Diagrams for UML and Function Blocks drawn with Libre Office

Dr. Hartmut Schorrig
www.vishia.org

2023-05-12

Table of Contents

Table of Figures

1 Basic idea using Libre Office for Graphical programming

As you can write any program with any desired editor, such as Notepad++, or VIM on Linux or just a powerful Integrated development environment, you may should do the same also for graphic programming.

But graphic programming has not a unified definition of content mapping to a file format. Textual programming – it is not a problem, it is basically plain text. This text is interpreted syntactically be the translators (compilers) as programming language. Sophisticated editors can show syntax highlighting, cross references etc and supports the compiling, but you can also use a simple editor to work.

This should be possible also for graphic programming.

For that a universal available graphic editor is necessary.

Libre office can substantial used for that.

TODO explain a little bit more.

Basic ideas:

- Use Style Sheets to designate semantic information to graphical blocks,
- evaluate it reading information from the odg file, it is a simple zip file containing XML information
- Translate the content to other graphic formats for the specific tool or make the own code generation.

This is possible to do with not to high effort.

2 Approaches

2.1 Question of sizes and grid snapping in diagrams

Commercial tools have often not a proper answers to this question. Often sizes are scalable in any kind, as the user want to have. Grid snapping is sometimes supported or not, and, sometimes sophisticated algorithm are implemented which avoids lines through blocks and make instead mad ways around all blocks. LibreOffice is here more friendly, it let the user decide about the connection path. This may be only a marginalia.

Let's think about font sizes and grid, requirements:

- In a usual document a proper font size is 11 pt, as written also here. A smaller font (9 pt, 6 pt) is not suitable for reading because of the recognizability of the words and their contexts, it is only for read the package leaflet of medical products.
- A diagram should have place in a document on a A4 or size-B page (~ 18 cm text width). It means the size of a proper view is **~18 x 10..12 cm**. Using a whole side in landscape orientation may have a size of 25 x 17 cm, but in landscape mode the document must be rotated only for this page, this is not suitable for reading a PDF document on the screen.
- A diagram has two tasks:

  a) Documentation

  b) Base for the software

  For the approach b) the diagram may be well editable as a whole on a large screen, for example with resolution 2650 x 1200 pixel. To document this complex diagram it can be shown in landscape orientation in a document, or better: It should be reduced in size to fit on a normal page in portrait format. Details are then no longer legible, but important things and orientation should be shown in larger font. Then the overview can be explained and details can be shown as part from exact the same diagram in a higher resolution.
- A common and contradictory question for diagrams is: How comprehensive should it be. Should it contain only one block and some less aggregated ones? Or should it contain the whole truth of a module? The answer of this question depends on the available size for presentation. There should not be to less content.

  The UML has the advantage that you can use more as one class diagrams to explain the same class in different contexts. That is a very great advantage and it should be usable also for some Function Block presentations! (Not yet in professional tools). This helps to decide how many content a diagram should contain.
- The readability of a word which is isolated of a sentence, an identifier of a block or line or such one is given also with a smaller font size than 11 pt, especially if it is present in bold font or maybe also in a non proportional font (as for programming language source code). Because in proportional fonts often important small characters such as "il" are to small and bad visible
- For positioning a proper grid size and the **possibility of positioning with cursor keys (!)** is essential. Libre office has the property that the step size for the cursor key is anytime 1 mm, independent of other settings. It's possible use cursor keys for fine positioning (Alt-Cursor...) but this is too fine.

There is a specific property of Libre office: The step width by moving with cursor keys is normally 1 mm. You can do fine adjusting in combination with the Alt-key, but this is too fine. If also a grid fine spacing with snap points of 1 mm is selected (a 5 mm grid with 5 fine divisions), then the placing is very proper. All elements are placed in a 1 mm grid, the 1 mm is enough fine for details and enough raw to simple snap in the grid points.

From that, the idea comes to have a standard size of small elements of 2 mm. The mid point is also in 1 mm grid snapping raster. You can have a near distance of lines of 1 mm, well obviously.

To show enough content in a diagram you may use an A3 paper in landscape orientation. On a larger monitor (2560 or 3280 pixel width) it is editable in entire page mode. The diagram has a width of ~40 cm. 1 mm space is ~ 6 pixel on the screen.

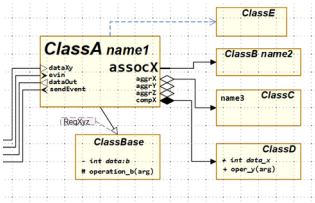

Figure 1: View 40%

If you present the whole diagram in a document in portrait format, it is demagnified to ~ 17..18 cm, it means ~40%. As you see right side, the name of `ClassA` is readable, also the "`assocX`" with a font size of 10 pt Consolas bold in the original. Here it is presented with ~ 4 pt because of the demagnification. The others or not readable, but you can recognize the aggregations, compositions and associations. The symbols may be obviously though they have a size of only 0.8 mm height.

The same content is presented here right side in original magnification. The font size of 6 pt for the most elements is just readable. It is Consolas bold. The type names of the classes are Arial 8 pt, the name of ClassA is Arial 14 pt. This is a 1:1 presentation, drawn in portrait A4 it is really 1/1 site width.

It means you can have an overview, but you don't see some details in the documentation. Parts of the same diagram can be shown in original size, then all is readable.

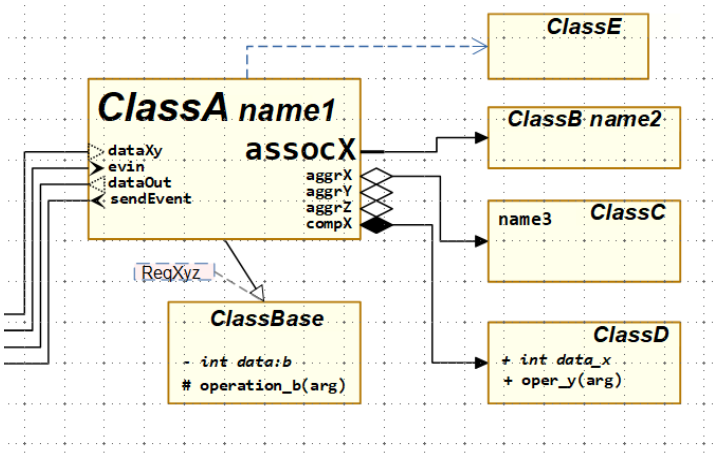

Figure 2: View 100%

You should place different approaches of the same module in more as one diagram. This is definitely supported by UML, and should also be usable for function block presentations. In commercial tools such as Simulink it is not possible, but here it is.

As living example look on the following Class-Object-diagram:

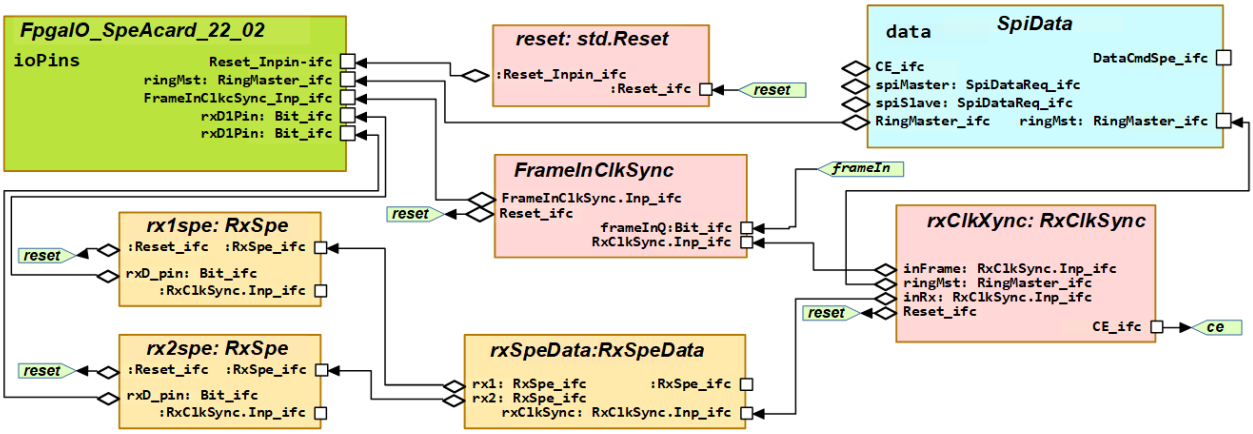

Figure 3: Example for a Module Diagram

This diagram should be well readable in normal view of a pdf viewer. The font and size of the names is consolas 6 pt bold. The original draw area is the width of a A4 page. The pixel solution is 1351 x 480, results from a Zoom of 200 % on a 1980 pixel width monitor.

The diagram shows a coherence of different blocks to build a synchronized *clock enable* (ce) in a FPGA. You see two receiver (Rx) modules, which are combined with a third module, with equal

light-brown colors. Its a selection of the active input. The output of this third module has the same interface type `RxClkSync.Inp_ifc` as the module in the mid. Both are selected from the red right module. With less explanations the coherence should be understandable.

2.2 Using figures with style sheets for elements

The original UML class diagram has the following approach:

- A class is a rectangle box containing the type name of the class.
- Some data or operations may be named inside the class box, it does not need to be completely.
- All relations to other classes are shown with references to the other classes. This references are often non directed, but sometimes only in a specific direction marked with an little arrow on end. This relations are associations, aggregations, compositions, inheritance, dependencies.

The last point is not mapped to the languages which presents the software which is presented by the UML diagrams. Because: The fact that a class has an aggregation to any other class is a property of this class, not a property of relations between this classes. It is exactly the same as for data. A data element has a type, and a reference has also a type, the type of the referenced class. It least the name of a reference is only a property of the class, it is not a property of the relation between the classes.

For that reason the shown relations to other classes are assigned to the class itself. They are existing also if there is no connection. Then, of course in the implementation it's a null or nil pointer. Or it is just not shown here in this diagram, instead shown in another diagram, but nevertheless shown as element of the class. Look on the images on the page before. There are some not connected aggregations, which may have a meaning on explanation to the diagram.

The elements are simple small figures with a fixed size, known from UML as the diamond (filled / non filled) for Composition and Aggregation, or they are simple rectangles. The elements contains a text, which is the identifier for the element or also the type specification. The text is written outside on the element itself by using the Libre Office property, that a text can exceed the bounds of the element's graphic. More as that, the left or right margin of the text is set to a value greater or equal the size of the element, and in this kind the text is written outside, left or right next to the element.

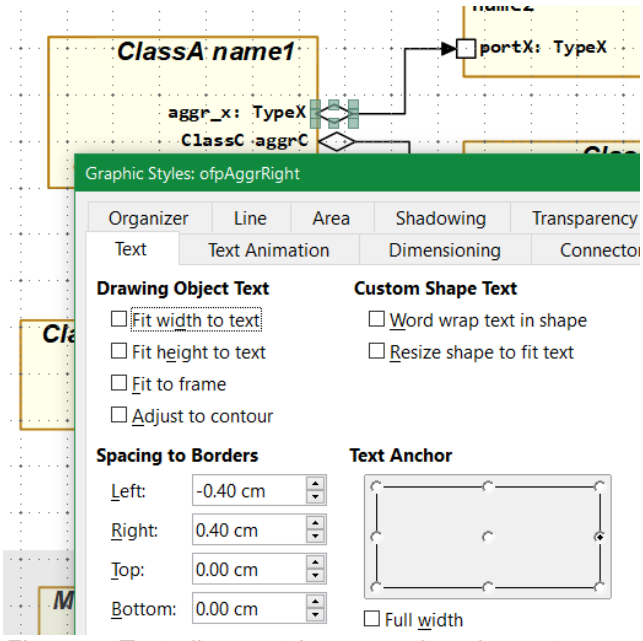

Figure 4: Text alignment in connection elements

Now, an element or more precise a connection to the class is shown as this small figure. But not the figure itself characteristics the element for code generation, instead the associated style sheet. The look of the figure can be changed, should not be changed, it is for human. But **the style sheet marks the semantic of the figure, the kind of the element.** The settings in the style sheet, especially the size of the text, can be overridden by direct formatting. This is for larger fonts explained in the chapter before and shown in Figure 1: View 40% page 5. The style sheet should not be changed by the user. It is defined for this kind of diagrams.

Style sheets are a proven concept for text writing. The direct formatting approach can be also used to a style sheet formatting approach, and both can be combined. A style sheet allows change a formatting style for all designated elements (paragraphs, parts of text etc.) to achieve a uniform

presentation. It is an advantage that is often not enough known. That are common statements.

2.3 Connectors of Libre Office for References between classes

The connectors as known from Libre Office are the proper possibility to connect blocks, which are classes or objects. The connection can be done for the class itself, or for one one of the elements.

You can use connectors with orthogonal lines, or straight or curve connectors as if you want.

Libre Office assigns four connection points ("glue points") to each element by itself. This is sufficient for elements of the class. It is very simple to connect for example the end point of a diamond of an aggregation with the mid of a port as destination of the aggregation, or also with any other class if the whole class is referenced.

For the larger class block with maybe more connections on different positions you can add some more glue points.

Using connectors between elements in your graphic, the connection remains stable if you move some blocks. You may adjust the inflection points (more precise the mid points between inflection). Some commercial tools such as Simulink try to adjust connections between blocks by itself by sophisticated algorithm, which should avoid lines crossing blocks, and make instead mad ways around all blocks only to avoid crossing a free but reserved area for a name of a block. LibreOffice is here more friendly, it does nothing by itself, only move the connection as necessary, and let the user decide about the outfit of the connection path.

A connector as reference between blocks should have also a Style Sheet. If the connected elements are well dedicated by Style Sheets, you can use the `ofRef` style for all connectors. It produces a small arrow on the end, and a line width of 0.2 mm, nor more.

But there is also a possibility using connectors as in UML. The connectors have especially the start arrow outfit as in UML necessary (diamond for aggregation). Then you can use for the connected elements the common style `ofPinLeft` or `ofPinRight` which does not specify the kind of the element. The connector specifies it. That is the originally approach of UML, also possible here (but not recommended). Both are supported by code generation.

2.4 Connect Points for more complex references

Libre Office seems to be have the disadvantage that additional inflection points on orthogonal connectors are not possible. Look on the example right side. The connection from `aggr2` to `port2` through ClassF is not nice.

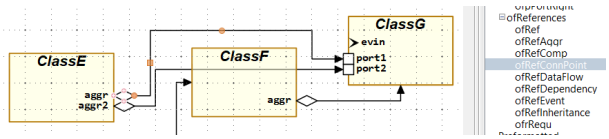

Figure 5: block crossing reference

The solution is shown also image. From `aggr1` to `port1` two connection lines are concatenated. The first line is of type (style) `ofrConnPoint`, its without arrow on end. Both lines together appears as one line, with proper inflection points.

Another question is: Having aggregations or other references with one destination and more sources. In UML often there are drawn parallel. But it is more consequently to use a connection point as it is known from any electrical circuit scheme and also from Function Block Diagrams for data flow. The difference is only: Data flow and electrical schemes has one source and more destination. An aggregation has one destination and can have more sources. The reference line to the connection point is either a simple `ofRef` which has an arrow on its end, or it is the same as in the image above for concatenation of reference lines, with style or type `ofrConnPoint`.

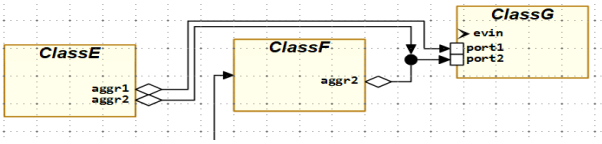

Figure 6: Using a connection point for aggregations

2.5 Diagrams with cross reference Xref

Image Cross Reference usage

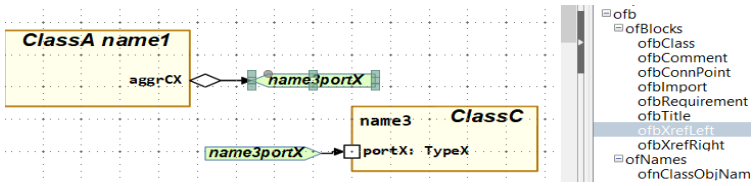

Figure 7: UMLdiagramXrefExample.png
Cross Reference usage

The cross reference or usual nominated as Xref is a often used symbol to replace too much lines in one graphic, or also to make connections to several sheets of a graphic. The last one should not be in focus here, because on graphic sheet presents one aspect, spread one diagram over several sheets is not familiar for UML or also Function Block Diagrams.

You may use a Xref for signals and connections, which are well known from name, and which have basically connection meanings (such as “reset”) and may be connected to more as one block.

- The figure for the Xref can have any form, but should use the given form (copy it from template). The Style Sheet should be either `ofbXrefLeft` or `ofbXrefRight`, whereby the difference is only the text alignment to left or right.
- The name in the Xref symbol should be a mnemonic name for the functionality, valid for this diagram. Here it is a combination of the type of the port and part of name, maybe proper.
- The line from a block to the Xref should be the same type (here a simple `ofRef`) as without Xref.
- The line from the Xref to the block should have usual the same type, but this is not evaluated. Because the type of connection can be also composition or association here, the type for the association is used here, it is not specificated to the aggregation or composition with the filled or non filled diamond.

You can use Xref connections for all line types. The evaluation of the graphic builds a list for all Xref by name per sheet, and checks the connections.

2.6 All Kind of Elements with there Style Sheets

The next image shows all given template elements. It is the content of the file

https://www.vishia.org/SwEng/oofb.wrk/src/UML_FB_DiagramTemplates/odg/

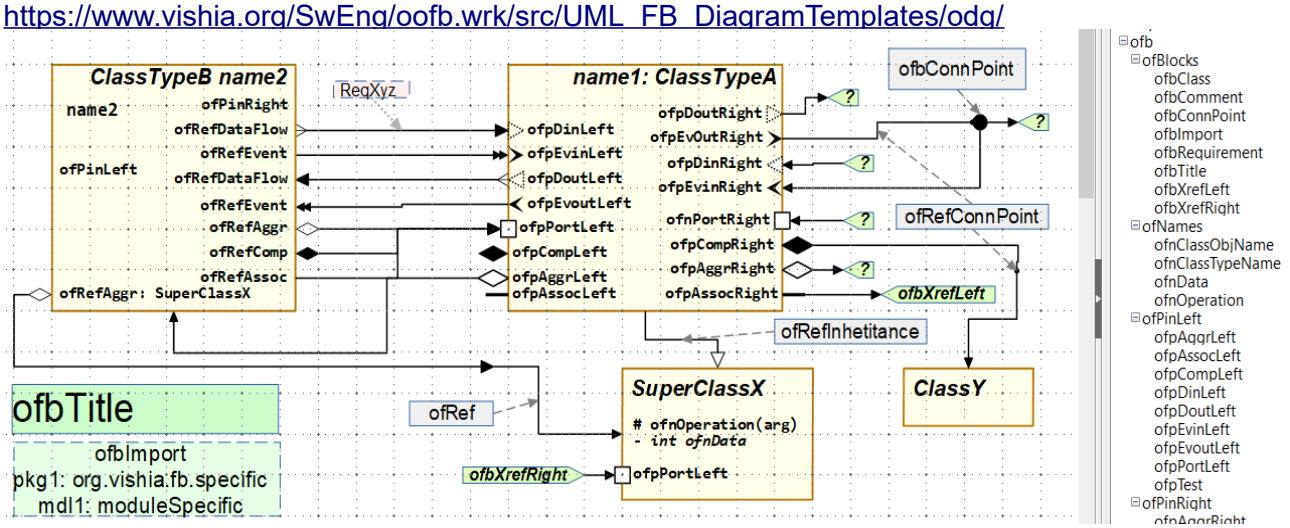

Figure 8: UMLallConnections.png

Right side you see some style sheets. You can use this image (given in the file UML_FB_DiagramsTemplate.otg) to pick an element, copy it to clipboard and insert it in your graphic. The style sheets are copied by opening this file and save it with your name. Unfortunately

Libre Office does not allow loading style sheets from another given odg document, only by copying the original one (see also https://ask.libreoffice.org/t/how-can-i-import-styles-from-other-draw-documents/8834).

The class in the mid with `name: ClassType` contains all connection elements as described in 2.2 Using figures with style sheets for elements page 6. The identifier of the style sheet is here used also as name.

The class left `ClassType name` contains simple connection elements of the base style `ofPinRight` and `ofPinLeft`, but using connections with the specific type. Their style names are shown here as pin names.

The type name of a class is marked with the style `ofnClassTypeName`. A class can have also an instance name, then it is an Object or a *Function Block*. A super class cannot have a name, also not a composited class. Because there are defined by its relation to the using class. A *Function Block* presents a class which is instantiated by a here usual not shown main class of a module as composite. Then its name is written either in the element `ofnClassTypeName` as first, then with colon, or after the class type name. This writing style is also possible for all names of the elements. The type can be written their also, shown only for the aggregation left bottom `ofRefAggr: SuperClassX`. The type of a connection (a reference) may be sometime interesting. In UML a reference should refer always the class which is the reference type. But in this here used combined class-object-Diagram (objects are *Function Blocks*) sometimes the reference goes to the used instance with a derived type, and the reference type should be shown.

The style `ofnClassObjName` is the other possibility to show the name of a Function Block or just class object. Then only the name is written here, parallel to the `ofnClassTypeName` which may/should only contain the class type name. In the left `ClassTypeB name2` both is used.

The internal data of a class can be shown, as usual in UML, with the style `ofnData`. The designation about private, public, protected should be written with a first character `- + #` as usual in UML. Writing the type of the data is recommended. The operations can be written with their argument names, if it is more informational. The operation itself, its body, should be define anyway in a programming code and not with a diagram. The association between the shown operation in a diagram and the real operation is only for documentation, should not be formalistic.

For the documentation blocks the style `ofbComment` should be assigned. A *requirement* is presented also usual in UML with a short identifier. It is written in a `ofbRequirement` rectangle block. The connector between `ofbComment` and `ofbRequirement` has the style `ofRefDocu`. If you copy this connectors from the template, you get also the style reference.

This diagram contains also data and event flow. This is described in todo

3 Working flow creating your own diagrams

First you should load and open the template file from

https://www.vishia.org/SwEng/oofb.wrk/src/UML_FB_DiagramTemplates/odg/UML_FB_DiagramsTemplate.otg

To create a new empty UML class or Function Block diagram you should save this template file under your specific location/name.odg. You should delete the content, the style sheets are not deleted.

Reopen the template file, you need it to copy figures and elements from.

If you have your own file with content but maybe an older version of style sheets, you may copy the style sheets immediately with zip: The odg or otg file is a zip file format. Add the extension .zip and unzip it (simple us the Total Commander). It contains a `styles.xml`. Replace the `styles.xml` in your own file (with zip extension). Remove the zip extension and reopen it in Libre Office. It should work. Do not forget to make a backup copy. This is a non documented way, but it seems to be stable since many years. It works also for OpenOffice in different versions.

Look for Grid and Snap

- Open "*Tools - Options*", select "*Libre Office Draw*" and then first "*General*". Look for the measurement unit, it should be "cm".
- Then open "*Libre Office Draw*" and "*Grid*", look for the proper grid settings (recommended 0.5 cm and 5 Subdivisions because the natural cursor step width is 1 mm. Select "Snap to grid". This is strong recommended, because you have a lot of work for unsnapped blocks and some small inflection points in orthogonal reference lines.

If you have copied from the template, it should be proper.

Create a class or function block:

- Create a simple rectangle in your diagram and assign the style sheet `ofnClass`. The it gets the yellow color with brown border. Alternatively you can copy a class block from the template.
- Create a simple rectangle and write first your class name into it (press F2 to write text in a selected rectangle). The assign the style sheet `ofnClassTypeName` to it. Now move the rectangle into the class box, usual (not necessary, but recommended) to the top right border. You should not place the name exact in the mid, it makes a little bit trouble by selecting the correct glue point for the class rectangle.

  Alternatively you can copy the rectangle from the template.
- Maybe write some data or operations into your class block in the same kind, either by copying from the template, or also by creating simple rectangles and assign the style.

Copy connection elements

- Then you may copy connections (aggregations etc.). For this you should use the template, copy the correct element in your diagram. On paste it lands on exact the same position as in the template, its on the top spread of the page. You can use the cursor keys to shift it to your destination firstly, so long it is selected. Sometimes the landing position is inside any other stuff, this is a little bit confusing. Unfortunately Libre Office does not paste a figure on the cursor position (as other tools do). It would be more proper.
- You can copy more connection points from the same type also from other ones in your diagram of course, it is usual faster.
- On copying and moving the figures the landing position should be any time in the 1 mm-Grid. Sometimes it may be wrong, you see it on small inflection points and obviously misplaced positions. Then you can press F4, correct the position to even mm. If you have activating

snapping, all will be proper after such an adjustment (till a next non obviously positioning which may be also caused by accidentally size changes).

Group and ungroup

This is basically and your daily work.

- You can catch all content of a class block inclusively the connection elements with selecting with left pressed mouse. Then you should select "Shape-Group-Group" from the menu. It is ctrl-G per default.
- After grouping you can move your class block simple with the mouse or by cursor keys as a whole. All Elements are moved together and are stable in the class block.

  The grouping is also recommended and necessary for content evaluation. You should anyway have a grouped situation.
- But in the grouping state you cannot connect the elements of the class with a connector. Only the glue points of the group itself are accessible, and that is not desired.

It means before connecting, and also before shifting some elements you should ungroup. This is in menu "Shape-Group-Group". You should assign the hot key strg-sh-G to ungroup to have a fast work flow. Sometimes also strg-U may be used. Originally this is not assigned. Use menu "tools - customize" and then keyboard. Search "ungroup" and assign the desired key.

You need very often Group and Ungroup.

Small problems with movement

The elements have a height of 2 mm and often only a size of 2 x 2 mm. If you select it, Libre Offce shows drag points to change the size, but because the size is not changeable, also a "non possible" symbol. The space for movement is small in the mid of this points. 2 x 2 mm is the smallest size where movement is possible on a 1920-pixel screen with full size width. This is a little bit stupid.

But you can also move with cursor keys.

Using a higher zoom factor (200 % is recommended) ameliorates this situation. Usual you don't need to see your page margins.

Hint: Bring to Back / bring to Front

The rectangle of a class should have a transparency. Then you see also elements which are arranged below (in the back) in relation to the class rectangle. But to work with, the inner elements should be in front and the class rectancle should be in back. Use the menu entry "*Shape - Arrange*" or the context menu with "*Arrange*" to adjust it.

4 Semantic discussion

What do the diagram contents mean?

This chapter should discuss some presentations in the ObjectOrientated Function Block diagrams in relation to the UML standards. The other topic is the Function Block presentation. Both should not be an objection. It should be thought together for the future

4.1 Content of one or some diagram/s is a package or a module

One Libre Office Diagram contains either a package in UML thinking, or a module in FBlock thinking. Both are suitable for each other because often classes from a specific module implementation are organized in an appropriate package. In FBlock diagrams presentation of one module in one diagram is familiar.

A Diagram should have a title, both for the module which presents it and for the package. This is shown right in the Box with style `ofbTitle`. `pkg`: and `mdl`: are the keywords. Furthermore, other packages can be designate with a alias. That is in the second green box with style `ofbImport` (marked here). The alias is left from colon, right side an unique identifying string for the package respectively for the module should be written, which can be evaluated.

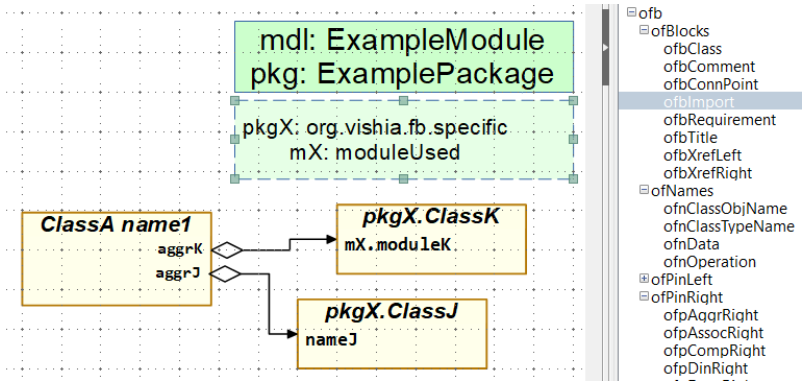

Figure 9: omdImportPkgMdl.png

Just, The `ClassA` is member of the own package "`ExamplePackage`". The instance of this class named "`name1`", a Function Block, is part of the module "`ExampleModule`". The FBlock `mX.ModuleX` can be found in another module with alias `mX`, which is located in "`moduleUsed`", described with other diagrams. It is only used here.

The second FBlock nameJ is member of this ExampleModule, but its type is locate in another package, here the `pkgX`, able to find in the package path "`org.vishia.fb.specific`".

Hint: The package path is Java-affine, for other languages for example a name space or the name of the header file can be written here.

The possibility to define different packages in one UML Class Diagram is not supported here. To show a package structure an enhancement should be done for this diagram writing approach. But usual, in UML there are often small problems in showing used classes outside of the own package because of sophisticated connections (the other package is separated in the diagram). Here using the alias approach classes from another package can be located closed to the using classes.

One package or one module can have more as one diagram! The content is merged as one content for the module, or for the package.

As well as other presentations (especially written source code) can be also used to generate the data content of one module or package. This is especially interesting for modules with the FBlock approach. Other tools, for example Simulink, does not support more diagrams for one module. But Simulink supports nested content in a Function Block with graphical content. On the other side Simulink has not a distinctive module structure. On code generation all modules (except "*Atomic Subsystem*" are flattend. This specificas may be known and regarding on planning of packages and modules.

4.2 UML Class Diagram with ports

Ports are introduced in UML with version 2.0. The meaning of ports differences from destination languages for UML and for semantic approaches. Any where it is an access to inner functionality of a class.

In Java language the anonymous interface implementation is known. This is a proper example for using the port technology.

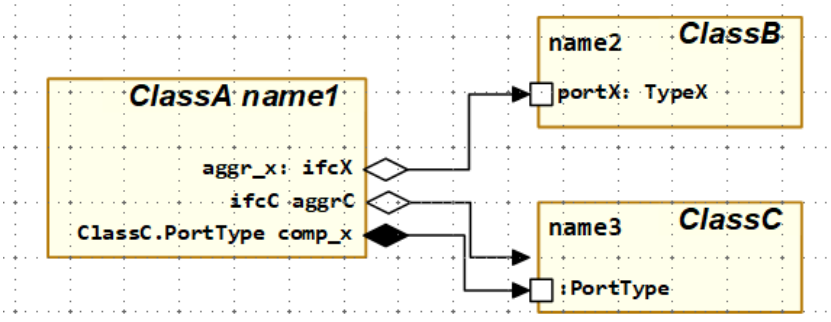

Figure 10: UMLdiagramPortExample.png: Class diagram with ports

An association line does not end on a class box, instead it ends on the symbol of a port (small square) which is designated with the style "`ofpPortLeft`" or "`ofpPortRight`". The difference between "`ofpPortLeft`" or "...`Right`" is only the text alignment, no more.

A port has a name inside the class, as shown here, but has also a type. The type can also be written here, but also in another class diagram or also only in the implementation software. The type should fulfill of course the type of the using aggregations, maybe inherit. The type can be written after the name separated with colon (shown here as `portX: TypeX`) or also before the name separated with space.

As also shown in this diagram, the aggregation itself has also a name and can have a type designation, written in the same way: After colon to the aggregation text, here `aggr_x: TypeX` or also in this example `ClassC aggrC`. As known in ObjectOrientation, a reference of a base or super type can refer an instance of a derived type. If the diagram is checked and correct, the `ifcX` is a super type of `TypeX` which is used for the port connection.

The types are detected also by the connection itself. If the type is given in graphic for the destination of a aggregation etc (a reference), the most abstract seen type is used automatically, it it is not determined by the software itself.

4.3 Using data and event flow

Data and event flow are usual for Function Block Diagrams such as IEC61449 automation device programming (https://www.eclipse.org/4diac/en_ide.php) or also Simulink (R) Mathworks. But events are familiar also in UML for State Charts, and they are familiar in Object Orientation.

Look on the right diagram. There is an event flow From `classE` to `classG`. It may meant that `ClassG` may have a state machine, and `ClassE` triggers it. This is an important information.

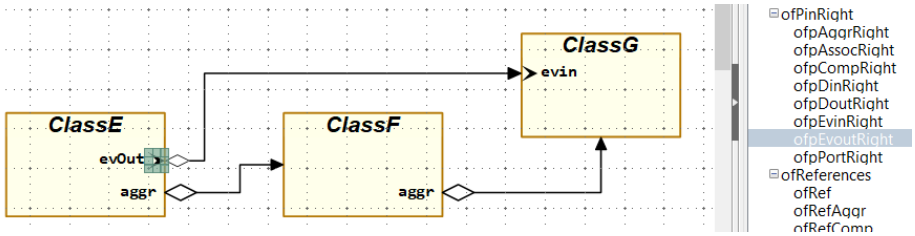

Figure 11: Event flow in UML

Secondly the connection is shown as A**ggregation**. It shows that `ClassE` knows the instance of `ClassG` as event destination (as event interface) in an aggregated form. From where comes the knowledge? It is also shown here, `ClassE` aggregates `ClassF` and this aggregates `ClassG`. Hence the knowledge about the event interface of `ClassG` may come (suggested, not anytime true) via this path. Details can be written in text form or documented in the sources. The important thing is, it illustrates it to support documentation. In Standard UML the aggregations may be present in the same way, and the event destination is described formally, but not in a diagram.

Here the class view is documented: "*Any instance of ClassE can sent an event to ClassG*", not the instance (Fblock) view "*which FBlock in this module sends the event, which receives?*""

Right side you see event and data flow and the fundamentals of Function Block programming. If you don't look at the grayed area, you see three classes with instance names. That are *Function Blocks*. There is a data connection from `nameE` to `nameH` and an event connection from `nameE` to `nameG` . The classes knows and contains the adequate pins and properties, but the instances are in focus. That is presented in this kind in a Function Block Diagram. But from where the instances are created and known? In a Function Block Diagram this is part of the module implementation. It is not specially documented, because it's a detail of implementation. In this diagram it's shown in the manner of the UML Class Diagram: All classes knows a `ModuleMainClass` via aggregation, and this main class creates the module classes which are the FBlock instances (composition). All classes can access all other instances of the module via this knowledge, and a data or event flow is immediately able to program. An Aggregations from `ClassE` to `ClassG` as shown in Figure 11: Event flow in UML for a direct reference is possible, not shown here, it is not contradictory.

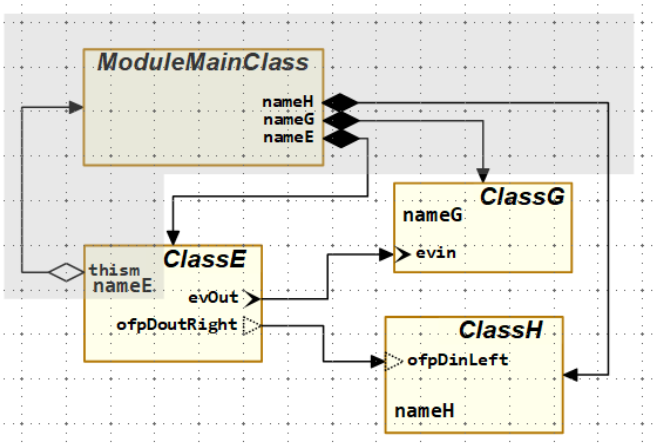

Figure 12: UMLFBlockDiagramDataflowExample.png

The grayed part can be dismissed for explanation of the correlations and also for code generation, because it is general clarified how the instances are created and referenced.

As you see, Function Block diagrams and Class (or Object Model) Diagrams in UML comes together.

4.4 How is an event or data flow implemented

The diagram shows, there is an event respectively a data connection, no more. How it is implemented, depends on the code generation or the associated code.

An event flow uses usual an event queue as intermediate store. The event is stored and red out from the organization software if the function block (with state machine) is executed. That can be occur in another thread, but maybe also in an other device. Then the event is not only stored in the queue, it is also transported via any telegram interface (Ethernet, Socket, Serial etc). For transportation and storing, it should be designated as "*message*". - Hence, its execution is separated from the transmission. But the execution order of events depends strongly from the event order itself.

Remark also that events are usual combined with data (contains data). This data are transmitted too via the telegram in the message. They are stored also in the queue, independent from other data. It means, the data associated to the event are (should be) any time consistent. Extra mutex mechanism are not necessary (other than on accessed data via references).

The data flow means primary, that the destination FBlock participates from defined data from the source FBlock. Often it is a data transport immediately while or after the emitting FBlock (the source of the data) is executed. This is similar as for example in Simulink implementations, whereas the organization software to execute the FBlocks read the data from the output and writes it in input structures. In this kind the data flow can be implemented as an operation call with the data as arguments. A so named "setter" of the destination FBlock is called.

But there are also other possibilities, for example offer the data in a memory range where the destination FBlock can access it if it is executed. This was for example implemented in the STRUC-G Function Block programming in the SIMADYN-D Hardware from Siemens (in the 1990th).

The access of the data from the destination back to the source is also possible, but this is intrinsically an aggregation from the data destination to the data source and may be written hence as aggregation. You have the problem of mutual exclusion and time correctness of the data, because the accessed data may changing in the moment of the access (on multiple threads) or are changed meanwhile from the time of there creation to the time of the usage. That's why a data flow should be definitely disclosed in a proper graphical software description.

But vice versa, in a pure Function Block programming environment (such as Simulink), a data flow can be used to map an aggregation (which is not possible in this environment): The data flow emits the reference to the data, for example in form of a handle, and then the destination FBlock can access data from the source FBlock. The data flow is a flow of the reference, only necessary one time on startup, and a non visible aggregation is existing in fact. See https://vishia.org/smlk/html/SmlkSfn/ObjOrientModels.html

The event and also the data flow can determine the execution order of Function Blocks. If all function blocks are organized with event flow, then the order of events determines the execution order, also if the FBlock instances are distributed in different devices. This is the approach of the IEC61449 automation device programming (https://www.eclipse.org/4diac/en_ide.php).

But also the data flow can determine the execution order in a pure data flow oriented system. Tthis is in Simulink. The rule is: If all data are presented, then the FBlock can be executed. If more as one Fblock can be executed in the same time, it is anywhere, the order does not be important, or better, it can be executed independent in multiple threads or cores.

If data flow and events are combined, it is a little bit more complicated.

5 Evaluation of the content for code generation and / or textual documentation

Working with Libre Office for drawing diagrams will be really non sensible if the content is not able to evaluate. Because:

An only documentation oriented documentation is dead. If the code is changed, sometimes the documentation cannot be updated, it is too much effort to find out all positions where the documentation should be updated. It is an old style to write documentation independent of the code.

But we know usual from other tools, that working immediately with text oriented sources in programming languages with proper IDEs (Integrated Development Environments) is often proper and effective. The combination with graphical programming is often sophisticated and lesser helpful. Precisely for this reason, this work, using Libre Office with evaluation of the content, was carried out as an alternative solution to the well-known and (advertised powerful) professional tools.

The goal for evaluation the content of this Libre Office diagrams is not: "*Just have another (more powerful?) code generation*". The goal is: present the content in a textual readable and comparable form. Then it can manually used, it can be used for text based comparison with older or other content. It can be used to compare with textual given source code, maybe with preprocessing the source code to get an adequate presentation (parsing the source code, output an intermediate format). In that kind the code can be tuned to the graphic so that the graphic matches to it.

The generation of the textual form from the graphic should be able to control by textual templates. So a specific code generation or a specific comparison format can be aimed.

That is shown in the next sub chapters.

5.1 The file format of odg

Let's have first a look to the file format from Libre Office. The odg format is a zip archive. You can add the extension zip, and then look into with a zip utility.

Right side you see a screen shot from the opened zip file (with Total Commander). The zip file contains three important xml files.

- content.xml contains the graphic itself
- styles.xml contains the style sheet settings. If you want to copy your settings between some files, you can copy this styles.xml inside the two zip file. It seems to be safe.
- settings.xml is not relevant for the content itself, also the other files are helper for the Office tool.

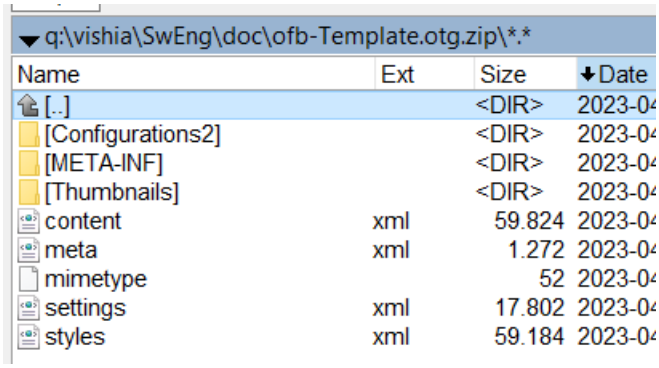

Figure 13: ContentOfodg.zip.png

Now have a look inside the content.xml (pressing F3 in Total Commander to view to pure textual content:

It is one very long line without structure not well human readable, but it is well formed XML.

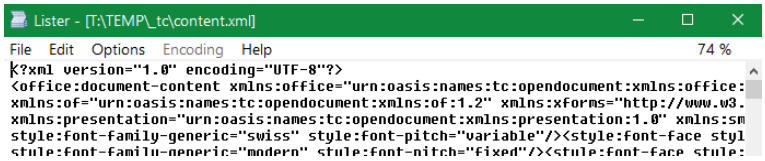

Figure 14: ContentOfodg-content-xmlPure.png

After beautification it looks like

```
<draw:g>
  <draw:custom-shape draw:style-name="gr21" draw:text-style-name="P1" draw:layer="layout"
    <text:p/>
    <draw:enhanced-geometry svg:viewBox="0 0 21600 21600" draw:type="rectangle" draw:enhan
    </draw:custom-shape>
  <draw:custom-shape draw:style-name="gr22" draw:text-style-name="P2" draw:layer="layout"
    <text:p text:style-name="P2">
      ClassA name1</text:p>
    <draw:enhanced-geometry svg:viewBox="0 0 21600 21600" draw:type="rectangle" draw:enhan
    </draw:custom-shape>
  <draw:custom-shape draw:style-name="gr23" draw:text-style-name="P7" xml:id="id18" draw:i
    <text:p text:style-name="P7">
      aggrCX</text:p>
    <draw:enhanced-geometry svg:viewBox="0 0 21600 21600" draw:glue-points="10800 0 0 1080
    </draw:custom-shape>
  </draw:g>
```

This is right side truncated, it shows the graphical "group" with the "`ClassA name1`" as shown in Figure 7: UMLdiagramXrefExample.png Cross Reference usage page 8. You can see here also the aggregation `aggrCX`. The style names are not written immediately plain here, instead a referencing is done, the `draw:style-name="gr23"` describes some possible direct formatting properties and the references to the knwon style "`ofpAggrRight`" as you see in the content.xml in the `<style...>` part.

```
<style:style style:name="gr23" style:family="graphic" style:parent-style-name="ofpAggrRight">
  <style:graphic-properties draw:marker-start-width="0.24cm" draw:marker-end-width="0.24cm" f
  <style:paragraph-properties style:writing-mode="lr-tb"/>
  </style:style>
```

This is all understandable and comprehensible. Hence read out of data is only a problem of sorting.

5.2 Software to evaluate the graphic information from Libre Office

This software is given completely in two jar archives. You should start per command line:

```
java -cp tools/vishiaBase.jar;tools/vishiaFBcL_odg.jar org.vishia.odg.ReadFBcL_odg
```

If you start the software without arguments, you get an argument description:

```
...Reader content from odg for FunctionBlockGrafic
obligate args: -o:... ...input
-o:path/to/output.file
-datahtml:path/to/data.html
-oxmldatahtml:path/to/xmldatahrml.html
-oxmltest:path/to/outTest.xml if given writes the red xml input back
-obeauty:path/to/outputBeautificated.xml if given writes the beautificated input
path/to/input.odg
```

Given proper arguments produces the outputs.

For all example diagrams which are contained in the `src/UML_FB_DiagramExamples/odg/ofb-UML-Examples.odg` you get a text file which contains for example for the ClassA the following read out content:

```
/**A type definition ^= class in this package.
 */
class ClassA {  //
    ClassB assocX; //association
    ifcC aggrC; //aggregation
    TypeX aggrCX; //aggregation
    ClassC aggrX; //aggregation
    ifcX aggr_x; //aggregation
    ClassD compX; //compositon
    ClassC.PortType comp_x; //compositon
    EvTypeZ evin; //event_in
    EvTypeZ sendEvent; //event_out
    (float) dataXy; //data_in
    (float) dataOut; //data_out
} // end class
```

This is the summarization of all drawn content for the ClassA.

For the instance name1 which is type of ClassA the summarization is:

```
/**An object (FBlock) in this module.
 */
FBlock name1 : ClassA {  //
  association assocX  =: name2.@fbSrc ;
  aggregation aggrC  =: name3.@fbSrc ;
  aggregation aggrCX  =: name3.portX ;
  aggregation aggrX  =: name3.@fbSrc ;
  aggregation aggr_x  =: name2.portX ;
  compositon compX  =: @ClassD.@fbSrc ;
  compositon comp_x  =: name3. ;
  event_out sendEvent  =: name1.evin ;
  data_out dataOut  =: name1.dataXy ;

  // back tracking info:
  event_in evin  =: name1.sendEvent ;
  data_in dataXy  =: name1.dataOut ;
  <??null??> <??null??> ;

} // end class
```

Here you see the connections from and to the pins of this FBlock.

To produce the results in this form there is a template file with the following whole content:

```
=== odgPin
    <&pin.sType> <&pin.name>; //<&pin.kind.sKind>

=== odgPinObj
  <&pin.pinClazz.kind.sKind> <&pin.pinClazz.name> <: >
    <:if:pin.idxPinConnected><:for:cpin:pin.idxPinConnected> <: >
      =: <&cpin.fb.name>.<&cpin.pinClazz.name> <:if:cpin_next>
          ~ <.if><.for><.if>;



=== odgClass
/**A type definition ^= class in this package.
 */
class <&fb.sType> {  //
<:for:assoc: fb.idxAssoc><:call:odgPin:pin=assoc><.for><: >
<:for:aggr: fb.idxAggr><:call:odgPin:pin=aggr><.for><: >
<:for:comp: fb.idxComp><:call:odgPin:pin=comp><.for><: >
<:for:port: fb.idxPort><:call:odgPin:pin=port><.for><: >
<:for:evin: fb.idxEvin><:call:odgPin:pin=evin><.for><: >
<:for:evout: fb.idxEvout><:call:odgPin:pin=evout><.for><: >
<:for:din: fb.idxDin><:call:odgPin:pin=din><.for><: >
<:for:dout: fb.idxDout><:call:odgPin:pin=dout><.for><: >
<:for:pin: fb.idxUnspecConn><:call:odgPin:pin=pin><.for><: >
} // end class



=== odgObj
/**An object (FBlock) in this module.
 */
FBlock <&fb.name> : <&fb.fbClazz.sType> {  //
<:for:assoc: fb.idxAssoc><:call:odgPinObj:pin=assoc><.for><: >
<:for:aggr: fb.idxAggr><:call:odgPinObj:pin=aggr><.for><: >
<:for:comp: fb.idxComp><:call:odgPinObj:pin=comp><.for><: >
<:for:evout: fb.idxEvout><:call:odgPinObj:pin=evout><.for><: >
<:for:dout: fb.idxDout><:call:odgPinObj:pin=dout><.for><: >
<:for:pin: fb.idxUnspecConn><:call:odgPinObj:pin=pin><.for><: >
<:if:fb.fbPinDst><:call:odgPinObj:pin=fb.fbPinDst><.if>
  // back tracking info:
<:for:port: fb.idxPort><:call:odgPinObj:pin=port><.for><: >
<:for:evin: fb.idxEvin><:call:odgPinObj:pin=evin><.for><: >
<:for:din: fb.idxDin><:call:odgPinObj:pin=din><.for><: >
<:if:fb.fbPinSrc><:call:odgPinObj:pin=fb.fbPinDst><.if>
} // end class

===
```

This file is used with the

https://www.vishia.org/Java/html/RWTrans/RWTrans.html#_outtextpreparer

or see also its Javadoc:

https://www.vishia.org/Java/docuSrcJava_vishiaBase/org/vishia/util/OutTextPreparer.html

contained in the `vishiaBase.jar` file als general solution. It accesses immediately to the converted data.

2023-05-08 by Dr. Hartmut Schorrig www.vishia.org