# Circle Language Spec Plan Automatic Containment

*Author: JJ van Zon*

*Location: Oosterhout, The Netherlands*

*Date: May 26, 2008 – June 21, 2008*

## Goal

The ideas behind *Automatic Containment* got a bit complex with quite a few loose ends. Working on this topic could include incorporating ideas, found during the project *Diagram Expression for Classes & Relations*.

## Automatic Containment for Relations

The desire to have most relationships become bidirectional, seemed to lead to a bit of a problem. The notation for a bidirectional relationship could be a *line merge* and a *symbol merge*.

In a strict approach, this may make the diagrams look much different. To get a clearer view of it, an example diagram might be drawn out with line and symbol merges and alongside it, one *without* those line and symbol merges. Then it might be better visible how that works out in these diagrams.

One thing suspected, is that more things will end up next to each other, instead of inside each other. That may anull the effect of having a containment structure / like a map of your code.

So the 'problem' here is the friction between wanting bidirectional relationships, a simple, clean notation for it and on the other hand how nice containment structures might look in the diagram notation.

Some of the possible solutions:

* Drop the idea that all relationships should be bidirectional.
* Drop the idea that referential structure should be automatically converted to a containment structure / automatically determining composite aggregation derived from association aggregation. It may be a nice idea for the future, but it seems to conflicts with other nice ideas.
* Make a relationship have a primary direction, like the arrow in Parent 🡪 Children. You tend to display the Children as actual children in the containment notation, not as sibblings, because of the bidirectionalyt.
* Lists may tend to be displayed as children, which may automatically define what is a child, and what is a sibling.

## Lower Contents Brainstorm

*This brainstorm was written in the context of efforts to program a prototype app 'Circle 3'. It might not be very readable.*

If something is declared on a higher level than it should then it is one real reference there pointing at an imaginary one at the same level with the contents in it, to which points one other imaginary reference on a deeper level.

And in that case the contents should be moved from the imaginary reference on the higher level, to the imaginary reference on the lower level, and the imaginary reference on the lower level should point to the real reference on the higher level.

That last reference line would be a qualified reference line, but that will not be well visible yet.

To get this right you have to make the process visually debuggable. And while you are at it make processes separately debuggable.

## Fixed Logical Residence Brainstorm

*This brainstorm was written in the context of efforts to program a prototype app 'Circle 3'. It might not be very readable.*

Fixed logical residence was supposed to be not thought through enough to be able to make the first version of Circle 3.

< Binding an object to a specific object reference gives an object a fixed logical residence.

What happens to other references, the unqualified references?

Do they automatically become qualified? Maybe.

If the parent object has a fixed logical residence too, you need a double qualifier to get to the deeper object.

If something has a fixed logical residence, other references will first redirect outward to imaginary references and then redirect inward to the fixed logical residence again. If this inward redirection is out of sight, which it usually is, you have to see an outward directed line with the qualification textually expressed at the outer end of the line.

>

< Do note that fixed logical residence requires qualification, but that does not mean that qualification requires fixed logical residence. You can point to something with a qualifier, even when it does not have its fixed logical residence there. >

## Project Steps

- Look at Computer Language Coding Principles.doc

- Maybe go through all ideas you can find.

- Update article Automatic Containment

- Features to possibly add:

- Implicit connection through parent

- Implicit contents through reference target

- Imaginary reference not created if single real reference already there.

- ~ Only an imaginary reference in a container that adds more references. Not in-between ancestors.

- Imaginary target, logical target, physical target

- Update the whole article

- Adapt the Classes & Relations articles accordingly

Among other things:

- You have to do a line merge between the reference and the referrers.

## Products

- Updated Automatic Containment article

## More Ideas

An option to ignore fixed logical residence would be welcome.

In that case the fixed logical residence might only be pointed out with an empty reference to the imaginary residence. That way automatic containment will work, but publically-accessible classes are still available with just the namespace qualifier. (-> Lower Contents)

Elements of different assemblies seem to intermix too much and parts of one assembly are shown as an intrinsic part of the an assembly that uses it. Things might be better off as better visible as being externally defined.

It is a problem that everything is displayed as composite aggregation, because in some places it works completely counter-intuitive (where you would expect objects to be sibblings, but they are nested instead?).