75 Fifth Street NW, Suite 312
Atlanta, GA 30308, USA
voice: +1-404-592-6897
web: www.Intercax.com
email: info@Intercax.com

Dr. Dirk Zwemer, Intercax LLC

Technote: SysML Parametrics for Roll-up Calculations

Abstract

Four approaches to adding roll-up calculations to SysML models using Intercax parametric solvers are demonstrated.

Introduction

SysML parametrics are a way to add spreadsheet-like analysis to descriptive architectural diagrams. Roll-up calculations are among the most common use cases in systems engineering. Starting with a multi-level bill of materials or master equipment list, modelers want to calculate total mass, total cost, or any system metric in general, based on the individual values of all the parts in the system.

This Technical Note describes a number of approaches that can accomplish these goals using Intercax parametric solvers

- ParaMagic® for MagicDraw
- Melody™ for Rational Rhapsody
- Solvea™ for Enterprise Architect
- ParaSolver™ for PTC Integrity Modeler

These approaches take advantage of SysML's inheritance, multiplicity, redefinition, and recursion features, without eliminating the individuality of each component and assembly.

We step through a series of methods with progressively increasing complexity and versatility.

1. The base level approach simply puts a constraint like $a = b + c + d$ inside each assembly, adding the values of each of the assembly's three parts.
2. The first step in adding flexibility is to use a constraint such as a = sum(b) to add the values for all the parts. Here, *b* is an array of *n* parts where *n* is not known. This requires that all parts have a common supertype.
3. Making the constraint a property of the supertype reduces the number of parametric relations necessary for the roll-up down to one by using redefinition and recursion.
4. The final example treats the case where multiple roll-ups are needed, but not all over the same set of parts. Multi-level inheritance trees can enable this.

This exercise assumes that the reader is familiar with SysML structure and parametric diagrams and the operation of parametric solvers. For those needing more background, the user guides and tutorials for Intercax parametric solvers are a good place to start. More general background is available in several published reference works on SysML modeling.

Not all calculations in systems engineering are complex, or require a powerful and expensive simulation tool. Roll-ups, unit conversion, and requirements verification are all valuable time-savers when embedded within the model. As SysML models seek interoperability with other engineering tools, translating between data models becomes a critical part of the process. Parametric solvers are becoming an important part of the system engineer's toolbox.

Strategy 1 – Quantity-Specific Constraints

A simplified vehicle assembly tree is shown in Figure 1. Our objective is to add up the masses of the individual components and subassemblies to find the total mass of the vehicle. This calculation will be applied to the instance of the vehicle model in Figure 2.

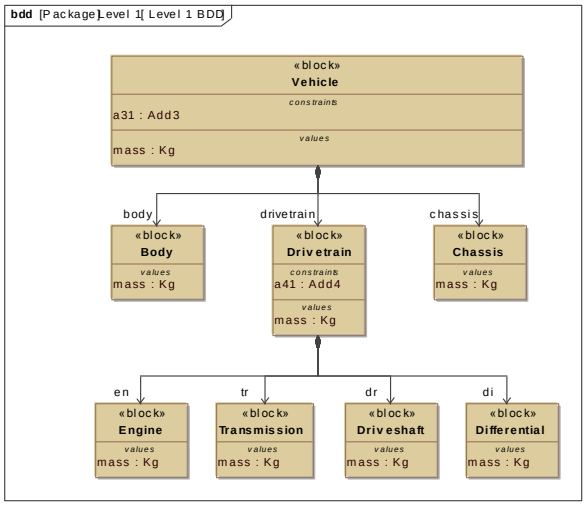

Figure 1 Vehicle Assembly Tree

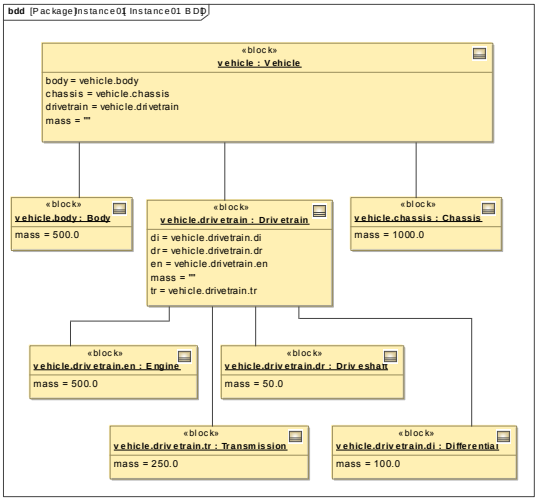

Figure 2 Instance of Vehicle Assembly Tree

The *Vehicle* and *Drivetrain* blocks contain parametric diagrams, shown in Figure 3 and Figure 4. Note the two constraints (in green) are different; one contains three terms in the sum and the other four. If an additional component is added to either the vehicle or drivetrain, it would be necessary to edit the constraint block and parametric diagram to include it in the roll-up calculation.

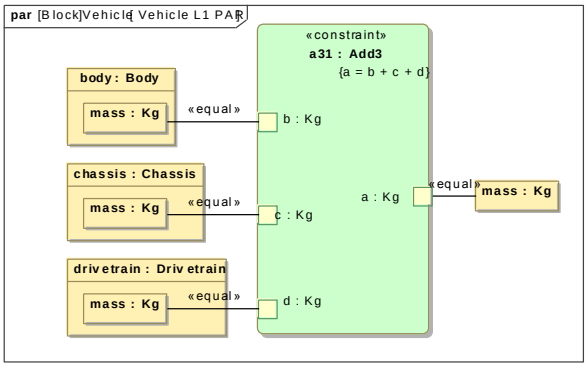

Figure 3 Vehicle Parametric Diagram

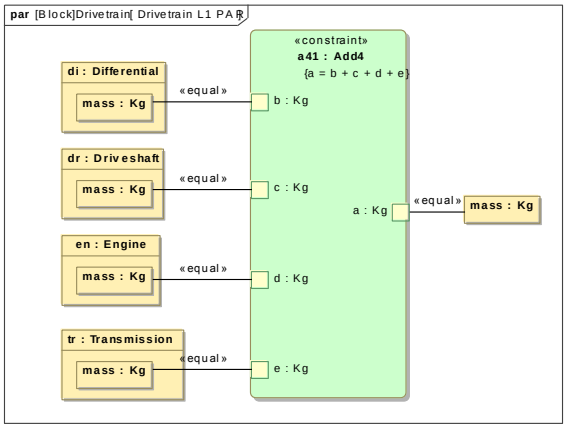

Figure 4 Drivetrain Parametric Diagram

However, this approach will work for the existing model. In Figure 5 below, the ParaMagic® browser shows the problem before solution. The mass of the *Vehicle* block is set with *target* causality and the *Drivetrain* subassembly mass is also an unknown with *undefined* causality. After solution (Figure 6), a total mass of 2400 kilograms is obtained. The mass of any component could be changed, either in the browser or the instance model, and the problem re-solved without changing the parametric models.

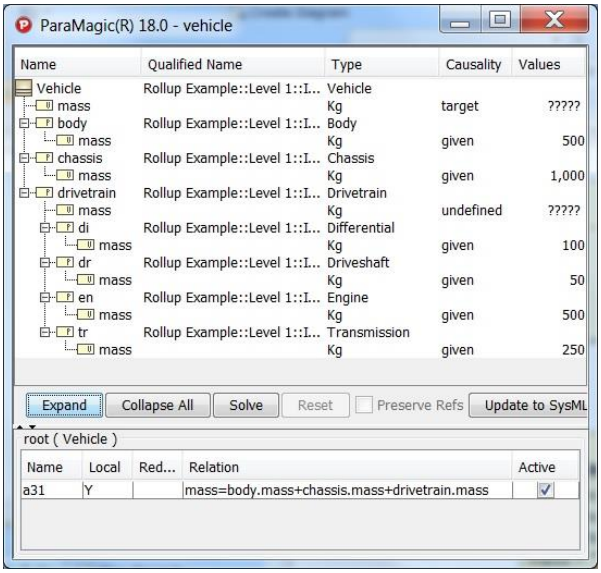

Figure 5 ParaMagic Browser before solution

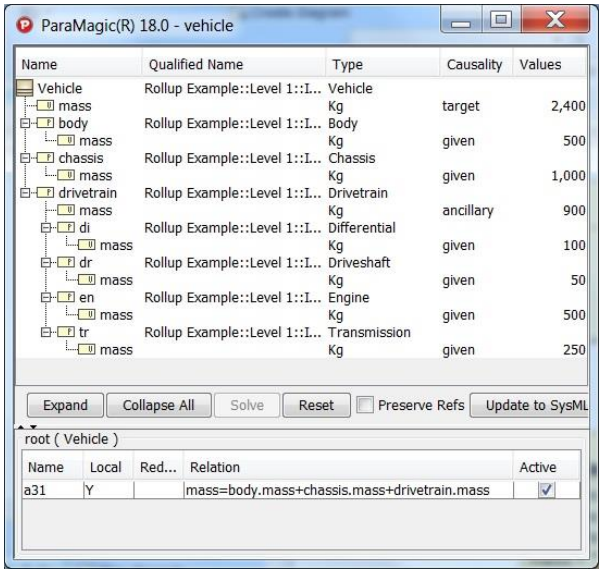

Figure 6 ParaMagic Browser after solution

At this point, it is worth mentioning an alternate approach that could be used, but with significant drawbacks. Rather than creating a separate block for each type of component, a single block, e.g. *Component*, is created and the individual parts are differentiated only at the instance level, e.g. *engine:Component*. Each assembly is composed of 1..* Components and the calculation is simply a sum over 1..* mass values. Additional parts can be added to the vehicle without changing the parametric model.

The drawback to this approach is that the interconnections between *Component*s cannot be shown in a SysML Internal Block Diagram. An IBD cannot be created at the instance level, so a diagram like Figure 7 could not be created. However, the next section will describe an extension to this approach without the same drawback.

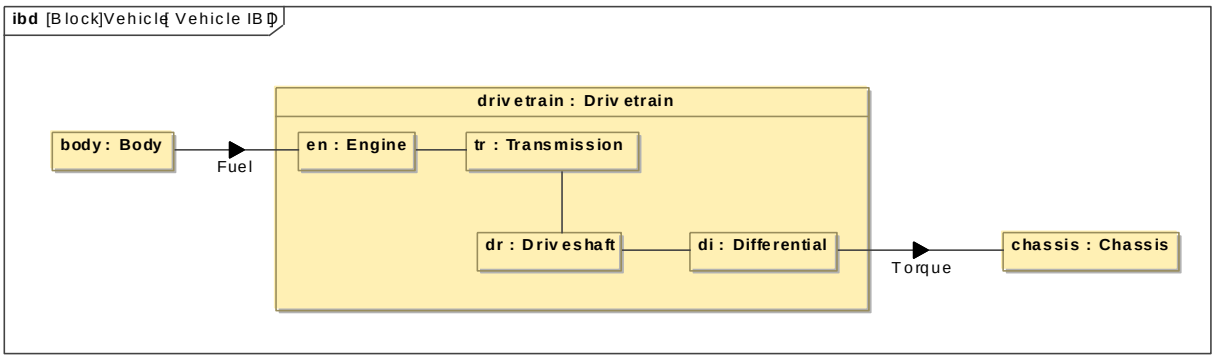

Figure 7 Vehicle Interconnections in SysML Internal Block Diagram

Strategy 2 – Complex Aggregates

Our second approach starts with defining a *Component* block, with *mass* as a value property. All the *Vehicle* parts and assemblies are subtypes of *Component* and inherit *mass*, as shown in Figure 8. The assemblies in the tree, *Vehicle* and *Drivetrain*, are each assigned 1..* shared properties of type *Component*, as shown at the bottom of the same figure. The main structure diagram (Figure 9) will be similar to the original in Figure 1 because all the part properties retain their individual identities and an IBD such as Figure 7 can be created. However, the shared properties typed by *Component* will be used in the roll-up calculations.

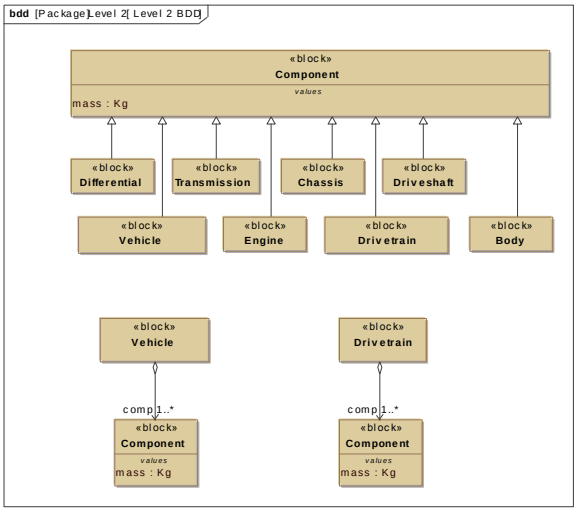

Figure 8 Adding Component to Vehicle Model

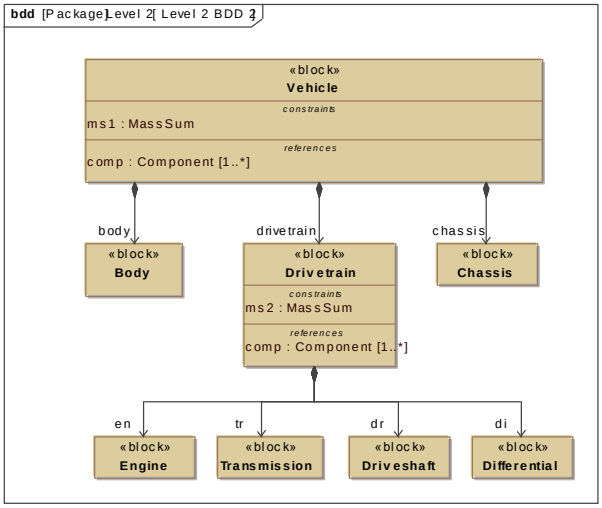

Figure 9 Modified Vehicle Assembly Tree

The *Vehicle* and *Drivetrain* blocks still contain parametric diagrams, shown in Figure 10 and Figure 11, but the two constraint properties (in green) are usages of the same constraint block,

MassSum. The constraint in *MassSum*, a = sum(b), is independent of the number of *Component*s whose mass is being summed over. Such calculations are called complex aggregates.

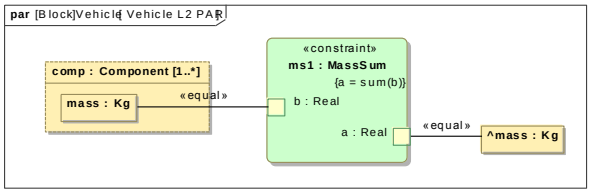

Figure 10 Vehicle Parametric Diagram

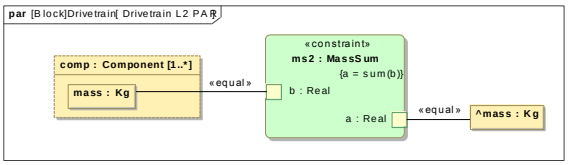

Figure 11 Drivetrain Parametric Diagram

The additional overhead in this approach arises in building the instance model. Each instance of a part, e.g. *vehicle.body:Body*, plays two roles in the next assembly up, as a specific part property, e.g. *body:Body*, and as a shared property, comp:*Component (1..*)*. In MagicDraw, this is accomplished by opening the specification window for the assembly instance, here *vehicle:Vehicle*, assigning *vehicle.body:Body* to the slot *body* and to the slot *comp.* In the second case (Figure 12), *comp* also includes *vehicle.chassis* and *vehicle.drivetrain*, the other direct parts of *Vehicle*. As shown in Figure 13, the same pattern follows at the *vehicle.drivetrain* level, where its four parts play double roles, as individual parts and as generic components. The fact that the same instance is playing a double role is why we use shared property relationships for the components rather than a second set of part property relationships.

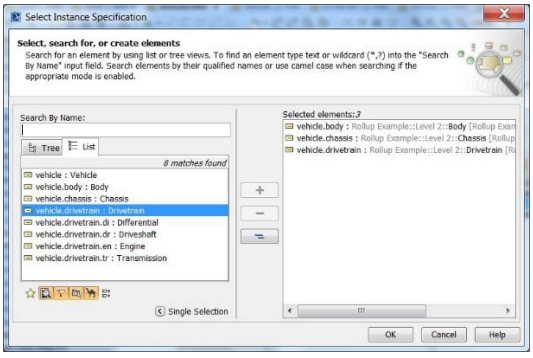

Figure 12 Assigning instance to shared properties

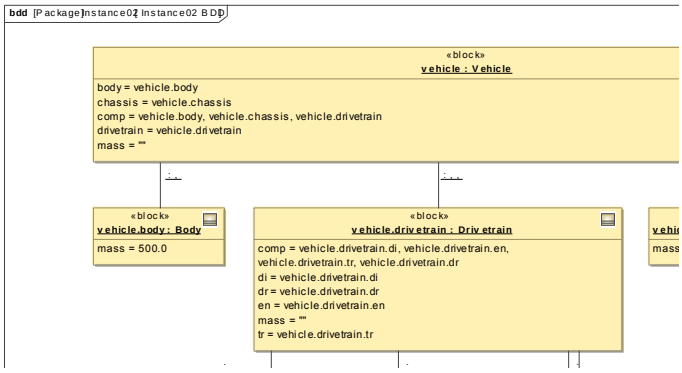

Figure 13 Detail from Instance diagram

The parametric browser appearance in this situation is very similar to Figure 5, except for a double appearance of the part instance values, and the results of the calculation are the same. At the same time, the IBD in Figure 7 can be created without problem.

Strategy 3 – Complex Aggregates and Recursion

Even greater economies of modeling effort can be enjoyed by using recursion. Noting that the same constraint block is used in both the *Drivetrain* and *Vehicle* assemblies (and any other assemblies that would be created), we can choose to build that parametric diagram at the *Component* level (Figure 14), where it is inherited by *Drivetrain*, *Vehicle* and all the other structural elements.

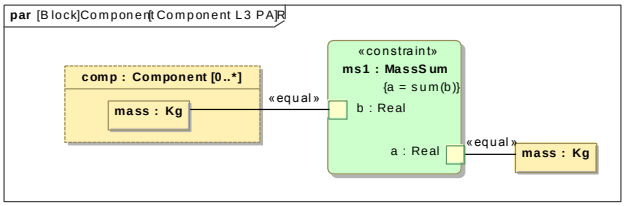

Figure 14 Component Parametric Model

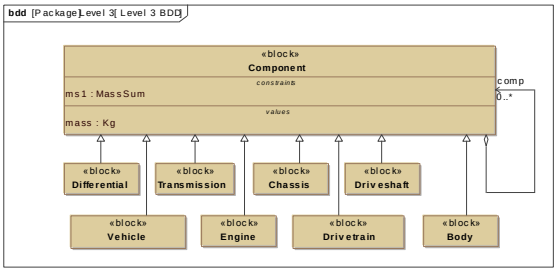

Figure 15 Component Inheritance and Recursion Relationships

To create that diagram, *Component* needs to be able to create references to the parts that make up that component. This is shown in the recursive shared property relationship on the right side of Figure 15. Note that this relationship has a 0..* multiplicity, zero applying for leaf-level components like *Engine* that have no subparts in this model.

The instance model is created as in the previous example, except that the slot *comp* remains empty for leaf-level components. The results of the roll-up calculation are the same. New parts and new levels of parts can be added to the structure hierarchy with no changes to the schema except

- Creating new blocks to represent those parts, and
- Making those new blocks specializations of *Component*

Strategy 4 – Multiple Inheritance

Adding a second set of roll-up calculations, e.g. rolling up cost as well as mass, is often straightforward. Adding *cost:$* as a value property to *Component* in Figure 18 and adding a second constraint property *cs1:CostSum* to the parametric diagram in Figure 14 would be all that was needed. The problem arises if the two calculations apply to different sets of parts, e.g. there are software components with cost, but no mass.

Figure 16 shows how to set up such a problem. Another level is added to the inheritance hierarchy, *Supercomponent*, with a value property *cost*. Two new blocks, *Engine SW* and *Cabin SW*, have been added to the model representing software with cost, but no mass. They inherit directly from *Supercomponent*. The original parts inherit from *Component*, which inherits from *Supercomponent*. *Component* retains the same parametric diagram (Figure 17), while *Supercomponent* shows recursion on itself and the cost calculation parametrics in Figure 18.

Building the instance model is similar to the previous two examples, but now instances of the original parts may fill three roles in an assembly, e.g.

- As a part property of type *Engine*
- As a shared property of type *Component* (1..*)
- As a shared property of type *Supercomponent* (1..*)

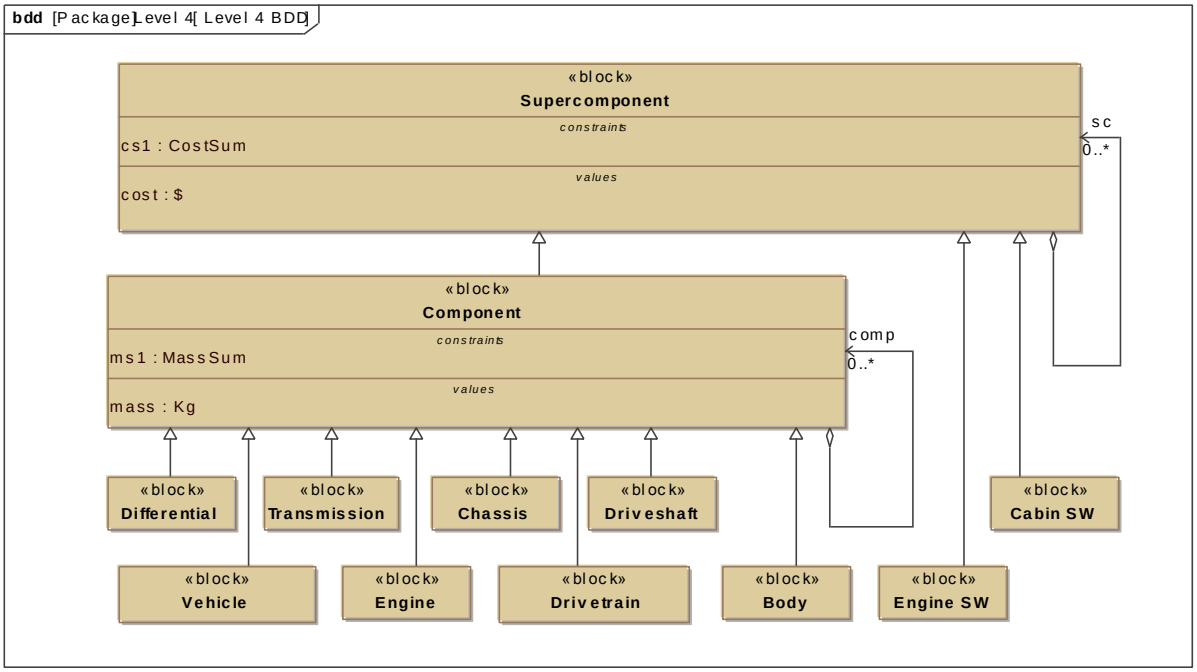

Figure 16 Two levels of Inheritance for Roll-up Calculations

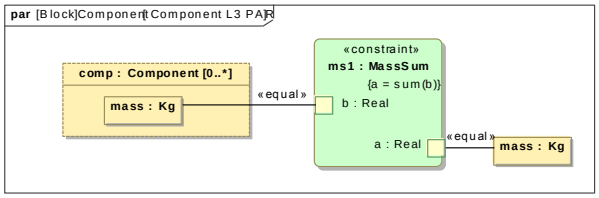

Figure 17 Component Parametric Model

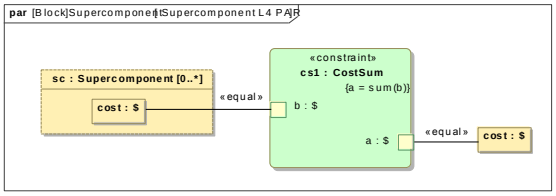

Figure 18 Supercomponent Parametric Model

These same general approaches can be extended to additional roll-up calculations, including additional layers of inheritance and multiple inheritance. It is important to evaluate the parametric solver tool capabilities to handle inheritance, recursion and complex aggregates for the class of problems the modeler needs to solve.

About the Author

Dr. Dirk Zwemer (dirk.zwemer@intercax.com) is President of Intercax LLC, Atlanta, GA and holds OCSMP certification as Model Builder - Advanced.

For further information, visit us at www.intercax.com or contact us at info@intercax.com.