ParaMagic 16.9 and MagicDraw 17.0 for SysML Parametrics:

Aug. 24, 2011

ParaMagic is a parametric solver and integrator plugin for MagicDraw SysML models. By modeling property relationships with constraint blocks and assigning input values to various properties, ParaMagic uses a solver to calculate unknown property values satisfying the constraint equations. It subsequently integrates with MagicDraw by populating the instance diagram with the property values calculated from the solver. It currently allows the parametric solver to be wrapped around the tools Mathematica, OpenModelica, MATLAB and Simulink. Overall, the use of ParaMagic in MagicDraw to populate property values in an instance diagram requires a significant amount of configuration and rigidity in the structure of the block definition diagrams. While this may seem like a downside to using ParaMagic, it actually enforces a level of completeness that should likely be required for the archetype models in the CML. This document should help identify the steps necessary for constructing a parametric model capable of successfully cooperating with ParaMagic.

Documentation: {MagicDraw directory}/manual/ParaMagic/User\_Guide.pdf

High-level steps:

1. First design block definition diagram, outlining necessary blocks, value properties, part properties and relationships
2. Create the constraint blocks for the equations needed for completing the parametrics model
3. Create the parametric diagrams for blocks in the block definition diagram
4. Create instance diagram and run

Blocks:

* Properties used in parametric equations must exist as type ‘value property’
* All value properties must have a name and type
  + If creating a new value type – the type must inherit ‘Real’
  + The value type must also have a ‘quantity kind’ (such as speed, force, energy, etc.) specified.
* Any referenced block, or part, must be named

Constraint Blocks:

* Only 1 output / 1 equation may be specified in a constraint block
* The output variable and all input variables must be specified as parameters
* All parameters must have a type specification (which may be one the user creates, as long as it adheres to the rules outlined above)
* The constraint equation must be entered as a constraint and must have a name
* The constraint equation may take the form of “A\_out = A\_in \* 0.1”, etc.

Parametric Diagrams:

* All constraint blocks used must be named
* Constraint blocks may not be directly connected together (output of one constraint equation may not be directly connected to the input of another)
  + All inputs and outputs of constraint blocks must be connected to a value property
  + This means that additional value properties must be specified, named, and appropriately typed in order to connect constraint blocks together
* MagicDraw enforces type consistency: all inputs / outputs must be type matched
* ParaMagic only allows a ‘crossing depth’ of 1 – meaning it can only cross into a block and reference a value property once (for instance, you cannot reference a value property such as Vehicle.traction.speed as this is of depth 2) There are ways around this, and it is detailed in the ParaMagic documentation (location shown above). This is true primarily for the support with multiplicity (as digging into further depths requires complex, non-linear solutions)

Associations / Compositions:

* ParaMagic only allows compositions and uni-directional associations between blocks
* Association/composition ends must be named
* Block associations have been demonstrated to work - allowing the blocks to reference properties on the block association without causing validation issues – but requires tweaking [still investigating, talk to Marty Fahey ([joseph.fahey@baesystems.com](mailto:joseph.fahey@baesystems.com)) with questions]
* Any unused association and composition links must be deleted

Multiplicity:

* Associations and Compositions with multiplicity are supported only with the use of a Mathematica solver backend (licensed software).
* Common functions used with multiplicity are sum(), average(), etc. [A library of functions available for handling multiplicity is documented in the ParaMagic documentation (location shown above) under section 5]

Instance Blocks:

* All instance blocks must be named
* Value properties must be initialized to either an input value or empty (an empty string)
* Decimal input values must use a leading 0 (such as 0.1)
* Any value property with an input value specific must have the type ‘given’ under causality in slot tags. This allows ParaMagic to discern what it needs to calculate from what it’s given.
* Value properties that are goals for the parametric solver must have the type ‘target’ under causality in slot tags.
* All part properties, value properties, etc. must be specified and named

More notes:

Pretty much everything needs to be named. This is a good thing to enforce, but requires a significant amount of time with complex models.

To operate the plugin, a ‘root block’ must be specified. This block essentially is the top level block and can essentially get to every other relevant block through composition or uni-directional associations. This block must be placed within a package. A CXS\_Heading must be created for this root node, which can be accomplished by simply accessing the ParaMagic context menu from right clicking on your node of interest in the package browser.

The instance diagram must be placed in its own package. In addition, a CXI\_Heading must be created for this package (and can be done in the same manner as creating the CXS\_Heading, but right clicking on the package containing the instance diagram).

Uni-directional associations - This is probably the biggest limitation of the plugin. It puts strain on the system to structure components hierarchically. While it doesn’t matter how you input and output values in the instance diagrams (since it can work in both directions), it does require you to create block definition diagrams that adhere to your parametrics model, which is less than ideal. A new edition of this software is coming out very soon and has a bunch of new features (according to an engineer I spoke with), so I’d be interested to see if it handles bi-directional associations.

ParaMagic has a handy ‘validate’ feature that may be accessed from the context menu of the package. The validate feature checks to make sure the model respects many of the requirements outlined above. In the event of a failed validation, it will pop up an error message that is often times very handy for quickly remedying the issue. Common error messages that may be seen include:

ERROR: SysML block structure is invalid. The specification in "thermal\_system\_power = FanRule \* engine\_power" in "IFV Reference Architecture::Reference Architecture Components::GroundVehicle::PowerTrain::PowertrainConstraints::Engine Thermal System Power::power::" is an invalid expression. Problem in connecting parameter "engine\_power" of the constraint property "IFV Reference Architecture::Reference Architecture Components::GroundVehicle::PowerTrain::Engine::thermalPowerEqn" to value properties.

– meaning the parameter ‘engine\_power’ in the block constraint is not connected to anything. Very straight forward.

On occasion, however, the error message may be convoluted and difficult to track down. I have witnessed that even using bi-direction associations can cause the plugin to throw an error and crash all together, requiring a re-start. Hopefully, this issue will be fixed in the next release.

ParaMagic allows feedback in constraint equations. This is a helpful feature as it is capable of solving more complex systems of equations modeled in the parametric diagrams.

ParaMagic also allows support for calling external matlab functions and scripts. A special syntax is available in the documentation describing how one can imbed an external matlab call from within a constraint equation. There may also be support for simulink calls, although I am unfamiliar with this.