Concise modeling

# General

We propose a novel approach we call Concise Systems Modeling. In our approach we do not model the detailed systems, but rather the rules for their composition.

By creating a special profile of SysML with an assortment of stereotypes and tags we give additional semantics to SysML constructs, such as blocks, parts, links and associations. Thus we are able, for example, to use a SysML part to define a set of parts and similarly use a link to define a set of links. We call these sets “prototypes”. Since these prototypes are legal SysML parts interconnected with ports and links, we can define the composition (or connectivity) rules for those prototypes.

The explicit prototypes are later instantiated in tables (MS Excel for example), which is much easier and less time consuming than the graphical SysML representation. Some of the fields in the tables are left unfilled and these are later filled by an architecture optimization process.

The attributes of these prototypes are also stereotyped allowing their different treatment in the design process.

An internal block diagram in the technical layer may combine prototypes and normal parts.

The model has three layers – functional, technical and an indexing layer (which in many cases represents geometry). The technical layer is the core of the concise model and all component composition rules are modeled here. The functional layer models system functions mapping them later to specific elements in the technical layer, in essence defining requirements that the technical layer needs to satisfy. The indexing layer, which is not necessarily present, is used to index the technical layer prototypes and facilitates the later expansion or optimization of the model.

In addition to the above constraints, objectives and variable algebras are defined in the model, to be used by the optimization engine. Several new sets definitions have been added to aid in the above definitions.

# Planes / Layers

The concise modeling approach has three planes in three different model packages:

* Functional plane (Figure 31 & Figure 32)– serves as the requirements definition for the system architecture.
  + Will generally be executable.
  + May be modeled concisely in some cases, but all parts and links will be explicit (i.e. «inventory»).
  + May be a result of a higher abstraction iteration using the same approach.
  + May have links connecting ports or not.
* Technical plane (Figure 33) – architecture modeling plane. Modeling is based on the requirements of the functional layer. The objects on this plane usually represent real components (or subcomponents) and real flows between them (data, energy). The flows’ media are the Typed Connectors, which are parametric and/or behavioral models of cables, shafts, ducts, pipes, wireless channels, etc.
* Indexing plane (Figure 34) – used to index the objects of the technical layer. Sometimes this layer directly represents the geometry of the system and is used as such. For example the instances of this layer may represent possible placeholders for the actual components on the technical plane with the optimization process tasked with finding the right combination of components and their locations.  
  Alternatively this layer can be an abstract collection of indices bounded by constraints.
* Mapping – the way to relate one layer to the other. Mapping is done by using the SysML «allocate» dependency. An object on the functional plane can only be mapped to one object on the technical plane, as otherwise there would be ambiguity in the definition. However, any number of objects on the functional plane can be mapped to a single object on the technical plane. If a multiple mapping is indicated, the meaning is that the optimization must select the best mapping subject to constraints and rules.



Figure 31 – Primary EPS Functional view



Figure 32 – Secondary EPS Functional view

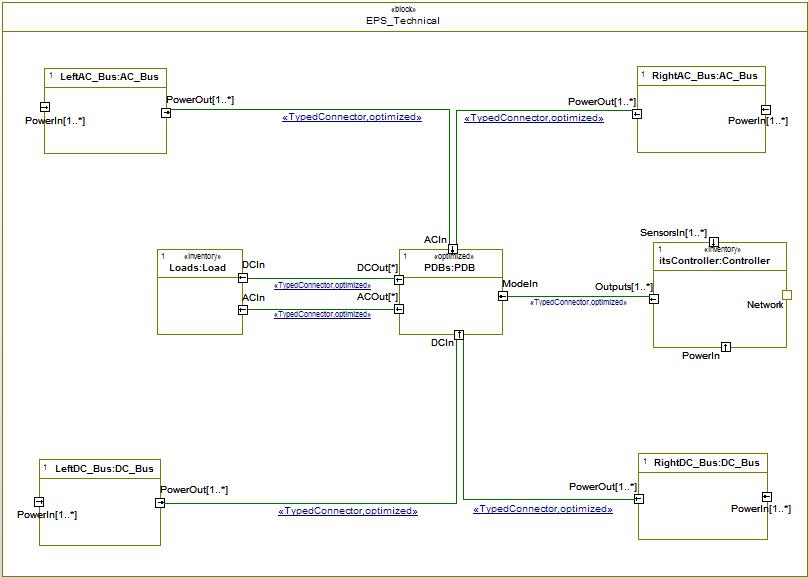


Figure 33 – Secondary EPS Technical View

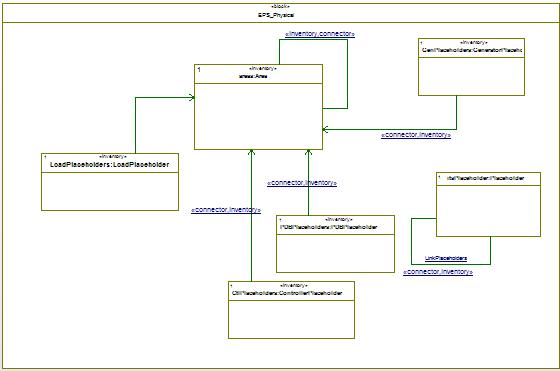


Figure 34 - Index (geometry) View

# SysML Extensions Profile

## «typedConnector»

This stereotype is used to denote a link (between parts or prototypes) that represents a concrete physical object, for example a cable, bolt or shaft. The stereotype contains a tag “type” that points to the block that models the physical object.

# Concise Profile

## «catalog»

Can be applied to blocks and to attributes.

When applied to a block, «catalog» indicates that a block has several variants (subclasses in the expanded model), which are specified in an external table instead of explicitly in the model.

Blocks marked with «catalog» may have attributes marked similarly. These attributes will have different initial values in each sub-class. These values are specified in the table – i.e. each variant will have a row in the table, and each catalog attribute will have a column.

When a normal (non-inventory) part instantiates a «catalog» block, its type will be replaced in the expanded model with a specific variant, as specified in the table. The table [of variants] may be filled from an external source or manually.

## «inventory»

Can be applied to parts (which are then called prototypes), attributes, tags, connectors (including typedConnectors) and dependencies. In essence this construct removes the need to model each and every part, connector or dependencies, using external tables instead (filled from an external source or manually)

* Parts, connectors and dependencies marked with «inventory» represent sets of elements in the expanded model.
  + In case a marked «inventory» part or typedConnector instantiates a block marked with «catalog», the specific type (variant) of each element in the set will specified in the external table.
  + In case of a connector or dependency, «inventory» also indicates that the end points for each element in set are specified in the table.
* When applied to an attribute or a tag, indicates that the value of that attribute in each instance in the set (part or typed connector) should appear in the instantiating tables.

## «optimized»

Can be applied to parts (prototypes), attributes, tags, connectors (including typedConnectors) and dependencies.

Much like «inventory», «optimized» allows to define sets of instances and to suggest that values for attributes and tags, instance type selection from catalog (where relevant), and endpoints (for connectors and dependencies) are defined in a table. The difference lies only in the source of the data – with «optimized», the data is to be generated by the optimization engine, not supplied manually or automatically from external tools.

Tags:

* max – the maximum number of elements to be created in the set. When applied on an attribute - this tag indicates its maximum value.
* min – the minimum number of elements to be created in the set. When applied on an attribute - this tag indicates its minimum value.

## «expand»

Can be applied to any object or link. Indicates that this object/link is to be present in the expanded technical model.

# Constraints Modeling Stereotypes

## «behaviorModeBased»

This stereotype is used when there is a need to model modes of behavior. The stereotype is applied to attributes to indicate that their value changes according to some predefined schema, which, when the stereotype is used, will be defined in the instantiation tables.

Since there may be various behavior scenarios with some scenarios decoupled from each other, the stereotype carries a tag “scenarioGroup” which helps to distinguish between such orthogonal behaviors.

Examples:

* In Figure 33 the “Loads” prototype represents a collection of electric devices on an aircraft. These electric devices power consumption varies with respect to the current mission phase of the aircraft (for instance a targeting system will consume only standby power during the cruise phase and will consume maximum power during the strike mission phase). Since one of our constraints in the architecture optimization process is to select PDBs that can carry enough current, all power consumption scenarios (mission phases) need to be checked. The attribute indicating the power requirements of each Load will be marked with the stereotype.
* In Figure 31 the power generating elements have finite reliability. Therefore the requirement from the relays’ network is to have a configuration (in terms of open-closed) for each failure scenario of the power generating elements, so that power is supplied to the buses (indicated by the “choice tree” concept. To model this with «behaviorModeBased» stereotype we create a new “scenarioGroup” and mark the “OK” attribute of the components with the stereotype.

## «derived»

This stereotype is applied to an attribute to indicate that the attribute value is calculated from other parameters. Normally when this is done that attribute will appear on the left side of a formula in a constraint owned by / anchored to the block owning the attribute.

## «objective»

This stereotype marks a «derived» attribute that is supposed to be one of the objectives of the optimization.

## «invalid»

This stereotype is applied to a functional link with the purpose of denoted an illegal link. When the functional layer is mapped to the technical the corresponding paths in the technical layer will be considered illegal and will not be chosen by the optimization engine.

## «VisualConstraintBlock»

This stereotype converts a block into a Visual Constraint. A Visual Constraint Block is a graphical (using SysML) method of depicting an architectural rule or a regular expression. A Visual Constraint Block is specified using a Constraint Diagram, and may contain explicit parts from the architecture, constraint variables typed by blocks from the architecture (representing some architecture part – or a set of parts, according to multiplicity - with the same type), or constraint variables typed by other Visual Constraint Blocks, representing a composite regular expression.

## «constraintDiagram»

This stereotype inherits from Internal Block Diagram. The newly defined Constraint Diagram is used to depict architectural constraints in graphical terms. Often the diagram would contain both functional and technical elements in order to define how a functional constraint is mapped to the technical plane.

Figure 35 shows a Constraint Diagram with a definition of illegal path between the left and the right generators, with a mapping to the technical level (modeled concisely). A constraints diagram may also include formulae constraining the values of attributes of some components.



Figure 35 - Constraint Diagram

# Variability Profile

Besides concise modeling, another useful approach to specifying architecture alternatives is to explicitly model them in a single model, and annotate them using variation points. For example, by specifying that a particular element has an "existence" variation point, the systems engineer indicates that the element may or may not exist in a given architecture. To allow for correlation of variation points (i.e. to specify that several elements go together) each variation point may be bound to a "choice" element. In order to derive a specific architecture from a model with alternatives, a decision for each choice (and unbound variation point) must be made by the optimization engine.

Choices may also be organized in a tree, which allows constraining the legal decisions for a given architecture: Each choice node in the tree defines how many of its child choices must be selected (if it is selected itself). For example, a choice may specify that exactly one of its child choices be selected. This makes all its children alternatives of one another. Respectively, their bound variation points also become alternatives. Other kinds of constraints (known as cross-tree constraints) between choices may also be used.

## The VariabilityRealization package: variation points

Figure 36 outlines the hierarchy of the Variability realization stereotypes.



Figure 36 - Variability Realization Package

### «ExistenceVP»

Specifies that the element to which it is applied may or may not exist in a particular architecture.

### «Substitution»

A kind of dependency which states that the source might substitute the target in a given configuration. All connections from / to the target (e.g. connectors) will be added to the source.

### «ValueAssignmentVP»

Element that allows assigning value to some field of a model element, e.g. an initial value of an attribute. The field to be assigned a value is specified by the tag "metaAttribute", which is a string, in the above example "initialValue".

## The VariabilityAbstraction package: choice trees

As explained above, the choice tree (actually, variability specification tree – including choices and values) allows modeling the high-level architectural choices and their relationships. Each such choice over variable may be bound to an element marked by a variation point stereotype.



Figure 37 - Variability Abstraction Package

### VariabilitySpecification

This is an abstract stereotype, which represents a node in the variability tree. Variability Specifications (VSpec) are a new kind of model element, implemented in Rhapsody as a new term based on SysML comments. This allows for their nesting, enables showing them on diagrams, and provides an easy binding mechanism to variation points, using anchors.

Each VSpec has the following tags:

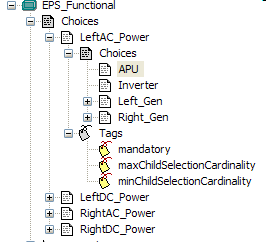


Figure 38 - Choice tree

* mandatory – indicates that a choice is selected together with its parent choice.
* maxChildSelectionCardinality – maximum number of child VSpecs that can be selected for a given architecture.
* minChildSelectionCardinality – minimum number of child VSpecs that can be selected for a given architecture.

The tree can have any number of levels and the leaf choices will carry no tags (the tag values will be empty).

### Choice

Choices are a kind of VSpec which requires a Boolean decision to be made when specifying a particular architecture out of the possible architecture variants. It may be bound to Existence and Substitution variation points.

On the functional plane a choice anchored to an object (or a link) indicates that the object may exist and may not. Since it is a functional object the above will indicate that the function is active or not at a given point in time. When a root choice has 4 sub-choices with max and min cardinality at 2 and 1 respectively, it would mean that at any given time in the system at least one function must be active and no more than two can be active.

On a technical plane the choice would indicate an architecture option. For example whether there should be a relay between two electric power components or just a wire. The architectural choice must, in this case, be made either by a human or by the optimization engine.

### Value

Values are a kind of VSpec which requires a value to be given when specifying a particular architecture out of the possible architecture variants. It may be bound to ValueAssignment variation points, which then takes the value and assigns it as explained above. The value type is specified using the "type" tag, whose type (VariabilityValueType) is an enumeration over basic data types (Integer, Float, Boolean, String, etc).

## Variability constraints

Besides the tree structure, which specifies constraints regarding allowed selections, we also allow specifying cross-tree constraints. We support both simple constraints such as excludes and requires and constraint expressions, which are logical expressions over the VSpecs.

### Requires

A simple constraint between two Choices. Implemented as a new kind of dependency. Denotes that selecting the source requires selecting the target as well.

### Excludes

A simple constraint between two Choices. Implemented as a new kind of dependency. Denotes that selecting the source forbids the target from being selected.

### VariabilityConstraintExpression

Implemented as a new kind of constraint. Allows to specify (in the constraint's Specification field) a logical condition which must occur. For example, referring to the choices in the figure above, we can write the cross-tree constraint "LeftACPower.APU || RightACPower.APU" to denote that at least one of the two APUs must be selected.

# Objectives and Algebras

Since every optimization needs objectives, a method was required to define them and the metrics that the objectives are defined over. In the course of the development we have started with the most simple metrics of Cost and Weight, their simplicity coming from the fact that they are a basic sum function over a set of single type parameters.

## Algebras

The purpose of algebras is the computation of certain values in the model that are needed for the analysis or the optimization process. These values are marked with the «derived» stereotype to indicate that a computation is required.

The definition of the computation may be done using several approaches. The first and a more immediate one is a textual definition of the computation formulae using some existing syntax. In our process, since the model is transformed into OPL code, the straightforward approach is to write the formulae in OPL. But since the automatic transformation of this part was not done yet, the representative syntax in Figure 39 is Modelica.

A complication of the above approach, introduced by the Concise Modeling profile, is the fact that at the time of the modeling we are not aware which model elements are present in the final model (which will be available after the optimization process).

For example, it is not possible to specify a weight formula – “The weight of a component is the sum of the weights of all its parts” – without explicitly referencing each part, e.g. weight = part1.weight + part2.weight + part3.weight…

To address this we have come up with a concept of Model Interrogation functions. In Modelica syntax these functions will be used in the following manner:

SystemWeight = sum (getAllParts().weight)

The function will be evaluated in the optimization process during selection of feasible solutions and used (in this case) as one of the elements of the multi-objective criteria.

An additional concept required when using Model Interrogation functions is the specification of their scope. For each such function we would like to specify the actual collection of elements that will be used in its evaluation – whether it is the entire system or a subsystem or perhaps the contents of a specific SysML diagram. Example of such scoping limitation is:

PrimarySystemCost = sum ((x.cost) for x in getAllParts() : x in getDiagramElements(“Primary”))



Figure 39 - Textual Approach in Algebras Definition

An alternative way of specification is to use SysML parametric diagrams as depicted in Figure 40. This approach may still require scoping limitations. The constraint may again be written in any suitable language (OPL, OCL, CSL).



Figure 40 - Parametric Diagram as Algebra Definition

## Objectives

Elements marked with «objective» stereotype will be used in the multi-objective optimization to generate the efficient frontier. In the current project a true multi-objective optimization was not done, but rather a weighted objective function was utilized.