**%figure 1**

SimulationSystem4MS(x) → ∃y InputGenerator(y) ∧ composes(x, y)

SimulationSystem4MS(x) → ∃y OutputCollector(y) ∧ composes(x, y)

SimulationSystem4MS(x) → ∃!y MS_sim(y) ∧ composes(x, y)

SimulationSystem4MS(x) → ∃!y Configurator(y) ∧ composes(x, y)

SimulationSystem4MS(x) → ∃!y ResultsManager(y) ∧ composes(x, y)

**%figure 2**

C_Port(x) → DataPort(x)

FU_Port(x) → DataPort(x)

%type ??

In both SysML and Modelica, it is necessary to distinguish between the creation of a port (which you can stereotype) and its definition (its type), which is defined in a block. The relationship established in the diagram means that any port stereotyped as FU_Port must be defined by a FlowUnit block.

**%figure 3**

MS_sim(x) → ProcessingResource_sim(x)

ProcessingResource_sim(x) → ManufResource_sim(x)

ControlResource_sim(x) → ManufResource_sim(x)

% ∃>=0 this is not a constraint → It's true, it makes sense

ProcessingResource_sim(x) → ∃>=0y ManufResource_sim(y) ∧ composes(x, y)

ManufResource_sim(x) → ∃>=0y C_Port(y) ∧ composes(x, y)

%??redundant with above rule, or perhaps it means that there are two parts, one C_Port and the other either again a C_Port or just a DataPort? → It can contain other generic ports, which do not correspond to either of the two defined specializations (C_Port or FU_Port), but as you say it is not a restriction.

ManufResource_sim(x) → ∃>=0y DataPort(y) ∧ composes(x, y)

%no cardinality between Product_sim and FlowUnit → It should be 1..*

%C5 → ¿?

ProcessingResource_sim(x) → ∃>=2y FU_Port(y) ∧ composes(x, y)

% no cardinality constraint? → 1

FlowUnit(x) → ∃>=0y Product_sim(y) ∧ Associated(x, y)%type ?? → Same in Figure 2

**%figure 4**

ControlResource_sim(x) → ManufResource_sim(x) %redundant with fig 3 → Yes, the same

ProcessingResource_sim(x) → ManufResource_sim(x) %redundant with fig 3 → Yes, the same

TransformResource_sim(x) → ProcessingResource_sim(x)

LogisticResource_sim(x) → ProcessingResource_sim(x)

ManufResource_sim(x) → ∃!y ManufResourceSpecif_data(y) ∧ aggregates(x, y)

TransformResource_sim(x) → ∃!y ActiveConfiguration(y) ∧ composes(x, y)

**%figure 5**

Product_sim(x) → ∃!y ActiveState(y) ∧ composes(x, y)

Product_sim(x) → ∃!y ProductSpecif_data(y) ∧ aggregates(x, y)

ActiveState(x) → ∃!y StateSpecif(y) ∧ Associates(x, y)

ProductSpecif_data(x) → ∃>=2y StateSpecif(y) ∧ composes(x, y)

ProductSpecif_data(x) → ∃!y NativeProcessPlan(y) ∧ composes(x, y)

NativeProcessPlan(x) → ∃y ResourceAllocation(y) ∧ composes(x, y)

% ?? also isAtomic: Boolean → In SysML stereotypes can have properties that allow conditioning some rules (if isAtomic==true then___ else__). In the case of this boolean property, it can be replaced, for example, by defining two specializations: Atomic_ManufProcess and NonAtomic_ManufProcess

NativeProcessPlan(x) → ∃y ManuProcess(y) ∧ composes(x, y)

**%figure 6**

ManufResource_sim(x) → BehavioralElement_sim(x)

InputGenerator(x) → BehavioralElement_sim(x)

OutputCollector(x) → BehavioralElement_sim(x)

BehavioralElement_sim(x) → ∃>=0y BehavioralElement_sim(y) ∧ composes(x, y)

BehavioralElement_sim(x) → ∃<=1y STM_MainBehavior(y) ∧ composes(x, y)

%and the isAtomic: Boolean thing → Same in Figure 5

**%figure 7**

%??

SimulationSystem4MS(x) → BehavioralElement_sim(x)

**%«BehavioralElement_sim»**

**%C1??**

BehavioralElement_sim(x) ∧ Atomic(x) → ∃!y STM_MainBehavior(y) ∧ composes(x, y) → Consider whether it is appropriate to establish the "Active" concept or to resolve it in another way, for example, with two specializations of the BehavioralElement_sim class.

%C1??

~~BehavioralElement_sim(x) ∧ Atomic(x) → Active(x)~~ → The term "active" was a way of expressing that it has its own defined behavior (previous rule), but it is not a new property.

%C1??

BehavioralElement_sim(x) ∧ Atomic(x) → ~∃y inheres(y,x) ∧ ?? → It cannot be composed of any part of type BehavioralElement_sim (or its specializations). Maybe it can be expressed as: ~∃y BehavioralElement_sim (y) ∧ composes(x, y) ??

**%C2**

BehavioralElement_sim(x) ∧ ~Atomic(x) → ~∃y STM_MainBehavior(y) ∧ composes(x, y)

**%«STM_MainBehavior»**

%dont see the metaclass <StateMachine> in fig. 7 → It is a UML metaclass (and inherited in SysML), not defined in the specified profile, but in the base language like Block.

**%«SimulationSystem4MS»**

%C3 unclear if there is some/any correspondance with constraints from figure 1 → This rule should transfer the same compositions represented in figure 1, but the multiplicities must be discussed.

**%«Configurator»**

%no constraints

**%«InputGenerator»**

%C4

InputGenerator(x) → ∃y FU_Port(y) ∧ composes(x, y)

%C4

InputGenerator(x) → ∃y C_Port(y) ∧ composes(x, y)

%??"although they can also be applied to the same port" → it is an internal comment to express that a port can have two stereotypes applied simultaneously. It could contain a port that is both FU_Port and C_Port. To do this, the rules in these stereotypes must be compatible.

**%«OutputCollector»**

%C5

OutputCollector(x) → ∃y FU_Port(y) ∧ composes(x, y)

%C5

OutputCollector(x) → ∃y C_Port(y) ∧ composes(x, y)

%??"although they can also be applied to the same port" → same in C4

**%«ResultsManager»**

%C6

ResultsManager(x) → ∃y DataPort(y) ∧ composes(x, y)

**%«DataPort»**

**%«FU_Port»**

%C7?? → "type" association in Figure 2. Any port stereotyped as FU_Port must be defined by a «FlowUnit» block.

**%«C_Port»**

**%«FlowUnit»**

%C8 ??why "composed of uses"?; already in figure 3? → From a manufacturing point of view, the FlowUnit is a batch, a set of products that can be a single product or a set of multiple products (1..*).

**%«ManufResource_sim»**

%??Is this a correct interpretation? → I think that is correct. Any manufacturing resource can be composed of other manufacturing resources (workshop, work cell, assembly line,...) except if it is a "subphase" (the atomic level). What I am not clear about is how to transfer the attributes of a stereotype. I understand that they are also classes in your model, is not?

ManufResource_sim(x) → (Subphase(x) ↔ ~∃ y ManufResource_sim(y) ∧ composes(x, y))

**%«ManufResourceSpecif_data»**

%configspecif did not appear before → It should be added to Figure 4

ManufResourceSpecif_data(x) ∧ isTransformative(x) → ∃y ConfigSpecif(y) ∧ composes(x, y)

**%«ProcessingResource_sim»**

%C5 already in figure 3

**%«TansformResource_sim»**

%C11 ??Type relation=?

TansformResource_sim(x) → ∃y,z,v,w,f,g FU_Port(y) ∧ FU_Port(z) ∧ FlowUnit(f) ∧ FlowUnit(g) ∧ Product_sim(v) ∧ Product_sim(w) ∧ composes(x, y) ∧ composes(x, z) ∧ Typed(y, f) ∧ Typed(z, g) ∧ composes(f, v) ∧ composes(g, w)

%C12 Object-Attribute relation=? → In SysML, stereotypes can be applied both to blocks (classes) and properties (each part, component). In this case, ActiveConfiguration is an stereotype for a property, not for the block defining it. I don't know if it fits with the proposed rule.

TansformResource_sim(x) → ∃y ActiveConfiguration(y) ∧ hasAttribute(x, y)

**%«ActiveConfiguration»**

%C13 ?? differences of typing vs subclassing = ? → Similar discussion in C12. Differences between stereotyping a class or a property.

**%«ConfigSpecif»**

%?? differences of typing vs subclassing = ? → Similar discussion in C12. Differences between stereotyping a class or a property.

**%«LogisticResource_sim»**

%??ProductBatchSim not introduced before → it should be "FlowUnit"

%C14 ??state of ProductSim = ? → Figure 5. A Product_sim (or ProductSim, the same) is composed by one ActiveState.

LogisticResource_sim(x) ∧ FU_Port(f) ∧ composes(x, f) ∧ FU_Port(g) ∧ composes(x, g) ∧ ProductBatchSim(y) ∧ Typed(f, y) ∧ ProductBatchSim(z) ∧ Typed(g, z) → ((y=z) ∨ ∃a,b (ProductSim(a) ∧ ProductSim(b) ∧ composes(y,a) ∧ composes(z,b) ∧ stateOf(a) = stateOf(b)) → I need some explanations to understand it completely, but I think it is OK.

**%«MS_sim»**

**%«ControlResource_sim**

%C15

ControlResource_sim(x) → ∃y C_Port(y) ∧ composes(x, y)

**%«Product_sim»**

%C16 %% "reference (shared relationship)" = ? → Aggregation to access to its content.

Add a new rule for the ActiveState property.

**%«ActiveState»**

%C17 same as other properties

**%«ProductSpecif_data»**

%C18

ProductSpecif_data(x) → ∃y ProcessPlan(y) ∧ composes(x, y)

**%«StateSpecif»**

%C19 same as other properties

**%«NativeProcessPlan»**

%?? action-activity relation = ? → In SysML, an action is a part of an activity, that can be described by another activity (similar to the relation between a property and its type)

%C20

NativeProcessPlan(x) ∧ Action(y) ∧ composes(x, y) → ManufProcess(y)

**%«ManufProcess»**

**%«ResourceAllocation»**

%C21

%?? partition = ? → It is a SysML concept to organize or group different actions in an activity description (similar to a package to organize actions), and also it can be related to specific parts of the model (all the actions performed by an specific part).