited value of *numberOfServers* is translated as a *Delay*, see Subsection 4.4.4.7.

A *CallBehaviorAction* typed by *Serve* or one of its specializations is translated as a pattern made of:

1. an embedded object typed by the `Seize` active object class, from the Process Modeling Library,
2. an embedded object typed by the `Delay` active object class, from the Process Modeling Library,
3. an embedded object typed by the `Release` active object class, from the Process Modeling Library,
4. a connector between the `Seize` embedded object and the `Delay` embedded object
5. a connector between the `Delay` embedded object and the `Release` embedded object

The name of the call behavior action is given to the three embedded objects, with a unique suffixed such as `_seize`, `_delay`, and `_release`.

The service time distribution of the *Serve* is translated like the inter-generation time of the *Produce* activity, see Subsection 4.4.4.4.

If the *Serve* activity has a resource pool, the name of that resource pool is provided as value of the `resourcePool` parameter of the `Seize` and `Release` object. Also in the `Seize` object, the `seizeFromOnePool` parameter is set to `true` and the `resourceQuantity` parameter is set to `1`.

The behavior of the *postEntry()* operation of the *Serve* is translated into code for the `onEnter` action of the `Seize` object, replacing the reference to the *CallBehaviorAction*'s input pin name by `agent`.

The behavior of the *preExit()* operation of the *Serve* is translated into code for the `onRelease` action of the `Release` object, replacing the reference to the *CallBehaviorAction*'s output pin name by `agent`.

Finally, the agent corresponding to the type of the *CallBehaviorAction*'s input pin is used as template parameter value of all the embedded objects.

The following code shows the content for a *Serve* action in XML.

```
<EmbeddedObject>
  <Id>2147483658</Id>
  <Name><![CDATA[aserver_seize]]></Name>
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Seize]]></ClassName>
  </ActiveObjectClass>
  <Parameters>
    <Parameter>
      <Name><![CDATA[seizeFromOnePool]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[true]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[resourcePool]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[aserver_rp]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[resourceQuantity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[1]]></Code>
      </Value>
    </Parameter>
  </Parameters>
</EmbeddedObject>

<EmbeddedObject>
  <Id>2147483659</Id>
  <Name><![CDATA[aserver_delay]]></Name>
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Delay]]></ClassName>
```

```
  </ActiveObjectClass>
  <Parameters>
    <Parameter>
      <Name><![CDATA[delayTime]]></Name>
      <Value Class="CodeUnitValue">
        <Code><![CDATA[exponential( 1/1.5 )]]></Code>
        <Unit Class="TimeUnits"><![CDATA[SECOND]]></Unit>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[maximumCapacity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[true]]></Code>
      </Value>
    </Parameter>
  </Parameters>
</EmbeddedObject>

<EmbeddedObject>
  <Id>2147483660</Id>
  <Name><![CDATA[aserver_release]]></Name>
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Release]]></ClassName>
  </ActiveObjectClass>
  <Parameters>
    <Parameter>
      <Name><![CDATA[releaseMode]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[self.SPECIFIED_QUANTITY]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[resourcePool]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[aserver_rp]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[quantity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[1]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[onReleaseUnit]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[agent.b.v = agent.b.v+2;
agent.c.v = agent.c.v+3;]]></Code>
      </Value>
    </Parameter>
  </Parameters>
</EmbeddedObject>
```

```
<EmbeddedObject>
  <Id>2147483686</Id>
  <Name><![CDATA[aserver_rp]]></Name>
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[ResourcePool]]></ClassName>
  </ActiveObjectClass>
  <Parameters>
    <Parameter>
      <Name><![CDATA[capacity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[2]]></Code>
      </Value>
    </Parameter>
  </Parameters>
</EmbeddedObject>
...
<Connector>
  <Id>2147483661</Id>
  <Name><![CDATA[aserveraserver_seize_to_aserver_delay]]></Name>
  <SourceEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[aserver_seize]]></ItemName>
  </SourceEmbeddedObjectReference>
  <SourceConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Seize]]></ClassName>
    <ItemName><![CDATA[out]]></ItemName>
  </SourceConnectableItemReference>
  <TargetEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[aserver_delay]]></ItemName>
  </TargetEmbeddedObjectReference>
  <TargetConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Delay]]></ClassName>
    <ItemName><![CDATA[in]]></ItemName>
  </TargetConnectableItemReference>
</Connector>

<Connector>
  <Id>2147483662</Id>
  <Name><![CDATA[aserveraserver_delay_to_aserver_release]]></Name>
  <SourceEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[aserver_delay]]></ItemName>
  </SourceEmbeddedObjectReference>
  <SourceConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
```

```
    <ClassName><![CDATA[Delay]]></ClassName>
    <ItemName><![CDATA[out]]></ItemName>
  </SourceConnectableItemReference>
  <TargetEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[aserver_release]]></ItemName>
  </TargetEmbeddedObjectReference>
  <TargetConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Release]]></ClassName>
    <ItemName><![CDATA[in]]></ItemName>
  </TargetConnectableItemReference>
</Connector>
```

4.4.4.11 Process CallBehaviorAction

A *CallBehaviorAction* typed by *Process* or one of its specializations is translated as a pattern made of:

1. an embedded object typed by the `Queue` active object class, from the Process Modeling Library,
2. an embedded object typed by the `Seize` active object class, from the Process Modeling Library,
3. an embedded object typed by the `Delay` active object class, from the Process Modeling Library,
4. an embedded object typed by the `Release` active object class, from the Process Modeling Library,
5. a connector between the `Queue` embedded object and the `Seize` embedded object
6. a connector between the `Seize` embedded object and the `Delay` embedded object
7. a connector between the `Delay` embedded object and the `Release` embedded object

The name of the call behavior action is given to the embedded objects along with a suffix to distinguish them (such as `_queue`, `_seize`, `_delay`, `_release`).

The processing time distribution of the *Process* action is translated like the inter-generation time of the *Produce* activity, see Subsection 4.4.4.4.

If the *Process* activity has a resource pool, the name of that resource pool is provided as value of the *`resourcePool`* parameter of the `Seize` and `Release` object. Also in this object, the parameter *`seizeFromOnePool`* is set to *`true`* and the parameter *`resourceQuantity`* is set to *`1`*.

The capacity of the *Process* activity is provided as the value of the *`capacity`* parameter of the `Queue` object if this capacity is not unlimited. If the capacity is unlimited, the *`maximumCapacity`* parameter is given the *`true`* value.

If the queue has a FIFO strategy, the *queuing* parameter is given the *self.QUEUING_FIFO* value. If the queue has a LIFO strategy, that *queuing* parameter is given the *self.QUEUING_LIFO* value. If the queue has a PRIORITY strategy, the *queuing* parameter is given the *self.QUEUING_PRIORITY* value, and the *getPriority()* expression is given as value of the *priority* parameter.

The behavior of the *postEntry()* operation of the *Process* is translated into code for the *onEnter* action of the Queue object, replacing the reference to the *CallBehaviorAction*'s input pin name by *agent*.

The behavior of the *preExit()* operation of the *Process* is translated into code for the *onRelease* action of the Release object, replacing the reference to the *CallBehaviorAction*'s output pin name by *agent*.

Finally, the agent corresponding to the type of the *CallBehaviorAction*'s input pin is used as template parameter value of the two embedded objects.

The following code shows the content for a *Process* in XML.

```
<EmbeddedObject>
  <Id>2147483671</Id>
  <Name><![CDATA[cprocessing_queue]]></Name>
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Queue]]></ClassName>
  </ActiveObjectClass>
  <Parameters>
    <Parameter>
      <Name><![CDATA[capacity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[0]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[maximumCapacity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[true]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[queuing]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[self.QUEUING_FIFO]]></Code>
      </Value>
    </Parameter>
  </Parameters>
</EmbeddedObject>

<EmbeddedObject>
  <Id>2147483672</Id>
  <Name><![CDATA[cprocessing_seize]]></Name>
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
```

```
    <ClassName><![CDATA[Seize]]></ClassName>
  </ActiveObjectClass>
  <Parameters>
    <Parameter>
      <Name><![CDATA[seizeFromOnePool]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[true]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[resourcePool]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[rp]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[resourceQuantity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[1]]></Code>
      </Value>
    </Parameter>
  </Parameters>
</EmbeddedObject>

<EmbeddedObject>
  <Id>2147483673</Id>
  <Name><![CDATA[cprocessing_delay]]></Name>
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Delay]]></ClassName>
  </ActiveObjectClass>
  <Parameters>
    <Parameter>
      <Name><![CDATA[delayTime]]></Name>
      <Value Class="CodeUnitValue">
        <Code><![CDATA[1.25]]></Code>
        <Unit Class="TimeUnits"><![CDATA[SECOND]]></Unit>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[maximumCapacity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[true]]></Code>
      </Value>
    </Parameter>
  </Parameters>
</EmbeddedObject>

<EmbeddedObject>
  <Id>2147483674</Id>
  <Name><![CDATA[cprocessing_release]]></Name>
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
```

```
    <ClassName><![CDATA[Release]]></ClassName>
  </ActiveObjectClass>
  <Parameters>
    <Parameter>
      <Name><![CDATA[releaseMode]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[self.SPECIFIED_QUANTITY]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[resourcePool]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[rp]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[quantity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[1]]></Code>
      </Value>
    </Parameter>
  </Parameters>
</EmbeddedObject>

<EmbeddedObject>
  <Id>2147483687</Id>
  <Name><![CDATA[rp]]></Name>
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[ResourcePool]]></ClassName>
  </ActiveObjectClass>
  <Parameters>
    <Parameter>
      <Name><![CDATA[capacity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[2]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[onReleaseUnit]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[agent.v = agent.v*3;]]></Code>
      </Value>
    </Parameter>
  </Parameters>
</EmbeddedObject>
...
<Connector>
  <Id>2147483675</Id>
  <Name><![CDATA[cprocessingcprocessing_queue_to_cprocessing_seize]]></Name>
  <SourceEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
```

```
    <ItemName><![CDATA[cprocessing_queue]]></ItemName>
  </SourceEmbeddedObjectReference>
  <SourceConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Queue]]></ClassName>
    <ItemName><![CDATA[out]]></ItemName>
  </SourceConnectableItemReference>
  <TargetEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[cprocessing_seize]]></ItemName>
  </TargetEmbeddedObjectReference>
  <TargetConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Seize]]></ClassName>
    <ItemName><![CDATA[in]]></ItemName>
  </TargetConnectableItemReference>
</Connector>

<Connector>
  <Id>2147483676</Id>
  <Name><![CDATA[cprocessingcprocessing_seize_to_cprocessing_delay]]></Name>
  <SourceEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[cprocessing_seize]]></ItemName>
  </SourceEmbeddedObjectReference>
  <SourceConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Seize]]></ClassName>
    <ItemName><![CDATA[out]]></ItemName>
  </SourceConnectableItemReference>
  <TargetEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[cprocessing_delay]]></ItemName>
  </TargetEmbeddedObjectReference>
  <TargetConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Delay]]></ClassName>
    <ItemName><![CDATA[in]]></ItemName>
  </TargetConnectableItemReference>
</Connector>

<Connector>
  <Id>2147483677</Id>
  <Name><![CDATA[cprocessingcprocessing_delay_to_cprocessing_release]]></Name>
  <SourceEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[cprocessing_delay]]></ItemName>
  </SourceEmbeddedObjectReference>
  <SourceConnectableItemReference>
```

```
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Delay]]></ClassName>
    <ItemName><![CDATA[out]]></ItemName>
  </SourceConnectableItemReference>
  <TargetEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[cprocessing_release]]></ItemName>
  </TargetEmbeddedObjectReference>
  <TargetConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Release]]></ClassName>
    <ItemName><![CDATA[in]]></ItemName>
  </TargetConnectableItemReference>
</Connector>
```

4.4.4.12 Replicate CallBehaviorAction

A *CallBehaviorAction* typed by *Replicate* or one of its specializations is translated as an embedded object typed by the `Split` active object class, from the Process Modeling Library. The name of the call behavior action is given to the embedded object.

Java code is created to manage the duplication of the input object, as follows:

1. in the owning active object class, a method is added to create an object of the input type. It creates a new object typed by the agent corresponding to the input pin type, assigns as value for the new object the attribute values from the input object, and returns the newly created object.
2. in the Split object, the *`newEntity`* parameter is given as value a call of the previously created method, with as method parameter the input value.

The behavior of the *postEntry()* operation of the *Replicate* is translated into code for the *`onAtEnter`* action of the `Split` object, replacing the reference to the *CallBehaviorAction*'s input pin name by *`agent`*.

Finally, the agent corresponding to the type of the *CallBehaviorAction*'s input pin is used as template parameter value of the Split object.

The following code shows the content for a *Replicate* in XML.

```
    <ActiveObjectClasses>
      <ActiveObjectClass>
        <Id>2147483649</Id>
        <Name><![CDATA[Complete]]></Name>
        <AdditionalClassCode><![CDATA
        ...
public static Commodity breplicate_duplicate(Commodity agent)
{Commodity _ret = new Commodity();
return _ret;
}
```

```
]]></AdditionalClassCode>
...
<EmbeddedObject>
  <Id>2147483671</Id>
  <Name><![CDATA[breplicate]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Split]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Split]]></ClassName>
      <ItemName><![CDATA[1412336242938]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[Commodity]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Split]]></ClassName>
      <ItemName><![CDATA[1412336242939]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[Commodity]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <Parameters>
    ...
    <Parameter>
      <Name><![CDATA[newEntity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[breplicate_duplicate(original)]]></Code>
      </Value>
    </Parameter>
    ...
  </Parameters>
  ...
</EmbeddedObject>
</EmbeddedObject>
```

4.4.4.13 Combine CallBehaviorAction

A *CallBehaviorAction* typed by *Combine* or one of its specializations is translated as an embedded object typed by the `Assembler` active object class, from the Process Modeling Library. The name of the call behavior action is given to the embedded object.

A *Combine* action may have up to 5 input values that will be given as attribute values of the output value.

The agent corresponding to the type of the output pin is given as template parameter of the object, as well as up to 5 agents, corresponding to the types of each input pin.

The embedded object has 5 parameters named `quantity1`, `quantity2`, `quantity3`, `quantity4`, and `quantity5`, which correspond to the expected quantities for each input. The value for these parameters is set to `1` if the *CallBehaviorAction* has corresponding input pins, or `0` otherwise.

Java code is created to manage the creation of the combined object, as follows:

1. in the owning active object class, a method is added to create an object of the combined type from the given input objects. It creates a new object typed by the agent corresponding to the output pin type, assigns as value for the attributes the objects from each input pins, and returns the newly created object.
2. in the Assembler object, the `newEntity` parameter is given as value a call of the previously created method, with as method parameter the values from each input.

The behavior of the *preExit()* operation of the *Combine* is translated into code for the `onAtExit` action of the `Assembler` object, replacing the reference to the *CallBehaviorAction*'s output pin name by `agent`.

The following code shows the content for a *Combine* in XML.

```
<EmbeddedObject>
  <Id>2147483656</Id>
  <Name><![CDATA[acombiner]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Assembler]]></ClassName>
      <ItemName><![CDATA[1412336243190]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[A]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Assembler]]></ClassName>
```

```
    <ItemName><![CDATA[1412336243191]]></ItemName>
  </GenericParameterSubstituteReference>
  <GenericParameterSubstituteValue Class="CodeValue">
    <Code><![CDATA[Commodity]]></Code>
  </GenericParameterSubstituteValue>
</GenericParameterSubstitute>
<GenericParameterSubstitute>
  <GenericParameterSubstituteReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243192]]></ItemName>
  </GenericParameterSubstituteReference>
  <GenericParameterSubstituteValue Class="CodeValue">
    <Code><![CDATA[Commodity]]></Code>
  </GenericParameterSubstituteValue>
</GenericParameterSubstitute>
<GenericParameterSubstitute>
  <GenericParameterSubstituteReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243193]]></ItemName>
  </GenericParameterSubstituteReference>
</GenericParameterSubstitute>
<GenericParameterSubstitute>
  <GenericParameterSubstituteReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243194]]></ItemName>
  </GenericParameterSubstituteReference>
</GenericParameterSubstitute>
<GenericParameterSubstitute>
  <GenericParameterSubstituteReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243195]]></ItemName>
  </GenericParameterSubstituteReference>
</GenericParameterSubstitute>
<Parameters>
  <Parameter>
    <Name><![CDATA[quantity1]]></Name>
    <Value Class="CodeValue">
      <Code><![CDATA[1]]></Code>
    </Value>
  </Parameter>
  <Parameter>
    <Name><![CDATA[quantity2]]></Name>
    <Value Class="CodeValue">
      <Code><![CDATA[1]]></Code>
    </Value>
```

```
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[newEntity]]></Name>
      <Value Class="CodeValue">
        <Code>
          <![CDATA[acombiner_assembly(queue1.removeFirst(), queue2.removeFirst())]]>
        </Code>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[delayTime]]></Name>
      <Value Class="CodeUnitValue">
        <Code><![CDATA[0]]></Code>
        <Unit Class="TimeUnits"><![CDATA[SECOND]]></Unit>
      </Value>
    </Parameter>
    ...
  </Parameters>
  ...
</EmbeddedObject>
```

4.4.4.14 Split CallBehaviorAction

A *CallBehaviorAction* typed by *Split* or one of its specializations may have up to 5 output values that correspond to attribute values of the input value. Assuming it has *n* output values, it is translated as a pattern made of:

1. if $n > 1$: $(n-1)$ `Split` objects, $(n-1)$ `Assembler` objects, and $(n-1)$ connectors,
2. 1 `Assembler` object and one connector.

The idea of the pattern is to duplicate the input value $n-1$ times, and use *n* `Assembler` objects to extract the *n* attributes from this object.

Each object of the pattern is given the name of the call behavior action plus a unique suffix.

The split objects are created as in the `Replicate` *CallBehaviorAction* paragraph.

The Assembler objects have a *`newEntity`* code that returns the corresponding attribute value from the input object. The template parameter for the assembler object are the agent corresponding to the output pin type, and the agent corresponding to the attribute value type being extracted. In all Assembler objects, *`quantity1`* is set to *`1`*, and the other quantities are set to *`0`*.

The behavior of the *postEntry()* operation of the *Split* is translated into code for the *`onEnter1`* action of the `Assembler` object, replacing the reference to the *CallBehaviorAction*'s input pin name by *`agent`*.

The following code shows the content for a *Split* action in XML.

```
    <ActiveObjectClasses>
      <ActiveObjectClass>
        <Id>2147483649</Id>
        <Name><![CDATA[Complete]]></Name>
        <AdditionalClassCode><![CDATA[
public static A asplitter_duplicate(A agent)
{A _ret = new A();
_ret.b=agent.b;
_ret.c=agent.c;
return _ret;
}
]]></AdditionalClassCode>
...
<EmbeddedObject>
  <Id>2147483659</Id>
  <Name><![CDATA[asplitter]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Split]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Split]]></ClassName>
      <ItemName><![CDATA[1412336242938]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[A]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Split]]></ClassName>
      <ItemName><![CDATA[1412336242939]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[A]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <Parameters>
    ...
    <Parameter>
      <Name><![CDATA[newEntity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[asplitter_duplicate(original)]]></Code>
      </Value>
    </Parameter>
    ...
```

```
  </Parameters>
  ...
</EmbeddedObject>
<EmbeddedObject>
  <Id>2147483660</Id>
  <Name><![CDATA[asplitter_1]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Assembler]]></ClassName>
      <ItemName><![CDATA[1412336243190]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[B]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Assembler]]></ClassName>
      <ItemName><![CDATA[1412336243191]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[A]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Assembler]]></ClassName>
      <ItemName><![CDATA[1412336243192]]></ItemName>
    </GenericParameterSubstituteReference>
  </GenericParameterSubstitute>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Assembler]]></ClassName>
      <ItemName><![CDATA[1412336243193]]></ItemName>
    </GenericParameterSubstituteReference>
  </GenericParameterSubstitute>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
```

```
      </PackageName>
      <ClassName><![CDATA[Assembler]]></ClassName>
      <ItemName><![CDATA[1412336243194]]></ItemName>
    </GenericParameterSubstituteReference>
  </GenericParameterSubstitute>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Assembler]]></ClassName>
      <ItemName><![CDATA[1412336243195]]></ItemName>
    </GenericParameterSubstituteReference>
  </GenericParameterSubstitute>
  <Parameters>
    <Parameter>
      <Name><![CDATA[quantity1]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[1]]></Code>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[newEntity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[self.queue1.removeFirst().b]]></Code>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[delayTime]]></Name>
      <Value Class="CodeUnitValue">
        <Code><![CDATA[0]]></Code>
        <Unit Class="TimeUnits"><![CDATA[SECOND]]></Unit>
      </Value>
    </Parameter>
    ...
  </Parameters>
  ...
</EmbeddedObject>
<EmbeddedObject>
  <Id>2147483662</Id>
  <Name><![CDATA[asplitter_2]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Assembler]]></ClassName>
```

```
    <ItemName><![CDATA[1412336243190]]></ItemName>
  </GenericParameterSubstituteReference>
  <GenericParameterSubstituteValue Class="CodeValue">
    <Code><![CDATA[C]]></Code>
  </GenericParameterSubstituteValue>
</GenericParameterSubstitute>
<GenericParameterSubstitute>
  <GenericParameterSubstituteReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243191]]></ItemName>
  </GenericParameterSubstituteReference>
  <GenericParameterSubstituteValue Class="CodeValue">
    <Code><![CDATA[A]]></Code>
  </GenericParameterSubstituteValue>
</GenericParameterSubstitute>
<GenericParameterSubstitute>
  <GenericParameterSubstituteReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243192]]></ItemName>
  </GenericParameterSubstituteReference>
</GenericParameterSubstitute>
<GenericParameterSubstitute>
  <GenericParameterSubstituteReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243193]]></ItemName>
  </GenericParameterSubstituteReference>
</GenericParameterSubstitute>
<GenericParameterSubstitute>
  <GenericParameterSubstituteReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243194]]></ItemName>
  </GenericParameterSubstituteReference>
</GenericParameterSubstitute>
<GenericParameterSubstitute>
  <GenericParameterSubstituteReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243195]]></ItemName>
  </GenericParameterSubstituteReference>
</GenericParameterSubstitute>
<Parameters>
  <Parameter>
    <Name><![CDATA[quantity1]]></Name>
    <Value Class="CodeValue">
```

```
        <Code><![CDATA[1]]></Code>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[newEntity]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[self.queue1.removeFirst().c]]></Code>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[delayTime]]></Name>
      <Value Class="CodeUnitValue">
        <Code><![CDATA[0]]></Code>
        <Unit Class="TimeUnits"><![CDATA[SECOND]]></Unit>
      </Value>
    </Parameter>
    ...
  </Parameters>
  ...
</EmbeddedObject>
...
<Connector>
  <Id>2147483661</Id>
  <Name><![CDATA[asplitter_to_asplitter_1]]></Name>
  ...
  <SourceEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[asplitter]]></ItemName>
  </SourceEmbeddedObjectReference>
  <SourceConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Split]]></ClassName>
    <ItemName><![CDATA[out]]></ItemName>
  </SourceConnectableItemReference>
  <TargetEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[asplitter_1]]></ItemName>
  </TargetEmbeddedObjectReference>
  <TargetConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[in1]]></ItemName>
  </TargetConnectableItemReference>
  ...
</Connector>
<Connector>
  <Id>2147483663</Id>
```

```
  <Name><![CDATA[asplitter_to_asplitter_2]]></Name>
  ...
  <SourceEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[asplitter]]></ItemName>
  </SourceEmbeddedObjectReference>
  <SourceConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Split]]></ClassName>
    <ItemName><![CDATA[outCopy]]></ItemName>
  </SourceConnectableItemReference>
  <TargetEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[asplitter_2]]></ItemName>
  </TargetEmbeddedObjectReference>
  <TargetConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[in1]]></ItemName>
  </TargetConnectableItemReference>
  ...
</Connector>
```

4.4.4.15 Transform CallBehaviorAction

A *CallBehaviorAction* typed by *Transform* or one of its specializations is translated as an embedded object typed by the `Assembler` active object class, from the Process Modeling Library. The name of the call behavior action is given to the embedded object.

Since the `Assembler` object has only one input, it is given parameter value of `1` for `quantity1`, and parameter values of `0` for `quantity2`, `quantity3`, `quantity4`, and `quantity5`.

Java code is created to manage the creation of the output object, as follows:

1. in the owning active object class, a method is added to create an object of the output type from the given input object. It creates a new object typed by the agent corresponding to the output pin type, uses the code from the *Transform*'s *preExit()* operation, and returns the newly created object.

2. in the `Assembler` object, the `newEntity` parameter is given as value a call of the previously created method, with as method parameter the input object.

The behavior of the *postEntry()* operation of the *Transform* is translated into code for the `onEnter1` action of the `Assembler` object, replacing the reference to the *CallBehaviorAction*'s input pin name by `agent`.

The behavior of the *preExit()* operation of the *Transform* is translated into code for the `onAtExit` action of the `Assembler` object, replacing the reference to the *CallBehaviorAction*'s output pin name by `agent`.

The agent corresponding to the type of the output pin and the agent corresponding to the type of the input pin are given as template parameter of the object.

The following code shows the content for a *Transform* action in XML.

```
<ActiveObjectClass>
  <Id>2147483649</Id>
  <Name><![CDATA[Complete]]></Name>
  <AdditionalClassCode><![CDATA[
public static B c_to_b_assembly(C cIn)
{B bOut = new B();
bOut = cIn+1;return bOut;}
]]></AdditionalClassCode>
...
<EmbeddedObject>
  <Id>2147483670</Id>
  <Name><![CDATA[c_to_b]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Assembler]]></ClassName>
      <ItemName><![CDATA[1412336243190]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[B]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[Assembler]]></ClassName>
      <ItemName><![CDATA[1412336243191]]></ItemName>
    </GenericParameterSubstituteReference>
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[C]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
```

```
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243192]]></ItemName>
  </GenericParameterSubstituteReference>
</GenericParameterSubstitute>
<GenericParameterSubstitute>
  <GenericParameterSubstituteReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243193]]></ItemName>
  </GenericParameterSubstituteReference>
</GenericParameterSubstitute>
<GenericParameterSubstitute>
  <GenericParameterSubstituteReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243194]]></ItemName>
  </GenericParameterSubstituteReference>
</GenericParameterSubstitute>
<GenericParameterSubstitute>
  <GenericParameterSubstituteReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[1412336243195]]></ItemName>
  </GenericParameterSubstituteReference>
</GenericParameterSubstitute>
<Parameters>
  <Parameter>
    <Name><![CDATA[quantity1]]></Name>
    <Value Class="CodeValue">
      <Code><![CDATA[1]]></Code>
    </Value>
  </Parameter>
  ...
  <Parameter>
    <Name><![CDATA[newEntity]]></Name>
    <Value Class="CodeValue">
      <Code><![CDATA[c_to_b_assembly(queue1.removeFirst())]]></Code>
    </Value>
  </Parameter>
  ...
  <Parameter>
    <Name><![CDATA[delayTime]]></Name>
    <Value Class="CodeUnitValue">
      <Code><![CDATA[0]]></Code>
      <Unit Class="TimeUnits"><![CDATA[SECOND]]></Unit>
    </Value>
  </Parameter>
  ...
</Parameters>
...
```

```
</EmbeddedObject>
```

4.4.4.16 Other CallBehaviorAction

Other *CallBehaviorAction*s are translated as embedded objects typed by the agent corresponding to the sysmlCallBehaviorAction's behavior.

The following code shows the content for other *CallBehaviorAction*s in XML.

```
<EmbeddedObject>
  <Id>2147483676</Id>
  <Name><![CDATA[Complete_callbehavioraction42]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[CompositeActivity]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[CompleteModel]]></PackageName>
      <ClassName><![CDATA[CompositeActivity]]></ClassName>
      <ItemName><![CDATA[i2147483680]]></ItemName>
    </GenericParameterSubstituteReference>
  </GenericParameterSubstitute>
  <Parameters/>
  ...
</EmbeddedObject>
```

4.4.4.17 ActivityParameterNode and Parameter

An activity parameter is translated as a `Port` object, owned by the active object class corresponding to the activity. In the activity, an *ActivityParameterNode* corresponds to the `Port` object related to its parameter (effectively equivalent to merging an activity parameter node and its parameter).

The following code shows the content for a *Parameter* in XML.

```
<Port>
  <Id>2147483685</Id>
  <Name><![CDATA[inVC]]></Name>
  ...
</Port>
```

4.4.4.18 MergeNode

AnyLogic does not have or need an equivalent of merge nodes, as these are equivalent to ports targeted by more than one connectors.

To simplify the software implementation and avoid redirection, a `Delay` object can serve as a placeholder for a *MergeNode*, with a distribution of `0`, indicating that anything going through it is not being delayed.

The template parameter value of the Delay object is the agent corresponding to the type of the first output pin encountered when navigating backward through the node's incoming edges.

See delay node, Subsection 4.4.4.7.

4.4.4.19 DecisionNode

A *DecisionNode* is translated as an embedded object typed by the `SelectOutput5` active object class, from the Process Modeling Library, which provides 5 outputs. The output is selected based on three criteria:

- *Conditions* are for when the outgoing edges from the *DecisionNode* have a guard with an *OpaqueExpression*. To use conditions, the *type* parameter must be given the *self.TYPE_CONDITIONS* value. The corresponding expression is used as value of the *condition1*, ..., *condition5* parameters of the embedded object, based on the position of the edge in the node's outgoings edges.
- *Probabilities* are for when the outgoing edges from the *DecisionNode* have a guard with a *LiteralReal*. To use probabilities, the *type* parameter must be given the *self.TYPE_PROBABILITIES* value. The probability corresponding to an edge is used as value of the corresponding parameter *probability1*, ..., *probability5* of the embedded object, based on the position of the edge in the node's outgoings edges.
- *Port selection code* is for when the decision node has a *decisionInput* behavior. To use port selection code, the *type* parameter must be given the *self.TYPE_EXIT_NUMBER* value. The code corresponding to that behavior is the value of the *exitNumber* parameter.

The template parameter value of the `SelectOutput5` object is the agent corresponding to the type of the first output pin encountered when navigating backward through the node's incoming edges.

The following code shows the content for a *Decision* node in XML.

```
<EmbeddedObject>
  <Id>2147483673</Id>
  <Name><![CDATA[Complete_decisionnode39]]></Name>
  ...
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[SelectOutput5]]></ClassName>
  </ActiveObjectClass>
  <GenericParameterSubstitute>
    <GenericParameterSubstituteReference>
      <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
      </PackageName>
      <ClassName><![CDATA[SelectOutput5]]></ClassName>
      <ItemName><![CDATA[1412336243132]]></ItemName>
    </GenericParameterSubstituteReference>
```

```
    <GenericParameterSubstituteValue Class="CodeValue">
      <Code><![CDATA[ValuedCommodity]]></Code>
    </GenericParameterSubstituteValue>
  </GenericParameterSubstitute>
  <Parameters>
    <Parameter>
      <Name><![CDATA[type]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[self.TYPE_PROBABILITIES]]></Code>
      </Value>
    </Parameter>
    ...
    <Parameter>
      <Name><![CDATA[probability1]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[0.5]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[probability2]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[0.5]]></Code>
      </Value>
    </Parameter>
    ...
</EmbeddedObject>
```

4.4.4.20 Opaque Action

An *OpaqueAction* is translated as a `Delay` object with a *`delayTime`* of *`0`*, and with the code corresponding to the node's *`preExit()`* operation given to the *`onAtExit`* parameter of the object. The name of the opaque action is given to the embedded object.

The following code shows the content for an *OpaqueAction* in XML.

```
<EmbeddedObject>
  <Id>2147483693</Id>
  <Name><![CDATA[vcaction]]></Name>
  <ActiveObjectClass>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
    <ClassName><![CDATA[Delay]]></ClassName>
  </ActiveObjectClass>
  <Parameters>
    <Parameter>
      <Name><![CDATA[delayTime]]></Name>
      <Value Class="CodeUnitValue">
        <Code><![CDATA[0.0]]></Code>
        <Unit Class="TimeUnits"><![CDATA[SECOND]]></Unit>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[capacity]]></Name>
```

```
      <Value Class="CodeValue">
        <Code><![CDATA[1]]></Code>
      </Value>
    </Parameter>
    <Parameter>
      <Name><![CDATA[onAtExit]]></Name>
      <Value Class="CodeValue">
        <Code><![CDATA[agent.v = agent.v-1;]]></Code>
      </Value>
    </Parameter>
  </Parameters>
</EmbeddedObject>
```

4.4.4.21 ObjectFlow

Object flows are translated as connectors. These have a source and a target, each of which can be either a port of the current active object class, or a port of an embedded object.

The following code shows the content for an *ObjectFlow* in XML.

```
<Connector>
  <Id>2147483715</Id>
  <Name><![CDATA[csourcecOut_to_acombinerinput2]]></Name>
  ...
  <SourceEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[csource]]></ItemName>
  </SourceEmbeddedObjectReference>
  <SourceConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Source]]></ClassName>
    <ItemName><![CDATA[out]]></ItemName>
  </SourceConnectableItemReference>
  <TargetEmbeddedObjectReference>
    <PackageName><![CDATA[CompleteModel]]></PackageName>
    <ClassName><![CDATA[Complete]]></ClassName>
    <ItemName><![CDATA[acombiner]]></ItemName>
  </TargetEmbeddedObjectReference>
  <TargetConnectableItemReference>
    <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]>
    </PackageName>
    <ClassName><![CDATA[Assembler]]></ClassName>
    <ItemName><![CDATA[in2]]></ItemName>
  </TargetConnectableItemReference>
  ...
</Connector>
```

4.5 Translation and Discrete Event Simulation of the Discrete Event Network example

The DEN model presented in Subsection 4.3 can be translated and simulated on two DES tools [4]. As mentioned in Subsection 3.7, DES results will be the similar on both, but they are not likely to be exactly the same due to the stochastic nature of these kind of simulations.

4.5.1 SimEvents

After translating the model and running the simulation for 500s, Figures 29, 30, 31, and 32 show the sources and the sinks of the LTL and parcel commodities, respectively.

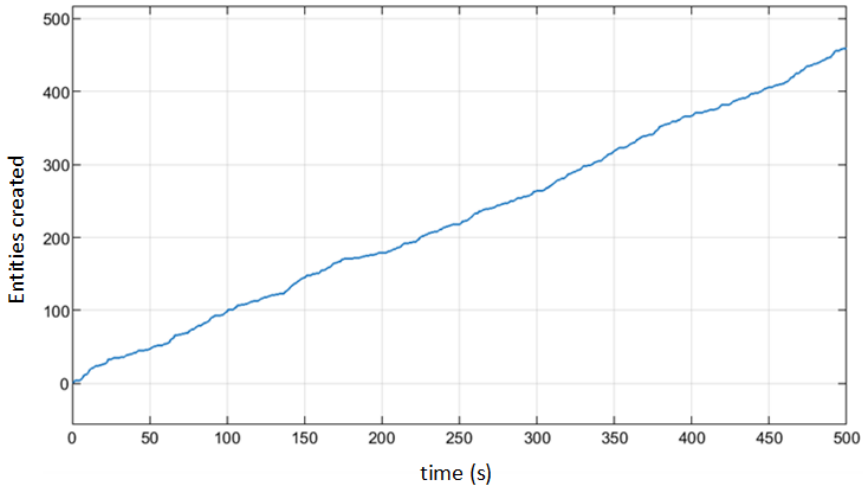

Figure 29: Discrete Event Simulation, activity, LTL source

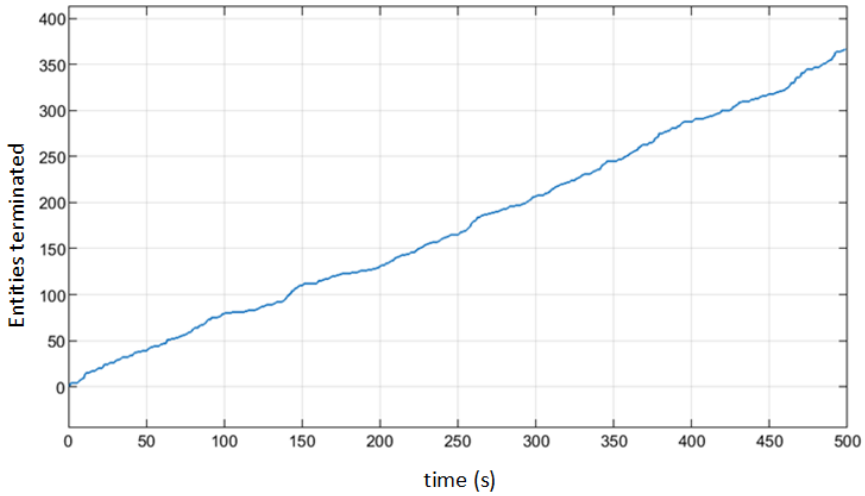

Figure 30: Discrete Event Simulation, activity, LTL sink

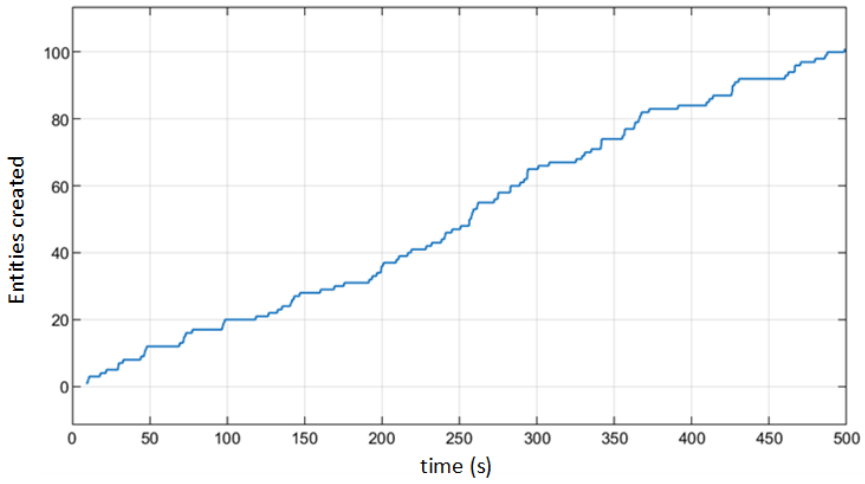

Figure 31: Discrete Event Simulation, activity, Parcel source

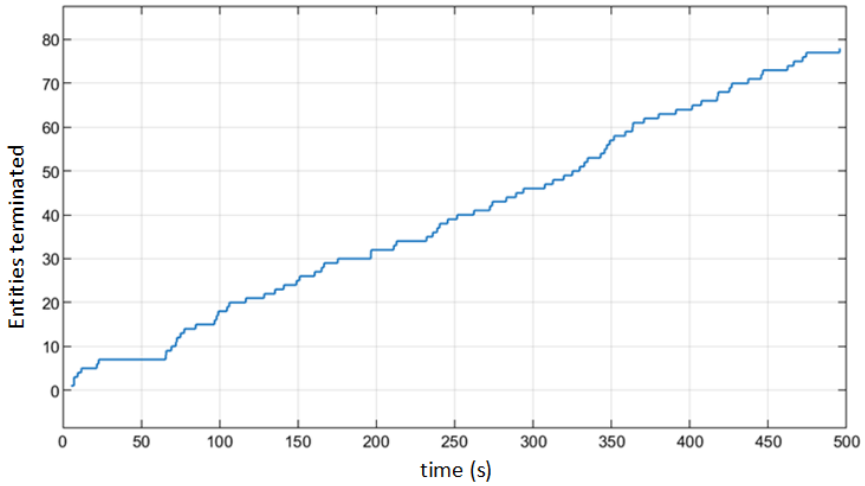

Figure 32: Discrete Event Simulation, activity, Parcel sink

These show that, in the Parcel shipment activity, there were about 100 carton orders received (produced), and 80 cartons were shipped (consumed). In the LTL shipment activity, there were about 460 pallet order received (produced), and about 370 pallets were shipped (consumed).

4.5.2 AnyLogic

After translating the model and running the simulation for 500s, Figures 33 and 34 show the content of the LTL and parcel shipment activities, respectively. The blue symbols with a cross inside are sinks, and the blue symbols with ”+>” inside are sources in AnyLogic. These are connected to ports, which are represented by a black dot.

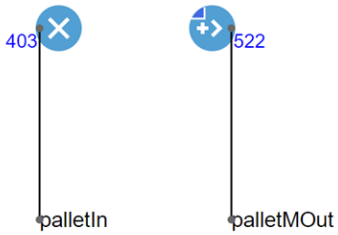

Figure 33: Discrete Event Simulation, activity, LTL

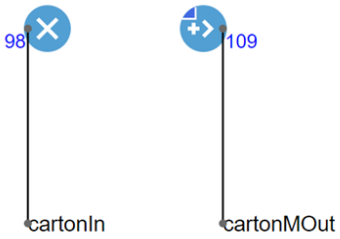

Figure 34: Discrete Event Simulation, activity, Parcel

These results are consistent with the results obtained in Subsection 4.5.1. The LTL shipment activity saw 403 pallets getting in from the `palletIn` port and going to a sink, and 522 pallet orders being created in a source and going out to the `palletMOut` port.The Parcel shipment activity saw 98 carton orders getting in from the `cartonIn` port and going into a sink, and 109 cartons being created in a source and getting out to the `cartonMOut` port.

5 Event distributions

This section defines Event Distributions, for network models to specify the time between random events in commodity flow networks, processing networks, and discrete event networks.

5.1 Modeling and analysis

Logistics modeling often involves specifying random times between events as probability distributions. These distributions can specify the time:

- between the creation of two successive commodities (inter-generation time) in *Produce* activities;
- taken to process or serve a commodity in *AtomicProcessNetwork*s, *Serve* activities, or *Process* activities (service time);

- between the passage of two successive commodities through a location in all kinds of network.

In Commodity Flow Networks (CFNs) or Processing Networks (PNs), stereotypes in the Event Distribution profile can be applied to the same properties as Rate (see Subsection 2.2.2) or ServiceTime (see Subsection 3.2.2, indicating that time intervals between commodities or service times are randomly distributed. In Discrete Event Networks (DENs), they can be applied to the *intergenerationTime* of *Produce* activities or the *delayTime* of the *Delay*. Finally, they can also be applied to slots referring to all these properties, which allows event distributions to be specified using initial values (see Subsections 2.4.2.2 and 3.4.2.2).

For each probability distribution, the following characteristics are given:

- probability density function (PDF): gives the probability of a random value being within an interval (by integrating over that interval);
- cumulative distribution function (CDF): gives the probability of a random value being below a certain value
- mean: the expected value, or average.

The rate of a distribution in this specification is the inverse of its mean.

5.2 SysML extension

The SysML representation of event distributions consists of a profile defining a taxonomy of DistributedProperty stereotypes corresponding to the distributions commonly supported by Discrete Event Simulation (DES) tools.

5.2.1 Profile

Figure 35 shows the Event Distribution profile. These stereotypes specialize SysML DistributedProperty, starting with EventDistribution, which extends UML Slot, enabling event distributions to be applied to either properties or initial values.

5.2.1.1 Beta

Values of properties stereotyped by Beta vary according to the beta distribution, which has the characteristics described in 4 on $(0, 1)$.

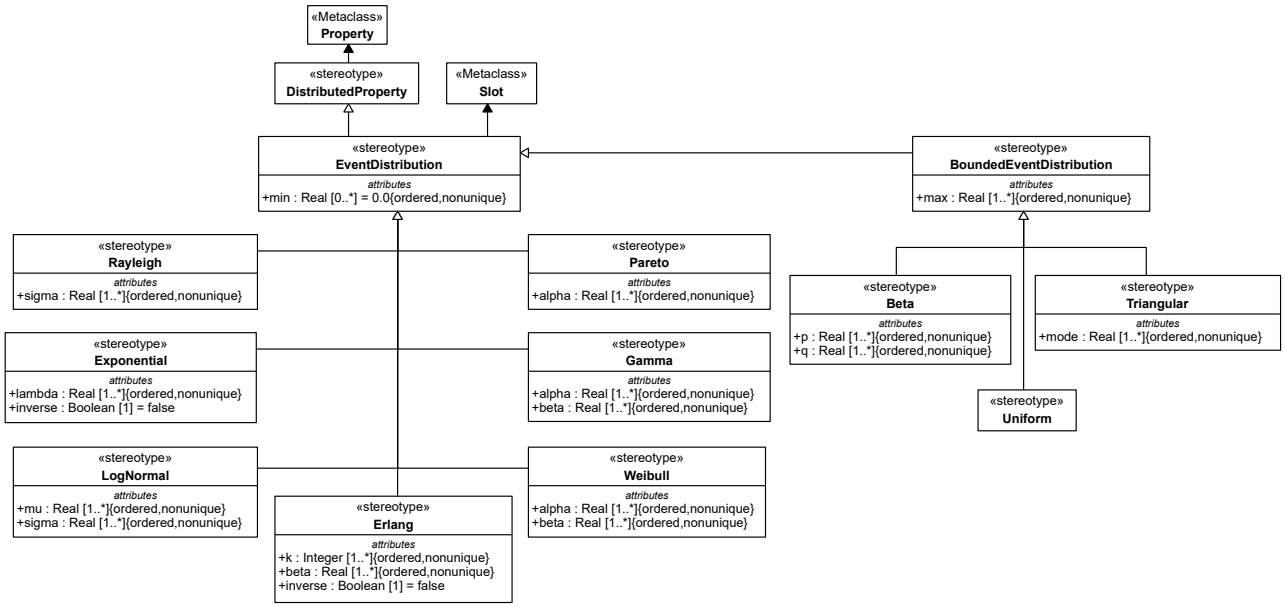

Figure 35: Event distribution Profile

PDF	$f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 t^{\alpha-1}(1-t)^{\beta-1}dt}$
CDF	$I_x = \frac{\int_0^x t^{\alpha-1}(1-t)^{\beta-1}dt}{\int_0^1 t^{\alpha-1}(1-t)^{\beta-1}dt}$
Mean	$\frac{\alpha}{\alpha+\beta}$

Table 4: Beta distribution

Values can be scaled and offset to fit the interval $]min, max[$

5.2.1.2 BoundedEventDistribution

Values of properties stereotyped by BoundedEventDistribution are random times or rates, with a minimum and maximum value. These values follow probability distributions specified by specializations of this stereotype.

Bounded distributions that apply to $(0, 1)$ can be rescaled to fit a given interval delimited by min and max: $x' = min + max \cdot x$

5.2.1.3 Erlang

Values of properties stereotyped by Erlang vary according to the Erlang distribution, which has the characteristics described in 5 on $[0, \infty)$.

Values can be offset to fit the interval $[min, \infty)$.

5.2.1.4 EventDistribution

Values of properties stereotyped by EventDistribution are random times or rates, with a minimum

PDF	$f(x) = \frac{x^{k-1}e^{-\frac{x}{\beta}}}{\beta^k(k-1)!}$
CDF	$f(x) = 1 - \sum_{n=0}^{k-1} \frac{1}{n!} e^{-\frac{x}{\beta}} (\frac{x}{\beta})^n$
Mean	$k\beta$

Table 5: Erlang distribution

value. These values follow probability distributions specified by specializations of this stereotype.

Distributions can be fitted to accommodate a given minimum min: $x' = min + x$

5.2.1.5 Exponential

Values of properties stereotyped by Exponential vary according to the exponential distribution, which has the characteristics described in 6 on $[0, \infty)$.

PDF	$f(x) = \lambda e^{-\lambda x}$
CDF	$f(x) = 1 - e^{-\lambda x}$
Mean	$\frac{1}{\lambda}$

Table 6: Exponential distribution

Values can be offset to fit the interval $[min, \infty)$.

5.2.1.6 Gamma

Values of properties stereotyped by Gamma vary according to the gamma distribution, which has the characteristics described in 7 on $[0, \infty)$.

PDF $f(x) = \frac{x^{\alpha-1}e^{-\frac{x}{\beta}}}{\beta^\alpha(\alpha-1)!}$	
CDF	$f(x) = \frac{\int_0^{\beta x} t^{\alpha-1}e^{-t}dt}{\int_0^{\infty} t^{\alpha-1}e^{-t}dt}$
Mean	$\alpha\beta$

Table 7: Gamma distribution

Values can be offset to fit the interval $[min, \infty)$.

5.2.1.7 LogNormal

Values of properties stereotyped by LogNormal vary according to the log-normal distribution, which has the characteristics described in 8 on $[0, \infty)$.

Values can be offset to fit the interval $[min, \infty)$.

5.2.1.8 Pareto

Values of properties stereotyped by Pareto vary according to the Pareto distribution, which has the characteristics described in 9 on $[0, \infty)$.

PDF	$f(x) = \frac{e^{\frac{(lnx-\mu)^2}{2\sigma^2}}}{x\sigma\sqrt{2\pi}}$
CDF	$f(x) = \frac{1}{2}(1 + \frac{2}{\sqrt{\pi}} \int_0^{\frac{lnx-\mu}{\sigma\sqrt{2}}} e^{-t^2} dt)$
Mean	$e^{\mu+\frac{\sigma^2}{2}}$

Table 8: LogNormal distribution

PDF	$f(x) = \frac{\alpha\ min^\alpha}{x^{\alpha+1}}$
CDF	$f(x) = 1 - (\frac{min}{x})^\alpha$
Mean	$\begin{cases} \infty, & \text{if } \alpha \leqslant 1 \\ \frac{\alpha\ min}{\alpha-1}, & \text{otherwise} \end{cases}$

Table 9: Pareto distribution

Values can be offset to fit the interval $[min, \infty)$.

5.2.1.9 Rayleigh

Values of properties stereotyped by Rayleigh vary according to the Rayleigh distribution, which has the characteristics described in 10 on $[0, \infty)$.

PDF	$f(x) = \frac{x}{\sigma^2} e^{\frac{-x^2}{2\sigma^2}}$
CDF	$f(x) = 1 - e^{\frac{-x^2}{2\sigma^2}}$
Mean	$\sigma\sqrt{\frac{\pi}{2}}$

Table 10: Rayleigh distribution

Values can be offset to fit the interval $[min, \infty)$.

5.2.1.10 Triangular

Values of properties stereotyped by Triangular vary according to the triangular distribution, which has the characteristics described in 11 on (min, max) .

5.2.1.11 Uniform

Values of properties stereotyped by Uniform vary according to the uniform distribution, which has the characteristics described in 12 on (min, max) .

PDF $f(x) = \begin{cases} 0 & \text{if } x < min \\ \frac{2(x-min)}{(max-min)(mode-min)} & \text{if } min \leqslant x \leqslant mode \\ \frac{2}{(max-min)} & \text{if } x = mode \\ \frac{2(max-x)}{(max-min)(max-mode)} & \text{if } mode \leqslant x \leqslant max \\ 0 & \text{if } x > max \end{cases}$

CDF $f(x) = \begin{cases} 0 & \text{if } x < min \\ \frac{(x-min)^2}{(max-min)(mode-min)} & \text{if } min < x \leqslant mode \\ 1 - \frac{(max-x)^2}{(max-min)(max-mode)} & \text{if } mode < x \leqslant max \\ 1 & \text{if } x > max \end{cases}$

Mean $\frac{min+mode+max}{3}$

Table 11: Triangular distribution

PDF $f(x) = \begin{cases} \frac{1}{max-min} & \text{if } min \leqslant x \leqslant max \\ 0 & \text{otherwise} \end{cases}$

CDF $f(x) = \begin{cases} 0 & \text{if } x \leqslant min \\ 1 & \text{if } x \geqslant max \\ \frac{x-min}{max-min} & \text{otherwise} \end{cases}$

Mean $\frac{min+max}{2}$

Table 12: Uniform distribution

5.2.1.12 Weibull

Values of properties stereotyped by Weibull vary according to the Weibull distribution, which has the characteristics described in 13 on $[0, \infty)$.

PDF $f(x) = \frac{\alpha}{\beta}(\frac{x}{\beta})^{\alpha-1} e^{-(\frac{x}{\beta})^{\alpha}}$

CDF $f(x) = 1 - e^{-(\frac{x}{\beta})^{\alpha}}$

Mean $\beta \int_0^{\infty} t^{\frac{1}{\alpha}} e^{-t} dt$

Table 13: Weibull distribution

6 Summary and future work

This report presents an extension of SysML for specifying logistics systems (SysLMA), and translation patterns from extended models to logistics analysis languages and tools. The extension is divided into three layers. Commodity Flow Networks (CFNs) describe networks with associated flow rates, translatable to Multi-Commodity Flow Optimization (MCFO) tools. Processing Networks (PNs) add network nodes that queue and serve commodities, translatable to Queuing Analysis (QA)

tools. Discrete Event Network (DEN) nodes give more detailed and specific behaviors, translatable to Discrete Event Simulation (DES) tools. A prototype software implementation of the translation patterns automatically translates SysLMA models to logistics analysis in several analysis tools.

This work could also be extended to cover other kinds of analysis, based on the current practices in logistics systems modeling, such as key performance indicators. It could also be applied to other areas, such as packet-based computer networks and patient logistics in hospitals. More realistic examples would be helpful, such as DENs that better coordinate flowing information and physical things, such as orders and commodities. The extension could also be upgraded second version of SysML (SysML2) [9], in part to represent analyses cases and connect logistics models and requirements, as a potential standard extension. Another research direction could be to incorporate more capabilities from the Discrete Event Logistics Systems specification [3], such as product, process, and resource (PPR) models.

Acknowledgments

We thank Tim Sprock, George Theirs, and Leon McGinnis for their past work on the topic of this report, and Theirs for his detailed review. Dave Wollman and Allison Barnard Feeney also provided helpful comments.

References

[1] Object Management Group, *Systems Modeling Language, version 1.6*, 2019. [Online]. Available: https://www.omg.org/spec/SysML/1.6/.

[2] Object Management Group, *Unified Modeling Language, version 2.5.1*, 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1/.

[3] T. A. Sprock, G. Thiers, L. F. McGinnis, and C. E. Bock, "Theory of Discrete Event Logistics Systems (DELS) Specification," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NISTIR 8262, 2020. DOI: 10.6028/NIST.IR.8262.

[4] R. Barbau, *SysML Extension for Logistics Modeling and Analysis translator*, 2025. [Online]. Available: https://doi.org/10.18434/mds2-3786.

[5] C. Bock and C. Galey, "Integrating four-dimensional ontology and systems requirements modelling," *J. of Engineering Design*, vol. 30, no. 10-12, pp. 477–522, 2019. DOI: 10.1080/09544828.2019.1642461.

[6] M. Marzolla, "The qnetworks Toolbox: A Software Package for Queueing Networks Analysis," in *Analytical and Stochastic Modeling Techniques and Applications, 17th International Conference, ASMTA 2010, Cardiff, UK, June 14-16, 2010. Proceedings*, K. Al-Begain, D. Fiems, and W. J. Knottenbelt, Eds., ser. Lecture Notes in Computer Science, vol. 6148, Springer, 2010, pp. 102–116, ISBN: 978-3-642-13567-5.

[7] T. Parr, *The Definitive ANTLR 4 Reference*. The Pragmatic Programmers, 2013, ISBN: 978-1934356999.

[8] L. F. McGinnis and T. A. Sprock, "Integrating analysis into a warehouse design workflow," in *13th IMHRC Proceedings*, vol. 5, 2014, ISBN: 978-1882780183. [Online]. Available: https://digitalcommons.georgiasouthern.edu/pmhr_2014/5.

[9] Object Management Group, *Systems Modeling Language, version 2*, 2025. [Online]. Available: https://www.omg.org/spec/SysML.

[10] R. Barbau, C. Bock, and M. Dadfarnia, "Translator from Extended SysML to Physical Interaction and Signal Flow Simulation Platforms, Version 1.1," *NIST Journal of Research*, vol. 126, no. 126027, 2021. DOI: 10.6028/jres.126.027.

[11] R. Barbau and C. Bock, "Verifying executability of SysML behavior models using satisfiability modulo theory solvers," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST IR 8283, 2020. DOI: 10.6028/NIST.IR.8283.

[12] Eclipse Foundation, *Eclipse Modeling Framework*, 2025. [Online]. Available: https://projects.eclipse.org/projects/modeling.emf.emf.

[13] Object Management Group, *XML Metadata Interchange, version 2.5.1*, 2015. [Online]. Available: https://www.omg.org/spec/XMI/2.5.1/.

[14] C. Bock, "UML 2 Composition Model," *Journal of Object Technology*, vol. 3, no. 10, pp. 47–73, 2004. DOI: 10.5381/jot.2004.3.10.c5.

[15] C. Bock, "UML 2 Activity and Action Models," *Journal of Object Technology*, vol. 2, no. 4, pp. 43–53, 2003. DOI: 10.5381/jot.2003.2.4.c3.

[16] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley, 2004, ISBN: 978-0201788914.

A Prototype implementation

The translations of extended SysML model examples to analysis tool inputs provided in this report, as well as results of analysis, were produced by a prototype implementation of the translation patterns also defined in this report. Figure 36 is an overview of the translator architecture. It is written in Java, leveraging previous work (SysPhS 1.1 translator [10] and behavior verification [11]). The Eclipse Modeling Framework (EMF) is used to deserialize SysLMA models, libraries [12] , and profiles in XML Metadata Interchange (XMI) into its repository [13]. The translator queries the repository and generates a single analysis tool-independent Unified Modeling Language (UML)[2] model of analysis input information, according to the translation patterns described in this report. These are serialized into input files for each analysis tool.

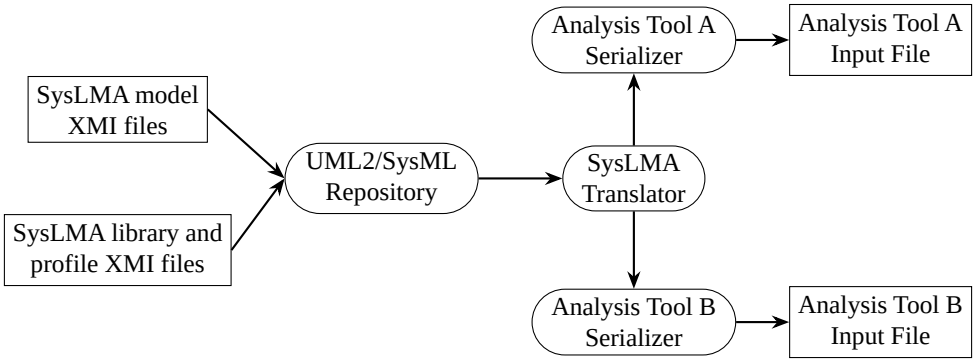

Figure 36: Prototype implementation architecture

Logistics systems models defined with the SysLMA libraries and profiles can be translated into analysis models as follows:

- CFN models can be translated into the following flow network optimization models:
  - AMPL model and data;
  - MATLAB script and data.
- PN models can be translated into the following queuing analysis models:
  - Octave script and data;
  - MATLAB model and data.
- DEN models can be translated into the following discrete event simulation models:
  - SimEvent file;

– AnyLogic file.

Figure 37 shows workflows for generating the above.

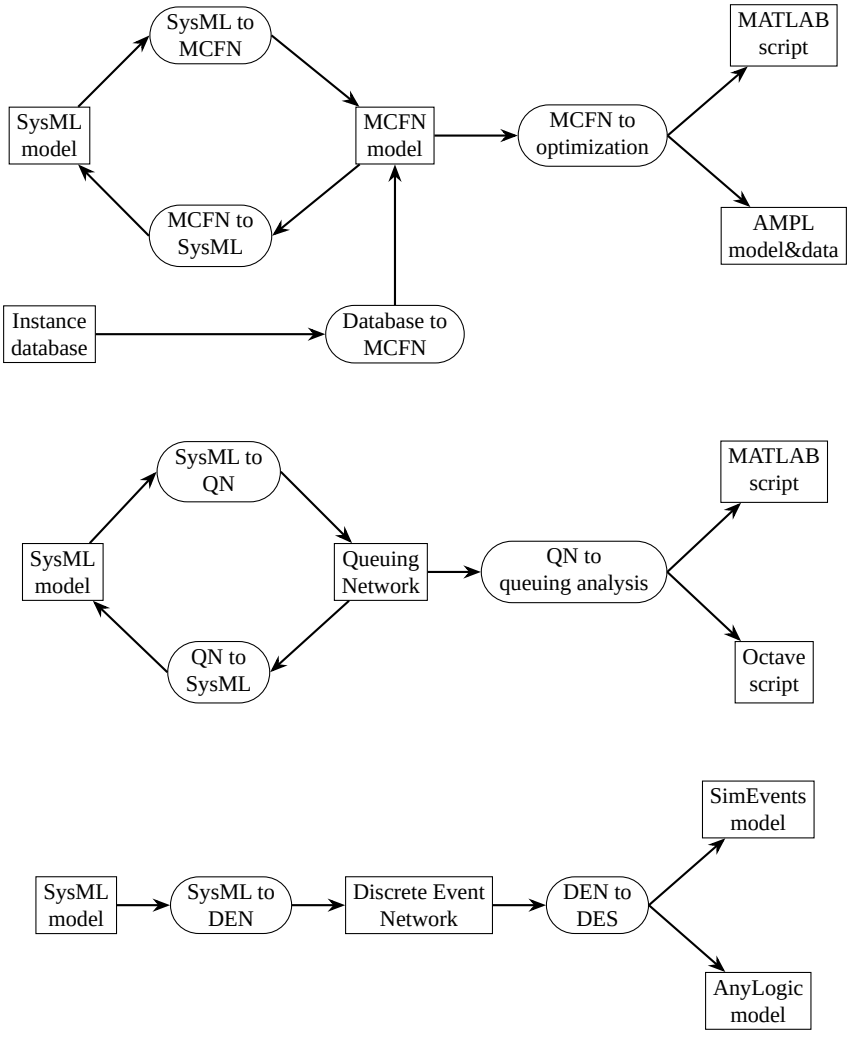

Figure 37: Prototype usage workflow

The translator enables analysis "chaining". For example, the result of Commodity Flow Network (CFN) optimization can be brought back to SysML and used as an input for Queuing Analysis (QA), as shown in the top workflow in Figure 37.

Source and compiled code for the translator is available publicly [4], along with all the library, profile, and example files used in this report.

B Naming style conventions

These case conventions are adopted for SysML model element names:

- Blocks and activities: start with an Uppercase letter, and the rest in CamelCase.
- Properties and metaproperties: start with a lowercase letter, and the rest in CamelCase.
- Enumeration literals are in all Uppercase.

These font styles are applied to names of the above, UML metaelements, and analysis tool elements:

- UML metaclasses, metaproperties, stereotypes, and stereotype properties: sans-serif, straight font.
- Blocks and properties: *sans-serif, italics*.
- Analysis tool elements: `teletype, straight`.
- Analysis tool parameters: *`teletype, italics`*.
- Analysis tool parameter values: *`teletype, slanted`*.

Italics in normal font are first use of a technical term.

C SysML introduction

SysML capabilities used in this report are:

- Classification of things being modeled.
- Attribution to specify their characteristics and relationships.
- Internal structure for aggregating them into larger ones.
- Activity modeling for process modeling.
- Model element classification (stereotyping).

SysML adopts UML's graphical notation for these. Classification gathers things being modeled according to commonalities between those things (*blocks*). For example, some objects might be gathered under a block for cars if they have wheels, transport less than six people at time, and have engines generating less than 500kW of power. Blocks form taxonomies according to degree of commonality among the things grouped under each class. For example, cars, buses, and trains all have wheels, but differ in the number of people they can transport and amount of power their engines can generate. A block for things that have wheels and transport people categorizes all the things that cars, buses, and trains do (*generalizes*) the blocks for cars, buses, and trains, which are *specialized* from the broader class).

Figure 38 shows blocks ("classes" in UML) for cars and four-wheel drive vehicles above the horizontal line, with dashed arrows from things being classified below the line (*instances*).[2,3] The dashed arrows indicate the instance called `Mary's Vehicle` is a car and a four-wheel drive vehicle, while the one called `John's Vehicle` is a car.[4] Figure 39 shows SysML's closed-head arrow notation for generalization. It introduces a block for four-wheel drive cars and generalizes it by the blocks from Figure 38. This generalization indicates that all instances classified as four-wheel drive cars are also classified as cars and as four-wheel drive vehicles. The classification arrows in Figure 38 and Figure 39 both express that Mary's vehicle is a car and a four-wheel-drive vehicle, while John's is a car.[5] Figure 39 could include the same classification arrows from Mary's vehicle as in Figure 38, but Figure 39's classification of Mary's vehicle as a four-wheel-drive vehicle, and its generalizations of four-wheel-drive vehicles, already imply the classifications in Figure 38.

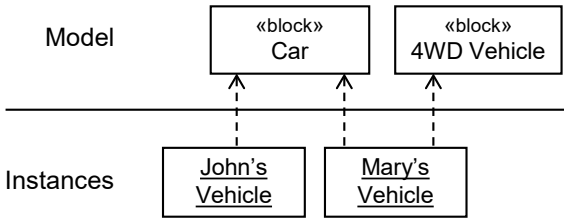

Figure 38: Classification

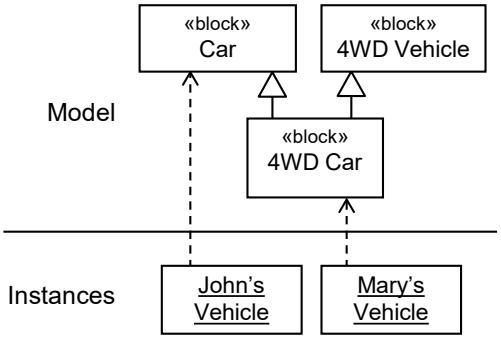

Figure 39: Generalization

Attribution covers potential characteristics of things being modeled as well as relationships between those things (*properties* or "roles"). For example, objects classified as cars might have a property for their weight, which would be required to give a single number (property *value*) for each object, based on a particular unit of measurement. These objects might also have properties identifying other objects they contain (with special values for that purpose), according to the role those things

[2]Dashed arrows for classification, and dividing lines between blocks and instances are used only in this appendix for illustration, they are not SyML/UML notation. Instances are real or virtual things. This appendix notates them as SysML/UML instance specifications, which are model elements, rather than real things.

[3]Blocks are typically shown in a *block definition diagrams* (BDDs), but the frames for these are omitted in this report for simplicity.

[4]Instance names are written in analysis tool element font, see Annex B.

[5]Since four-wheel drive vehicles are also cars, John's car might be four-wheel drive or not. Models capture knowledge at the time they are created, which might be incomplete.

play in them. For example, objects classified as cars might have properties identifying their wheels, with one property identifying the ones in the front of the car, and another identifying those in back. These properties would be required to identify objects for each individual car that play the corresponding roles (front and back) and are classified properly (under a block for wheels). Figure 40 shows SysML properties, appearing in blocks below their names, separated by horizontal lines (*compartments*). Property names are shown before the colon and the kind of thing they identify appears after (property *type*). For example, the block for cars has a property *lfw* identifying objects classified as wheels, and playing the role of the left front wheel. Instances of (classified by) cars use property names followed equal signs to show values of properties for each particular instance. For example, the value of *lfw* for John's car is an instance `JLFW`, which plays the role of left front wheel in John's car, and is classified as a wheel on the right of the figure. As a wheel, `JLFW` has a `size` property with the value `499mm`, which conforms to the property type in *Wheel* (the type name includes a numeric range and unit for brevity). Properties of a block of things also apply to specialized blocks of those things, as illustrated on the left in Figure 5. All cars have weight, including four-wheel drive cars.

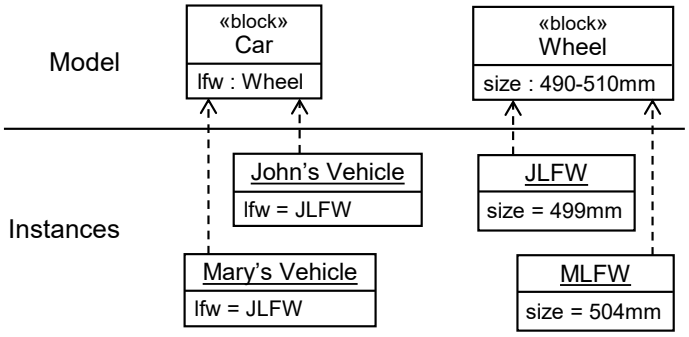

Figure 40: Attribution

A more graphical notation for properties uses lines drawn from blocks that have properties to other blocks that type those properties (*associations*), as shown in Figure 41. The *lfw* property on cars in Figure 40 is shown near a line between blocks for cars and wheels, on the end closer to the wheels. Another property can be shown on the other end, in this case a property of wheels identifying the car they are in. A similar notation can be used between instances showing *links* between them that are classified by associations. In Figure 41 these link John and Mary's cars to their left front wheels. Properties on each end of an association must have consistent values in instances of the associated blocks. For example, the value of *lfw* on John's car is `JLFW`, so the value of *inCar* on `JLFW` must be John's car, as shown at the bottom of Figure 41. Black diamonds on one end of associations indicate instances at the other end are dependent for their existence on the instance at the diamond end (*composite* aggregation). For example, the black diamond on the car end in Figure 41 indicates that destroying John's or Mary's car will destroy their left front wheels.

Properties can be restricted by constructs similar to generalization:

- One property might have values that are always values of another (*subsetting*), such as people's sisters always being their siblings. Figure 42 shows a property *iw* for impeller wheels of cars (the ones being driven by the engine) subsetting another property *w* for all wheels.

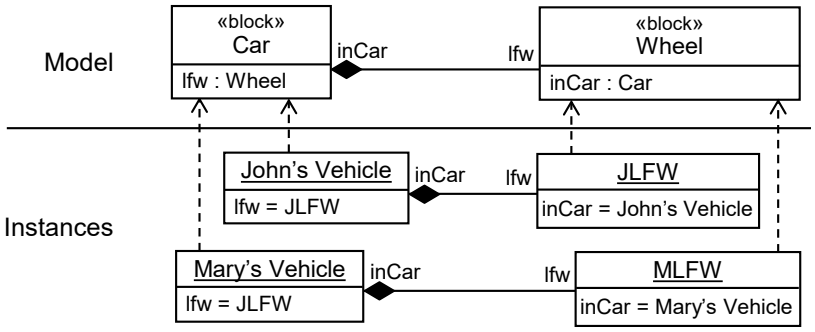

Figure 41: Association

This means values of *iw* are always included among the values of *w* (impeller wheels are always wheels of the car). Numbers in square brackets after property names indicate how many values a property might have on each thing in its block (*multiplicity*). The properties of car wheels in Figure 42 indicate cars might have 2, 3, or 4 wheels, some or all of which might be impellers (this is restricted in four-wheel drive cars, see next). The multiplicity symbol * indicates a property might have any number of values, including none. Subsetting properties can also limit the type of their values.

- An inherited property might have its values restricted to narrower blocks and numbers of things (*redefining*). Figure 42 shows the property *iw* inherited to four-wheel drive cars and restricted to have exactly four values that are large wheels. Redefining properties can also change the inherited name.

Figure 42 also shows the caret notation for the inherited *weight* property, see the discussion of Figure 40. These properties are not restricted in any way and are often omitted from diagrams.

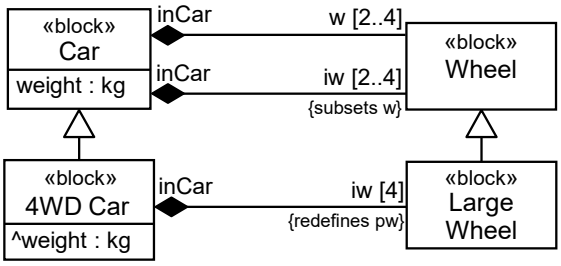

Figure 42: Property and association generalization

Internal block structure specifies how things are linked together into larger wholes based on *whole-part* properties identifying the objects being composed, and *part-part* relationships specifying links between the identified objects. For example, objects classified as cars might have a whole-part property for their engines, and another for wheels driven by the engine. These two properties can be linked by a part-part relationship that uses an association between engines and wheels they drive. The part-part relationship ensures that the object playing the role of engine in each individual car is linked (using its association) to the objects playing the role of driven wheels in the same car as the engine (not wheels in other cars).

Figure 43 shows SysML notation for this as a *structure* compartment in blocks[6] It uses the same graphical notation as blocks and associations, but with different meanings. Rectangles show whole-part properties, while lines show part-part relationships (*connectors* in SysML). Connectors are typed by associations (analogous to typing properties), and show property names from those associations. Values of whole-part properties use properties of connecting associations to identify (link to) each other. For example, Figure 43 shows whole-part properties of cars as rectangles in a structure compartment. These properties identify objects that power and impel cars, which must be engines and wheels, respectively. Text inside property rectangles is the same as appears in property compartments. The connector between whole-part properties is typed by the association between engines and wheels on the right of the figure. The association provides properties for linking values of the whole-part properties (identifying power sources and impellers) to each other in each individual car. Examples are shown by the instances at the bottom of the figure (classification is sometimes notated in instance labels to the right of a colon after instance names, for brevity). Connectors ensure the association between engines and wheels is used within each car individually, rather than between engines and wheels in multiple cars. More explanation of block structure in SysML is available in [14].

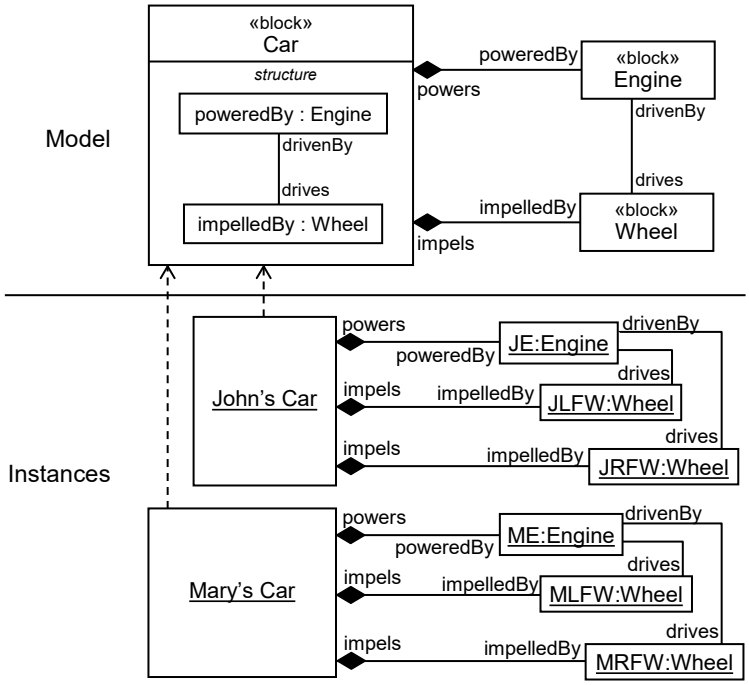

Figure 43: Internal block structure

SysML can model behaviors in the same way as above, as another kind of thing to be classified, attributed, and aggregated. One of these is for specifying sequences of steps in a process and how things flow between them (*activities*) [15]. Activities can be treated as blocks, forming taxonomies, as in Figure 44. It classifies various car behaviors according to whether a car is operating or not, if operating whether is is moving or stationary, and so on. Attributes such as a car's location and speed appear as properties on the applicable blocks. Composite associations between activities indicate

[6]It can also appear in *internal block diagrams*, IBDs, see example in Figure 5, Section 2.3

one activity happening during another. The other associations relate a car to objects involved in carrying out a behavior. Each instance of an activity represents the activity being carried out, during particular time periods, one for each time the activity is carried out.

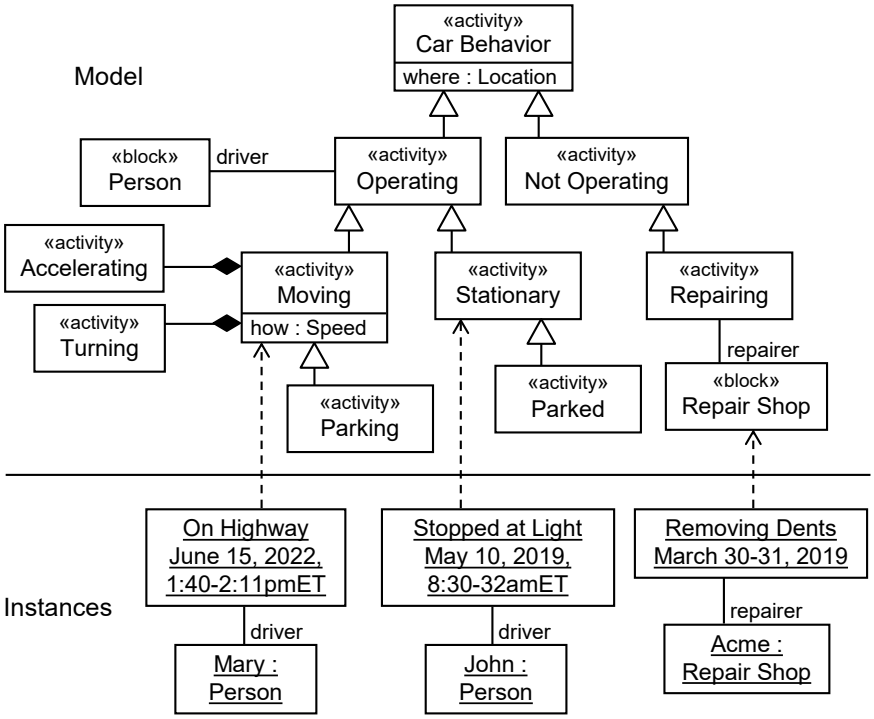

Figure 44: Activity classification

Elements within activities included these elements used in this report:

- Parameters for activities to accept input and provide output.
- Actions for steps taken by an activity. This report uses
  - CallBehaviorActions to identify another activity to carry out a step.
  - OpaqueActions to specify a step in non-SysML language.
- Pins for actions to accept input and provide output. In this report they can accept and provide any number of values (commodities in this report) at any time, including while actions are ongoing (*streaming*).
- Decision and merge nodes to split and merge the object flows.

See Figure 24 in Section 4.3 for an example activity model using some of these elements. Actions are notated as round cornered rectangles, pins as small recangles on actions, and object flows as arrows between pins. Section 4.4.1 gives more information about activity elements used in this report.

The techniques for classification and attribution outlined at the beginning of this appendix can also describe model elements (*metamodeling*) [16], as shown in Figure 45. Here *metaclasses* from UML's *metamodel* at the top classify model elements at the bottom, shown in SysML modeling notation rather than as instance specifications, for clarity. For example, Generalization classifies the generalization between *4WD Car* and *Car*, while Property classifies the property *lfw*. SysML adopts UML's facility for specializing its metamodel (*profiles*), including *stereotypes*, which enables models to have the effect of specializing metamodels in implementations where metamodels cannot be modified. SysML's Block stereotype effectively specializes (is *based on*) UML's Class, the metaelement for model elements that classify instances, notated by a filled-head arrow, as shown in Figure 45. This arrow indicates that all model elements classified as blocks are also classified as cars and as classes, enabling them to classify the instances shown in earlier figures. Stereotypes also support attribution, identifying instances of other metaclasses and stereotypes, see Figure 3 in Section 2.2.2.

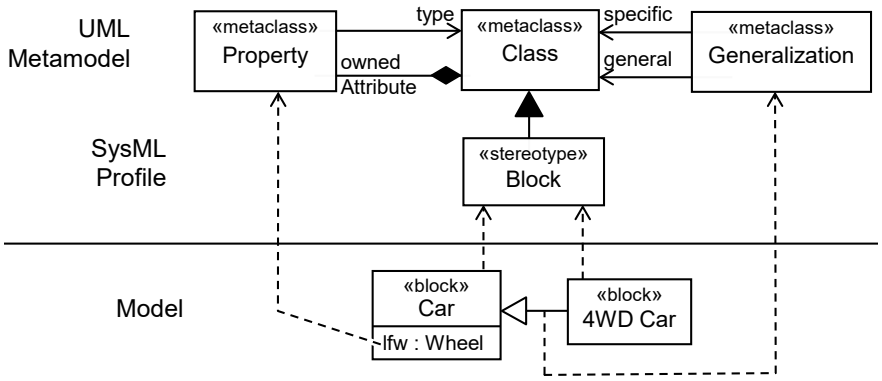

Figure 45: Metamodeling