A software to display Sysml graphicaly

Tommy Andersson : anetom-6@student.ltu.se

July 31, 2020

1 Introduction

This will be a document where I have documented my experience with Sysml. First there will be some explaining about Sysml and instruction how to use the basic code for Sysml. After that the user cases that I've worked with and how they worked or not worked. Lastly will be how some features are implemented, ideas and thoughts for those features who's not implemented yet and ideas how the graphical user interface could look like.

From here on the the system will be called simply the "software" since there is no name for it.

Jesper Nilson is the creator of the software and I have simply tweaked and added functionality. I got his latest version from him 12/6 which had some modifications from the previous version.

If you would like to look at the code it's on my Github.

2 SysML the short version

SysML is a tool for system engineers to visualize all the components in a system, they are used to show stakeholders but also for the people working on the project how everything should fit together. SysML shows both how the system is connected but also, how it should be used, in what order things happens etc etc. In this part I will focus on BDD and IBD.

The most used block is the block that describes an object. This object can be connected with other blocks and can show how the chain works. The block can have attributes to it such as values, parts etc. It can also just be described as a block with a name. The second most used block is the part block. This block shows that it's a part of something, it can show that it's a part och a block but also a part of a part. A part needs to be defined with a type, this type must either be the same as the name of the part or be a block. If wanted you can assign a number to the part with [number].

2.1 Subsetting and redefinition

Subsetting is to give a part the same type as a different part, that part can then have two types of types. This also works for block as to give a block the same type as a part.

You can redefine a block with another block and it will then accept the types the first block had. You can then redefine the part inside the block and give it different values, this is to specialize the block with specialized type.

3 Eclipse and plantUML

Bellow is a short summary and observations of the different blocks in Eclipse PlantUML.

3.1 PlantUML

The diagram is structured in a certain way and this subsection will explain what everything means. In the package there is squares with different names and a symbol the left of the name. So far I've seen "B", "P", "T", "F" which says what type that square is. "B" is a block, "P" is part, "T" is value type and "F" I don't know yet. You can have packages in packages but you can't have two "outer" packages in a file. For example package x (package y()) is ok but not package x () package y().

3.2 B blocks

Is one of the most used block in the diagrams. It is used to describe an object. When going to another block it's a line with a hollow arrowhead attached to the child block. When going to a part block it will be a line with a diamond attached to it.

Depending of the start node and end node the arrows will look different. When an arrow goes from a block to a part it will be a line with a diamond attached at the end at the parent. The same is applied when there is a connection between two parts. When going from a part to a block the arrow will be dotted with a hollow arrowhead at the child. The connection between two block will have a filled arrow with a hollow arrowhead.

3.3 P blocks

Is also used often, describes that it's a part of something. A block can be a part of another block. A part can also be a part of a part. When a part is connected to another part it's a line with a diamond attached to the parent part. From what I've seen if a part introduce a new value there will be a dotted line with a hollow arrowhead. (This can be wrong, needs to check more of it)

3.4 T blocks

The "T" square can be a value definition block but also be used to show some sort of action, in the example "$VehicleModel_evs_update.sysml$", "provided power", in-value is fuel and out comes torque. This describes what the action is. You can use the port command also and get different "T" square and then you can just define the in value.

Value type and value is not the same. Value type can be referenced but not value.

3.5 F blocks

What I have studied right now is "Wheel package" in sysml/src/examples/v1 Spec Examples/8.4.1 Wheel Hub Asssembly. There I noticed that the "P" and "B" in the blocks stands for "Parts" and "Blocks". If a block is part of something it will have a dotted line and if a part is a part of something the it will have a line with a diamond from it's parent. From this example when a block is a part, the block is defined first. When a connection between to blocks the connection is a line with a hollow arrowhead. "T" is value type, can't say what's different between it and just value. "F" is when something is redefined or if it's a requirement.

4 Introduction to Sysml commands

To start a package is needed to encapsulate the file like a class in Java.

```
<< package name_of_the_file {
        All the code is between these brackets.
} >>
```

A package can contain another package but a file can't contain 2 different packages on the top level. When the package has been declared all the code of the sysml file can be written.

There are different ways to define blocks, parts, value type etc. One way is to define blocks at the beginning of the package, this is often used when defining parts who will be used later. It will look like this:

```
<< block Name_of_the_block; >>
```

Another way is to write the block and then it's content, remember if parts is to be declared in the block they need to be defined before. It is the part_type of the part that must have the same name as the block. It can look something like this:

```
<< block Part_type;

block Name_of_the_block {
    part : Part_type[amount];
} >>
```

Blocks are supposed be able to be declared as public or private. This has not been implemented yet since other functions felt more urgent.

Part can also be used in the package and in other parts. When declaring parts of part it's used as block except the part_type also needs to be declared. When using parts instead of blocks:

```
<< part eng : Engine {
    part cyl : Cylinder[4];

    part oil_tank : Oil_tank[1];
} >>
```

From the examples in the notes [1] 3 iterations of parts has been the maximum number of iterations. So far 3 iterations have been tested and works, more iterations should work but haven't been tested yet. There exists both value and value type, value type encapsulate the value. Typical a value type contain one or more values where value only contain one. One example is:

```
<< value type mass {
    value kg : mass;
    value g : mass;
} >>
```

Value and value type can be used inside the package and in blocks but value can be also be used inside parts. When writing values they need to either be defined as a value type or if the values are imported from another file. Imports does not work at the moment. It works as when importing libraries to software. More about imports will come later in this document.

```
<< package name_of_the_file {
    import ISQ;
} >>
```

Specialize or ":>" can be used on blocks for it to take on the same blocks/parts/properties as an abstract block or normal block, a block can specializes more than one block. The abstract block or normal block needs to be declared before it can be specialized/subseted.

```
<< abstract block abstract_block_name;

    block new_block :> abstract_block_name; >>
```

When using specialize/subsets redefines can be used to inherit parts from that block. It's a special command which can be used to either inherit the part type or change the amount.

```
<< block Car {
    part eng : Engine;
}
```

```
block Engine {
    part cyl : Cylinder[1..10];
}

block Small_car :> car {
    part small_engine : Small_engine redefines eng;
}
block Small_engine :> Engine {
    part :>> cyl[2];
} >>
```

It can get a bit messy understanding the content but in short. First a car is made with an engine as a part. Then a block named engine is created with cylinders made as parts. This part is given a lower bound number and a upper bound number, it can be give a static number. When this is complete a very simple generic car is made. We want to make a small car with nearly the same content but with some modification for example a smaller engine. The smaller car subsets the car, the smaller in turn subsets the engine and then redefines the cylinder but with only 2 cylinders instead of the generic 1 to 10. Note that redefines and ":>>" is the same command and either works.

Assoc block is used to connect different parts with each other. It will act as a bridge. The command is:

```
<< assoc block name_of_the_block_assoc_block {
    end : A_deeper_part[amount];
    end : The_second_part[amount];
} >>
```

Assoc block is then used inside a block/part to link different parts with each other. The command for this is link, since it can be a bit confusing a whole example is written.

```
<< part example_part : Example_part[amount] {
    part_one : Part_one[amount] {
        tfp : The_first_part[amount] {
            adp : A_deeper_part[amount]};
    }

    part_two : Part_two[amount] {
        tsp : The_second_part[amount];
    }

    link : name_of_the_block_assoc_block connect tfp::adp to part_two::tsp;

    } >>
```

The "::" is used as a search way and the part before the desired part should be before "::" and then the desired part. Note that in the assoc block the type is

written at the *end* command while in *link* the name of the part is written. This feature hasn't been tested to 100 %.

Port hasn't been implemented yet but here's an example from the Introduction to the Sysml 2.0.

```
<< port def FuelOutPort {
        value temperature : Temp;
        ref out fuel_supply : Fuel;
        ref in fuel_return : Fuel
} >>
```

5 Usercases what's working and what's not

5.1 Home

Home works except the new defined names on both the "objects" and the name of the lengths.

Made a new files called home works. Had to remove value ¡value¿ because the system crash. Can't enter values can't be entered since both systems doesn't accept the other program's solution. Changed the type name so they have the same name. link have the same problem as before.

Update 12/6: Changed the value to "value local_IP : IPAddress;" and made a block called IPAddress. No soft error but need to check if it shows correct paths.

5.2 Wheel package

The package works with the "software" but when I don't import ISQ I get soft errors on some of the values. New defined types still returns soft errors. When I import ISQ, the "software" crash when reading the file. Needs to check if a string can be a value without getting a soft error.

Update 12/6: Didn't crash when I wrote "value inflationPreassure : 'Pressure';", still gives a soft error since that unit hasn't been declared.

5.3 Usecase_6

PlantUML have problem showing special char as "&" and Swedish letter like "å,ä,ö". When I have more than one package in the same file PlantUML says "missing EOF at 'package'" at the next package. When splitting it up the error goes away but then I have problem with the links in the file I named "Main". The "software crash when I try to load that file but the other files works fine. Need to check if I can change the links so they work.

5.4 dist_heating

You can't have two package in the same file before you get a soft error, sort of make sense. Put the package in different files and it worked wonders. It didn't

even give soft errors on the name for some reason. Still same soft error when assigning values.

"part single_family_houses : 'DH family house'::'Single family houses'[1];" the part after "house :" seems to be the link from one package to another.

5.4.1 Status of user cases

Further tests with user cases has been scraped since the main mission is to get it working with Eclipse and Plant UML. Only the Wheel_package has been used lately and a new one created called Computer is frequently used. These cases will remain in this document to see how far I've come with the "software".

6 Questions

A value who haven't been assigned a number, should it be unassigned or given a number? Right now they are given 1 for amount if it's not specified. The upper bound is left unassigned if now number is given.

Should all parts be shown in the diagram? Can get messy.

7 Ideas and problems with the software

7.1 Imports

For the "software" to work imports aren't necessary but it helps a lot. Instead of declaring known value and value types in every sysml file a simple import will handle that. These files are also checked so they are correct and will also decrease errors.

7.1.1 Implementation

The idea at first was to make a local textfile and every time the file needed to be imported it was going to read from it. This didn't work since javascript for security reasons doesn't allow reading from local files without you the user to accept which files who should be read. This would be meaningless because the user would need to know what to include in the file. The new approach is to upload these files on a server and then the "software" should be able search for the right file, read that file and save all the information in that file and the proceed with the current file. Javascript should be able to handle the search for the correct file as long as the path to the directory is set. If the current parser of the "software" can handle it is unsure but if not a tweak should only be needed.

The only import that works right now is ISQ and reason for that is to avoid the "software" to crash when it reads that line. This also helps when reading example files from the Eclipse Sysml plugin since many of those examples contain ISQ.

7.2 Special characters

Swedish letters can't be used without leaving a soft error and at risk of losing letters after it. Since this a problem at Eclipe and PlantUML there is not much to do but can be worth to mention.

7.3 Navigation

As of now there exist a back button which work, the code used for it to work is almost the same as the draw function and could be better written so duplicate code could be removed. For even easier navigation a home button could be place so the user could go back to the homepage. This should not reset the current settings of the depth level.

The sliders for depth needs to be worked on since they don't do anything. Not sure but this can maybe be fixed by checking the depth of the objects and only draw the ones in the interval.

Change so that the canvas is drawn automatically when the users choose a block in the drop down menu.

7.4 Value and value types

Value and value type are easy to mix up but they are different in some way. From my understanding value type can encapsulate values and other key function. For values to show correct values a value type must be made or imported. Both can exist in blocks and both can exist in packages. example is on page 70 in intro to sysml. Value types can't have other value types inside them.

At this point parts can have values inside them but not value types, this can be worked around at the moment to just put value types in the package.

7.5 Parts

As of now parts is given 1 when they are undeclared inside a block and undefined when they are created inside parts. One can also argue that the default should be 0. More time and testing is required but if a part have a part the default should be 1.

Parts can contain other parts, the reason for this is possibly to avoid chaining blocks and part, bellow is a demonstration:

```
block_b;
block_c;

block_a {
    part Block_b : block_b[1];
}

block_b {
    part Block_c : block_c[1];
```

```
}
```

It can be written like this.

```
block_b;
block_c;

block a {
    part Block_b : block_b {
        part Block_c : block_c[1]
    }
}
```

Not a big difference but it can save time if a package have many parts. Right now they are saved as different types and are separated from each other. This may be wrong as it can mess up the tree structure but it should be easy to fix so they work the same.

For some reason when clicking on parts defined in the package the user is returned to the home screen.

7.6 Port

Haven't worked with ports but from the documentation a port is defined with values and 1 reference in and 1 reference out. This is then declared inside blocks with their own name but with the same type (the type is the same name as the one used in port def).

7.7 Subsets/specialize

Subsets are implemented from my understanding. When using the subsets command a copy of that block is made with the name declared before the subsets command. This command can be used on both abstract blocks and normal blocks, a block can subsets more than one block, this doesn't work 100 % since copy mechanism doesn't handle that right now but all block who's specialized are saved in an array in the new block object.

7.8 Redefine

Redefines works for parts in a block and parts in parts. Blocks are easier to handle since much of the info is stored in the blocks and more functions in the blocks are implemented. Redefines works with both of the examples in the Sysml plugin for Eclipse but when only using parts the redefines command becomes blurry. The parts is saved but the objects that is being redefine is stored in an array in the object.

7.8.1 Redefines of parts

If a block has a block saved in the subset block array, the part wanted is searched for in that block. The name and property is then copied to the current part with the same amount or a new one if wanted.

7.8.2 Redefines of whole parts

First the active block adds a subset block and then makes a part as usually. After the amount the redefines keyword is used with a name. The name is searched for in the subset block, it then takes the part block from the subsets and stores it in an array called redefines in the current part. Right now the whole object is stored.

7.9 functions that haven't been implemented (bdd, idb)

import, public and private blocks, port, alias, comment, ref(implemented but maybe things needs update or added?),

7.10 Other functions for other diagrams

snapshot, individual, succession, timeslice, interface, stream.

7.11 Action diagrams

Is to be implemented or at least be planned for.

7.12 Sequence diagram

Is to be implemented or at least be planned for.

7.13 Current status

The core behind bdd is mostly done, it needs some tweaks but can be used as long as it's not unimplemented features. It should work for "everyday usage". ibd should have the same features as bdd.

The biggest trouble when using the "software" is the graphical representation. This will probably need an overhaul and bellow this section ideas how it can look like and some thought how it can be implemented are written.

8 Graphical representation

8.1 Current graphical representation

Blocks and parts can be viewed and connections lines goes from a parent block to it's parts. The structure is like a tree structure but not always. In the current state most people can understand how it works but usually needs some

explanation before grasping it. This is a problem since the main goal of the "software" is to make it easier to share Sysml code so user can use it directly without learn more than the basics of Sysml. .

8.2 Plan/idea for the graphical view

Blocks/parts will have a title declaring if they are used as part in a block or if they are part of a part. Most will be parts of blocks but some will be parts of parts. Blocks/parts can have different values, the picture bellow is just examples how it can be structured.

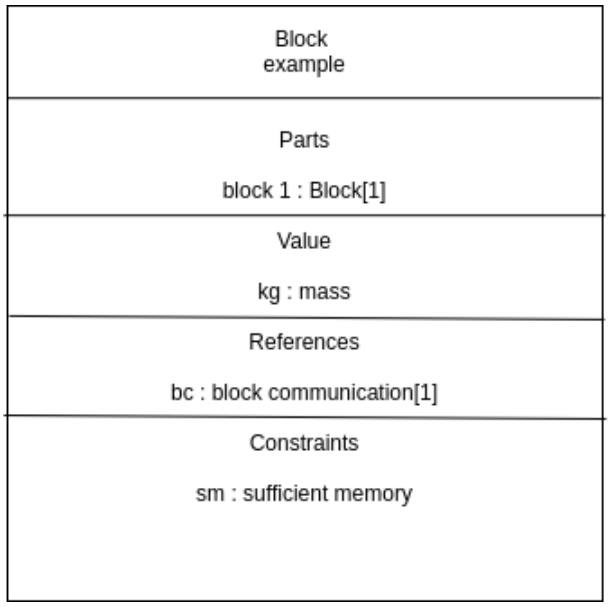

Figure 1: Concept how blocks with values and other properties can look like

The package will be placed at the top like a block but with only it's name and maybe values declared in the package, from there the structure will look like a tree (maybe rename it so it's clear that this is the package). Useful information such as value and value type should be visible in the boxes that represents the blocks and parts. More information such as constraints should also be visible when it's implemented. When pressing a box the boxed pressed should be the new root of that view and the following boxes should have a tree structure (maybe put an arrow pointing up from the root with the name of it's parent?). This can continue until the last box is in view.

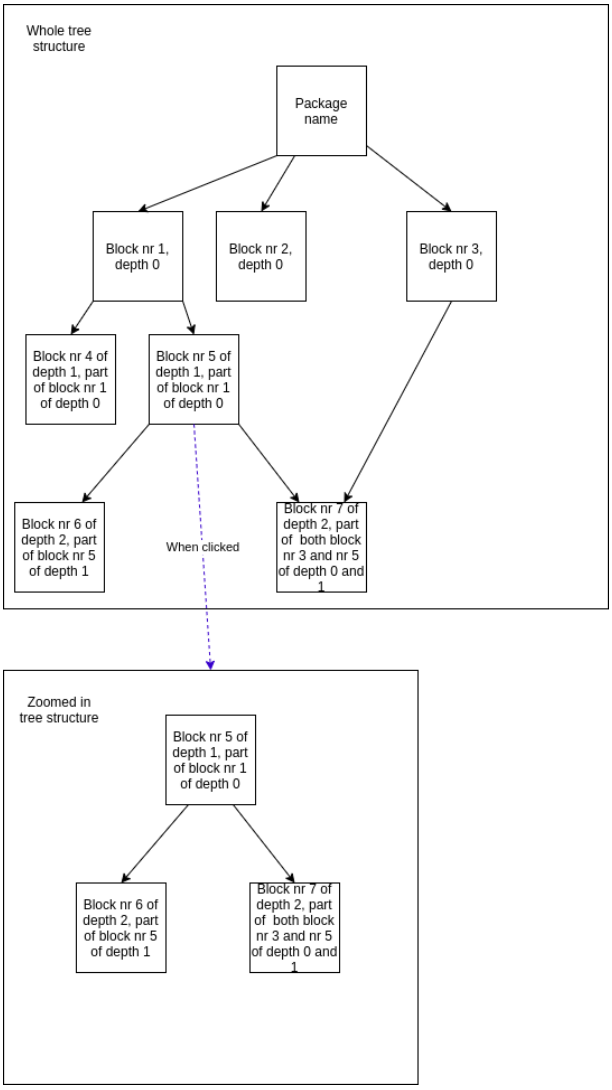

Figure 2: View of the current package

Users can press back when they want to either go back one level or home to go to the top level. The level of depth can probably be done from the property "isPart_count" in block/part.

Users should be able to set how deep they want to view with the sliders in the navigation bar. By default the whole tree should be drawn but can be set so only the tree from level 1 to 3 is drawn (level 0 is the top level), the slider can only go as high as the depth of the tree.

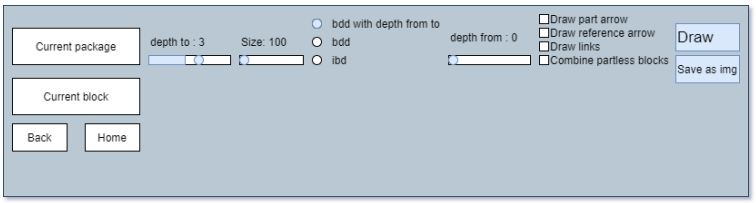

Figure 3: View of the navbar

For easier readability the lines should not be overlapping if it's not obvious where they go (if that is to hard maybe draw them in other colors).

If a block/part is specialized from another block/part it should be shown in some way. One way is to display in the box that it is a specialization from that block, it should also be a level under the block that it's specialized from.

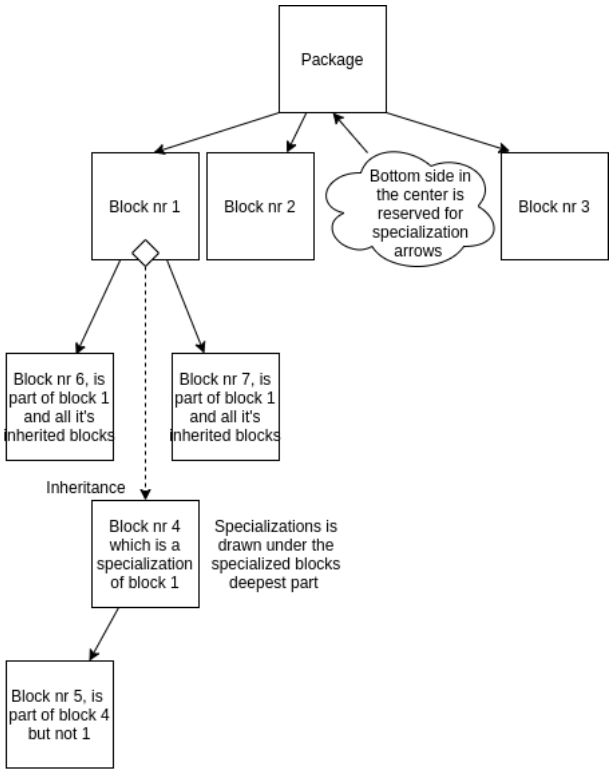

Figure 4: Concept how specialization could look like

Boxes with no parts or isn't part of any box can be put into a box called "components" or "unused boxes" to minimize to many boxes on the screen. This can be toggled with an extra option if this is wanted or not.

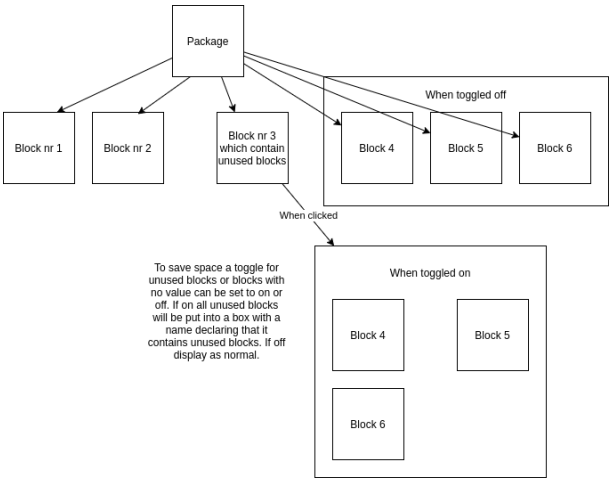

Figure 5: Concept how unused blocks could be grouped

Value type can be set to the right of the tree to avoid the view to be messy. Maybe integrate it with the boxes.

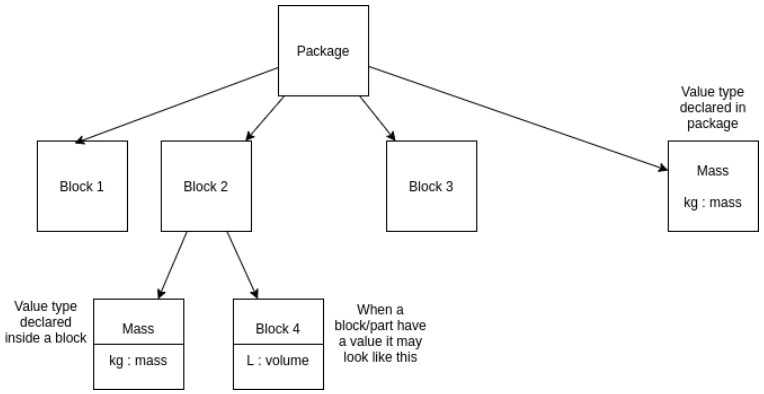

Figure 6: Concept how value types could look like

To achieve this structure the plan right now is load in all the blocks/part to see how deep the tree will be. When that is done the tree should be drawn from bottom to top to avoid that blocks who is parts to to different blocks on different levels is place on the same level or even higher than one of them. All arrows should go down unless there is a link between blocks on the same level, then they should go horizontal

References

[1] OMG. Introduction to the sysml v2 language textual notation. 2020-03.