|  |
| --- |
| Circle Language Spec: Interfaces |

## Member Grouping

Triangles could be used to indicate a benign member grouping:

|  |  |  |
| --- | --- | --- |
|  | 🡪 |  |

Because a triangle basically merges with its container, it is like you are not bothered by the borders of a triangle.

At first glance the grouping does not seem to affect how the objects are used. However, two problems arise:

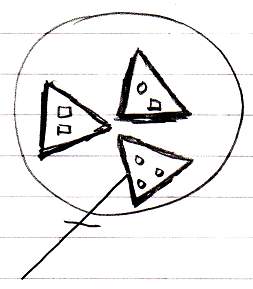
- Explicit interface dependency

- Call overhead

Below these two problems are explained and possible solutions are given. Solutions for these problems are still very much open to discussion.

*Explicit Interface Dependency*

*Existing* functionality may not be affected by member groupings. However, you could make *new* connections directly to the new interface objects.



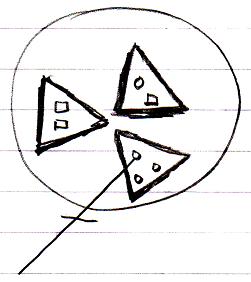
The new connections are now dependent on the member grouping, which was only meant to be a loose definition.

To allow member grouping to be benign, some solution must be found for this. The solution probably lies in the area of access controlling system commands. The member groupings need to be access controlled enough to not be able to explicitly connect to them, but not so much that it makes their members totally inaccessible. Separately access controlling Get For Access and Get For Copy might solve this problem. However, it needs to be thought through if this really only blocks something’s usage as a qualifier. If it does not, probably another split up in system commands will do the trick.

Ideas about the separation between Get For Access and Get For Copy can also be found in the *Black Box* and *System Objects* chapters.

*Call Overhead*

Member grouping may seem benign, but when you approach a member of a group, an Object Get command on the group must be executed before accessing the member, which causes (a small) processing overhead on the call.



The solution could either lie in *optimization*, or making the triangles *imaginary*.

Imaginary elements of a diagram were introduced in the *Automatic Containment* article. Making the member groupings imaginary might be a bad idea, because up until now imaginary elements of the diagram were automatically calculated by the system, and not set by a programmer and it might be wise to keep it that way.

A better solution may lie in optimization. Programmers often add structure to their programs even though it is not absolutely necessary, but just makes things more tangable and orderly. Taking such structurings out of a program before it is run, is more of an optimization issue than anything else.