|  |
| --- |
| Circle Language Spec: Misc Diagram Topics |

## Diagram Expression

*Expression* includes a *diagram notation* for visualizing the program code. The diagrams are a mere expression of binary interlinked objects. Systematics, stored in a binary way as interlinked objects, are translated to a diagram, expressed on screen. The diagram should be fully automatically drawn out. The metrics and positioning of the shapes and lines is automatically determined. The diagram can be edited, resulting in changes to the stored binary objects.

How *each concept* is expressed in a diagram is already covered in the documentation *of those* concepts. The basics of the diagram notation were already covered at the beginning of the new computer language documentation. *This* documentation folder explains everything *else* about diagram the diagram notation, such as how the shape positions of the diagram are automatically determined and how the diagram can be *colored*, to make the systematics even more clear.

The articles in the *Diagrams* documentation folder are not finished. They consist mostly of texts coming out of an earlier unfinished design of the diagram notation language. The diagram notation has not been realized in software yet.

This folder contains the following articles:

### Diagram Metrics

This article is not finished. However, *diagram metrics* are very important to make the new computer language a success, because if the diagram is not automatically drawn out, and you have to position all the shapes in it yourself, then the language is not that easy to use anymore. The diagrams are automatically drawn out and you get very little say in where the shapes are positioned and how the lines between shapes are drawn out. The interconnection between shapes and the embedding of it, should pull the shapes and lines into the right position like molecules.

### Coloring

There is not a lot of text in this documentation folder yet. This documentation folder is to contain explanations on how the shapes inside the diagram can be colored to highlight certain parts of the system. A system can be fully drawn out with diagrams not using any coloring at all, but coloring may highlight certain parts of the system. You could highlight all objects of a certain class, or you could highlight all classes, or highlight all interfaces, or highlight all objects with classes from a certain site. The possibilities for coloring are endless and you could apply coloring with an infinite possibility of conditions. In the future, a few standard colorings may prove to be particularly handy. All this coloring is for analysis purposes, to highlight certain aspects of a system. You could also just let circles be blue, squares be red, triangles be yellow, diamonds be orange, etcetera. You could just draw the outline of shapes or you could fill the shapes up. Each containment level could get its own color. You could switch between two colors on each deeper containment level. These are all sorts of beautiful possibilities for coloring.

### Example Diagrams

This documentation folder contains a zip file with pictures of *example diagrams*. Some of the diagrams come from analysis of real-life situations, and they are just too inspiring to throw away, so they have been given a place here.

## Ideas

*The texts below are loose ideas, yet to be turned into good documentation.*

Diagram Expression concepts:

- Argument Reading, Argument Writing

- Diagram Line Direction

- Diagram Metrics

- System Methods

The diagram does need to be able to display the whole system in terms of just coding principles, and not coding concepts, because you need to be able to start programming coding concepts using the diagrams. The diagrams need to already be in the language, when you don’t yet have coding concepts.

However, coding concepts will be applying to the display of diagrams.

-----

For practical reasons, a triangle's purpose may have to be maintained. This does make the diagram not be an exact representation of ALL the systematics, so that sucks. The different purposes of the triangle may be base class, interface, group, joint access modifier, etcetera. I guess if you want to group members by different aspects, triangles must overlap. I already brainstormed about that some time in the past.

> But overlap can be avoided by automatic containment? Couldn’t it? How would that look? It would create secondary references, won’t it?

-----

Symbol:

The CONNECTIONS between symbols, when 'molecularly' positioned, can form shapes that tell about about the flow of the system. These shapes can become placeholders for the system if the view on it is too small. That way, the connections between symbols form a symbol itself, that stands for the system.

-----

This is nice to put in documentation: that the symbols and their connection, besides being separate connected shapes, when positioned molecularly, will form symbols of their own, consisting of a combination of symbols. When you use standard positioning, these symbols can be interpreted as a whole, instead of looking at each symbol individually.

-----

Show literal values in the symbols at design time, so you can see the result of a calculation immediately.

-----

Not all modeling information is most usefully presented in a graphical notation. Some information is best presented in a textual or tabular format.

-----

Je kunt interfaces toch laten overlappen. Elke drhk die je aanbrengt om te groeperen, creeert meteen een nieuwe interface. Je kunt dus naar een object refereren met als type een groep functies van het ding. Dit creeert meteen een geraffineerde interface onderverdeling in functionaliteit. Omdat dingen in S overzichtelijk blijven, ondanks de vele objecten, is dit alleen maar ok.

-----

Een Public interface zal er zijn, een Private interface base interfaces en dus interfaces per groep. Wat dus wel zo is, is dat een member bij verschillende interfaces kan horen. Eigenlijk creeert dat een member per interface, die doordelegeert naar een hogere member, dus daar zit nog wel een vraagstuk. Bij het omzetten van referentieel naar encapsulatief, maak je ook referenties aan? Nee. Dat een referentie maximaal maar 1 hoger mag zijn klopt niet. Een super-super object kan makkelijk een referentie aanleggen in een sub-sub-object, naar een ander object binnen het super object. De interfaces zijn aparte objecten, dus moet er wel een referentie per interface zijn. Er zou een 'lapmiddeltje' kunnen komen om overlappende interfaces toch mogelijk te maken, door een mechanisme in te bouwen, dat referenties naar een hoger, zelfde symbool als overlapping ziet. In wezen zet je een overlappingsbril op, want de afzonderlijke referenties en target symbool bestaan gewoon wel. Net zoals je eigenlijk een encapsulatiebril op kan zetten, terwijl in wezen de structuur gewoon helemaal referentieel is.

-----

You CAN reduce the variety of things computers are applied for (flows of application of computerprocesses) to a set of standards, such as UML attempts to do. However, I think UML is very incomplete in that area. I would like to integrate diagrams of flows of application of computerprocesses in Symbol and make it integrate into the other kinds of diagram.

-----

Along with very custom views that are the actual applications for for instance playing or manipulating sound files or graphics files.

-----

You'd want a diagram that represents every aspect of a system with processes and schedules and user programs Sy isn't good for that yet.

-----

Perhaps sometimes in Symbol diagrams, only the foreward relation should be shown.

And the backward relation should be implicitly there?

(not all relations are dual, though)

Perhaps the fact that something is a foreward or backward relation should be expressed by direction the way I do that already in symbol. Perhaps the containment structure should be derived looking at the relation direction also, and favor outward pointing? (highly disputable) For sy, you should first be looking at how to express a relational structure in an as simple matter as possible, before looking at how to express the object oriented part, and perhaps use elements to express relational, that were originally reserved for expressing OO. The thing about sy was to have an as simple but as explicit as possible expression of OO, because OO was the most important concept. Then it was important to me that all of the system could be expressed in Sy. But now it turned out relational is the most important concept. And now I have to again look first for the simplest and most explicit way to express relational. A dual relation should be expressed as a single line between two circles, I would say, in that case.

-----

Ik zie allemaal molekuulstructuren,

als ik aan cre denk,

al die objectjes die leven...

in 3D. Maar dat worden natuurlijk symbol diagrams, maar goed.

-----

If you select an item in one symbol diagram, you can have a second symbol diagram next to it, that shows that single selected item in more detail.

You can also have two diagrams that independently show different parts of the system.

Don’t want to consider

Ik zie hier duidelijk dat je uit code niet zo makkelijk de structuur van een programma kunt zien. In S kan dat wel. Met UML ook, maar UML is extra werk om te doen, In S niet.

-----

Weet je wat ook leuk is aan Sinas:

UML is niet zo direct meer nodig, omdat je sowieso nooit met code zit zonder diagram.

(ik merk bij veel projecten op het werk, dat als ik oude code in moet zien. Het eest moeite kost om de structuur weer te doorgronden. UML geeft meteen de structuur weer, maar dan moet je dat wel eerst maken. Bij Symbol hoef je het niet apart te maken. Er zijn wel tools om een code geraamte te creeeren uit UML. Together bijvoorbeeld.

Hmmm...

-----

Refer to the reference target coding concept, saying that that is the way to make a single symbol in the diagram represent the actual object, whereas the others are just seen as references to the object. Or put the other way around: to have the actual object only represented by a single symbol.

##### Invert means of display of containment and reference

< (So there is no exchangeability between containment and a line. I used to think that) >

< Somehow I think you could inverse the picture and replace containment by reference and reference by containment. >

< For that I display the containment to a flat reference structure two levels deep.>

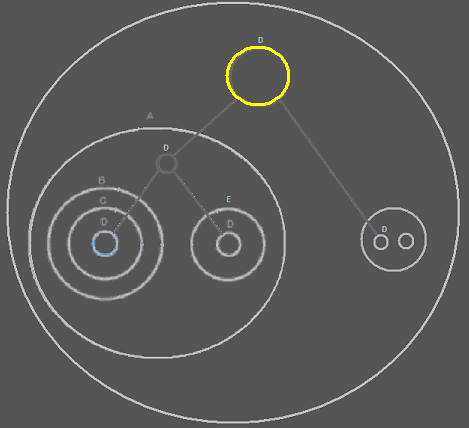
< I replace the references going outward in that, by lines directly connecting the shapes. >

< I take the tree that connects the symbols representing the same object. Instead of direct lines I display the lines as embedded references >

< Then I convert that to a containment structure. >

< Now I somehow have to place the two produced pictures inside each other. >

< I just don’t know what the use of it is. The concepts of containment and lines between references to the same object are not interchangeable: one is about how one object refers to the other, the other is about indicating which are the same objects. >



-----

Informal line types:



In that picture I also see, that I connect the two method interfaces inside the imaginary class definition to the method interfaces on the separate sites using interface lines. Theoretically that is not right: not only the interface of the symbols connected with an interface line is the same, the definition is the same, so theoretically they should be connected with class lines.

But I made them interface lines just for clarity, because it is all about interface definitions, and now all things about interface definitions are expressed in dotted lines, and all things about class definitions are expressed in dashed lines; it is just clearer this way.

But does this create an ambiguity? The interface line between the interface definition in the imaginary target class and te interface definition on a site, litterly does mean, that only the interface of the two objects is the same: that means: the public contents of them is the same, but the private contents are different, which means the methods have the same interface, but they are different methods, with just the same signature. That’s not what I intended to express, though. The interfaces in the sites should even have been drawn with dotted lines either, because that would imply you are only using it as an interface, while really you’re using it as a definition.

The ‘wrong’ notation is just so much clearer, but it does not formally express the system correctly, because it implies two symbols only have the same interface, but really they have the same definition. But the ‘wrong’ notation of it is just so much clearer. I’m going to have to invent rules that make the ‘wrong’ notation formally ok, or the ‘right’ notation more clear.

-----

J Code Diagrams,

Je kunt ook J Code diagrammen als een 'getilt-te' landkaart weergeven.

JJ

-----

Diagram Expression,

"Your screen doesn't have to do everything. Your brain does the rest."

JJ

-----

Diagram viewer,

Doel van je target symbol in een apart scherm ernaast laten zien /

definitie meteen ernaast.

JJ

-----

Diagram Expression Implementation,

2008-06-10

The diagram expression module can get extended after programming a concept, that affects the diagram notation. You then get a new version of the diagram expression module. To show less or more detail in the diagram regarding basic or advanced concepts, you can might go up and down in version of the diagram expression module.

JJ

-----

Diagrams, showing definition:

2008-06-10

When you're at a certain object, show the definition of the class at the side of the screen. You could also choose to display all the definitions of the sub objects.

When you're at a definition, you may opt to show all the sub objects' system interfaces.

JJ

-----

Diagram Expression,

A special thing about the notation of definitions (command definitions,

classes, interfaces, command interfaces) is, that it expresses all the POSSIBILITIES, to connect objects to eachother. An OBJECT will only express the current situation the object might be in. The difference is visible in, that connections that are not yet made, ARE displayed as the possible connections, that can be made. Only if you for instance type something as a class, or USE something as a class, the possible connections are displayed. Heck, even for objects I would probably need to display the possible connections to make with it, in order to display all the connection points.

JJ

-----

Diagrams,

Diagram metrics is not even present in among my articles, even though it is basically a fundamental principle, because the system cannot be programmed swiftly, if symbols in the diagram are not positioned automatically.

JJ

-----

Diagram Expression,

2008-07-06

There is a difference between an implicit expression and an abstract expression. Abstract expressions are important. What is an abstract expression needs to be defined. It expresses the type of correlation between objects, but for one side of the correlation it is not expressed what it is specifically tied to, just *the way* it is tied to it.

JJ

-----

Diagram Expression,

2008-07-20

I realized I frequently use the ellipsis symbol, for a placeholder of stuff not displayed in a diagram. Perhaps I should give it a more defined place in the diagram notation in the future.

See the article System Interface Summary in a Diagram.

If you decide to use the ellipsis symbol, you also need something to *collapse* a part of the structure again.

JJ

-----

Basic diagram elements,

2008-07-27

< 2008-09-02 Part of this story is outdated, but some things are still relevant and nice explanations. >

The line types solid, dashed and dotted are all three to be used for commands after all, because solid can be used to indicate when two are the exact same definition, because a definition is a permanently created command object. Then you always need to use a dashed line when you indicate a call’s command definition. But imaginary references to the same command definition in automatic containment of command definitions, you have to tie the command definition symbols together with solid lines, so that the static parts of the command definition are the same for all references to the same command definition. If you’d use dashed lines between command definition symbols, then the symbols are not the same object, so won’t have the same instance of static parts.

Or the rule must be: a solid line going from a command call to a command definition is actually a class line. A solid line between two command definition symbols is always an object line.

Not creating such a simplified notation, produces more dashed lines, but it does make you more aware of what kind of connection there is between definitions and calls, and it becomes more apparent, that calls are briefly created objects that are destroyed after they’re done, and that definitions are more permanently available. It makes converting an object into a command and back more understandable as well.

JJ

-----

Basic Diagram Elements,

2008-08-02

For each basic diagram element, you have to motivate why this is the most appropriate symbol for it. Why this symbol is chosen. This motivation is often just philosophical of nature. In the articles about a coding concept, that also introduce a basic symbol, you also have to put this motivation.

JJ

-----

Diagrams,

2008-09-09

Rules or guidelines about where to start when reading a diagram.

JJ

### Out of the original Symbol documentation

A possible notation: showing things further away as smaller.

-----

Seeing everything that's directly or indirectly referring to another thing.

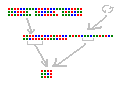
Seeing everything that's directly or indirectly referenced by something.

-----

Computers,

2009-01-29

How I visualize computer processing on bit-level in my head.



JJ

-----

Diagram Expression,

2009-06-21

Conditional visibility. Is that abstract diagram expression or a special form of coloring?

What about coloring that sets the transparency too?

JJ

-----

Diagram Expression,

2009-10-22

In Diagram Expression you can easily see correspondence in 'object' through lines, rather than through identifier. Identifiers are the only way in which mutual usage is visible in text code. That's the big difference.

Any sign of repetition is an invitation for simplification / refactoring.

The key in all of these things is mutual usage and repetition.

The key in the build-up of diagrams is also mutual usage.

JJ

-----

Diagram Expression,

2009-10-22

What if correspondence in value is what you would want to express. Then the wavy line would stop being an assignment, but start meaning correspondence in value. That would make things more consistent, and perhaps from a mutuality point of view, you would decide, that this is the way the language has to be.

JJ

-----

Diagram Expression,

2009-12-08

I think I have already mentioned this idea. The diagrams for now express value assignments, but not mutuality of value. Mutuality of everything else is visible, but mutuality of value is not.

JJ

-----

Taken out of the Interface chapter on 2010-05-07:

Where to start when you read a diagram like this? You first read the containment structure. You start at the largest container, then the smaller ones. After that you look at the relations made by the connecting lines.

You don’t always start with the largest container. Sometimes one symbol is highlighted. Then this symbol is what the diagram is about and it's all about the aspects of this symbol. Then you start at the highlighted symbol and look around. There’s no one way of doing it.

JJ

-----

Software System, J Code

Mogelijk kun je bij een bepaalde view op gegevens aangeven of alle objecten 1x vertegenwoordigd zijn, objecten meerdere keren vertegenwoordigd worden, of niet alle objecten vertegenwoordigd worden, en mogelijk de beperking waarop... De diagrammen, daar staan wel meerdere referenties naar hetzelfde object in, maar de referenties wijzen expliciet naar 1 object, dus er is toch sprake van 1x vertegenwoordiging van het object, want het target symbol vertegenwoordigd het object.

JJ

Out of the original Symbol documentation

Language Integration

< State cannot be expressed in diagram code. It can only be expressed in an alternative notation>

Symbol Language embeds techniques that makes you able to integrate any other language into it.

To demonstrate how other languages, including graphical ones, can be integrated into Symbol I will show how text code is established.

Text code is made out of a symbol by giving a reference to a symbol to a translation procedure. The translation procedure will construct a String object that contains the text code. Thus, a translation procedure takes a system and translates it to an object that represents the system. If you don’t translate the system to a String, but another type of object, containing shapes and lines, for instance a Shapes object, then you can establish a graphical language.

The symbol diagram language is such a translation to a graphical language… … yes, unfortunately the diagram language introduced so far is nothing more than a utilization of Symbol. It isn’t symbol itself. A system is retained in a J Data Base. The J Data Base stores the symbols and their relations and also a few standard translation procedures, which are also systems. Some of those standard translation procedures draw out the Symbol diagram language. Another set of those standard translation procedures produce the Symbol text language.

-----

Dit drukt er een stempel op dat alles binnen de computer gebeurt. De taal heerst in de computer. De taal laat het toe weer gegeven te worden, maar alles gebeurt binnenin de computer. De weergave (in diagrammen) is niet het primaire element van de taal. Het primaire van de taal is de betekenis. De taal zelf is daaraan ondergeschikt.

-----

The parrallel between on one hand

strings, fonts, GDI

and on the other hand:

simple shape description, GDI, ways to draw the shapes (anti aliased, not anti aliased …)

-----

Other

2008-10-10,

A text out of the old symbol documentation said:

List diagram elements

List programming aspects

Programming aspects with diagram elements.

Then associate diagram elements with programming aspects and

Not so long ago I separated the conceptual explanations from the expression in a diagram and in text code.

But the list above leads to not only explaining all concepts in a diagram,

but also all diagram elements as a concept.

That can lead to the specification of the most basic concepts, which have the most basic expression in a diagram.

Maybe the order of explanations will be adapted to that.

But I won’t start working on that right now. It is something for the future.

JJ

-----

More Ideas

Expression,

New Computer Language Functional Design,

2009-04-20

You'd have to integrate tooltips into the diagram, e.g. to explain what to do with an access connector.

JJ