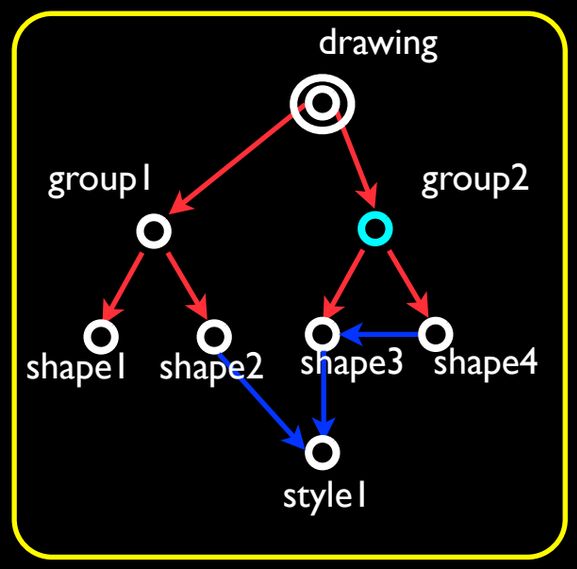
- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root*

*As far as copying is concerned, the distinction between a persistent root and a purely in-memory context is irrelevant. The yellow box indicates a context where all embedded object UUIDs must be unique.

1. copy group2



Copy algorithm is:

- gather the set of objects to copy by following all composite references from the root object to copy

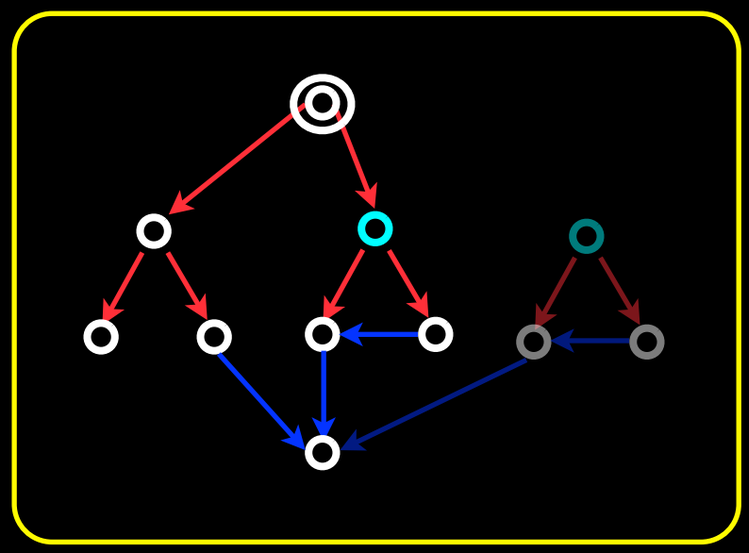
- assign new UUIDs to each object in the copied objects set, and update references and composite references within the copied object set to use the new UUIDs. As a consequence, referenced objects that were not in the copied object set are aliased.

The spirit of it is that you control the copying process by your placement of composite relationships, and hopefully avoid writing any custom code to control copying.

- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

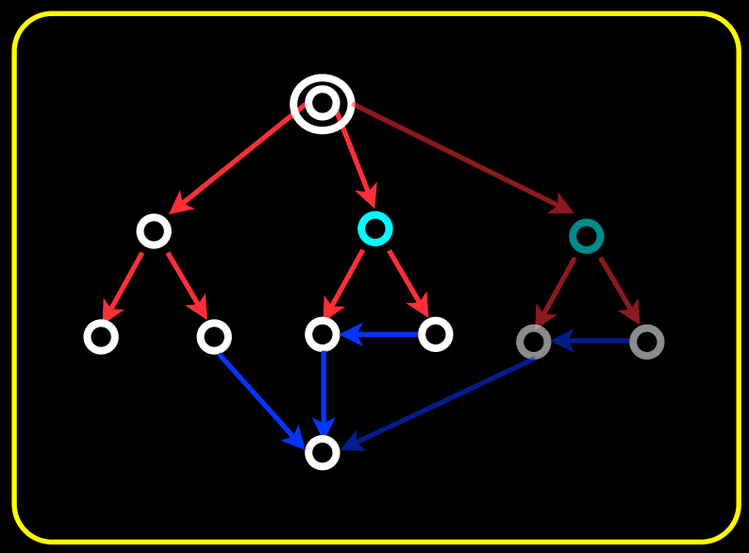
1. copy group2



- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

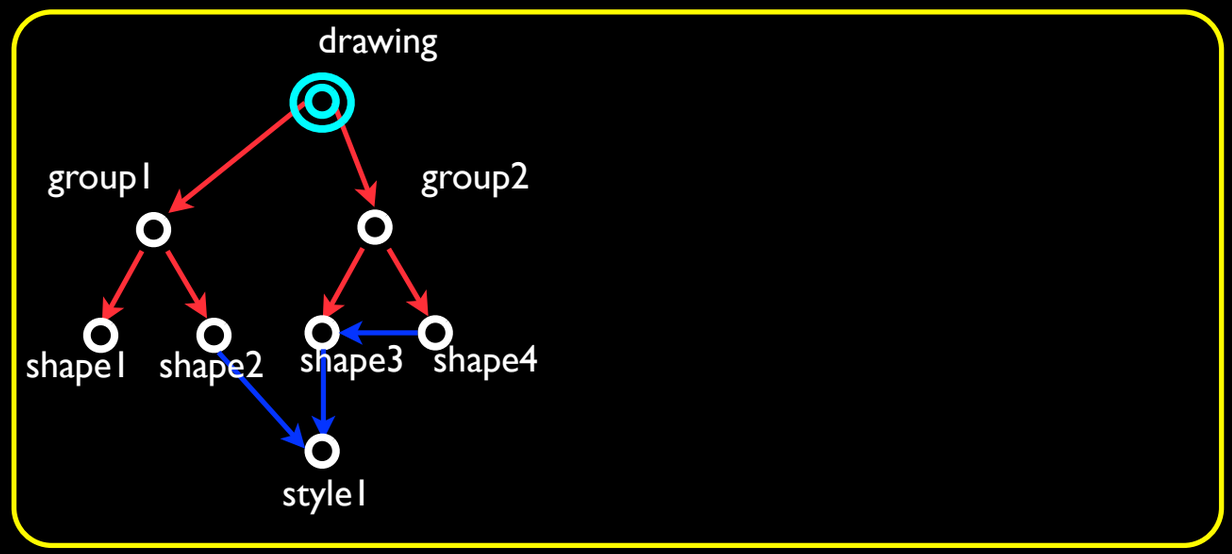
1. copy group2.. and insert into drawing



Root embedded object
Embedded object
Persistent root
Composite reference
Copy reference
Reference
Root object to copy

Copying within a context/ persistent root

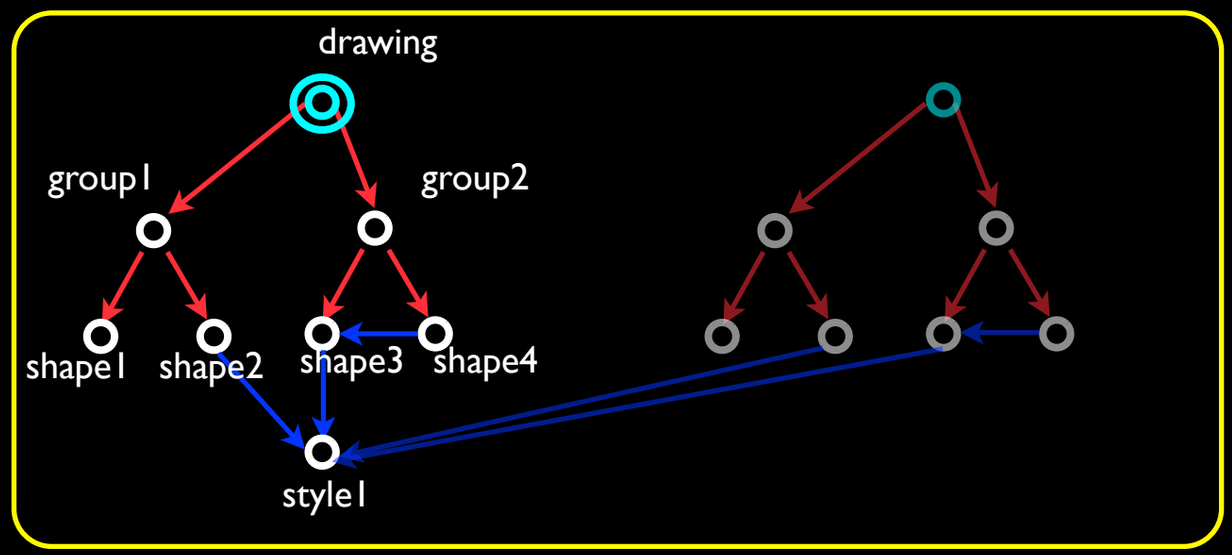
2. copy drawing



- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

2. copy drawing



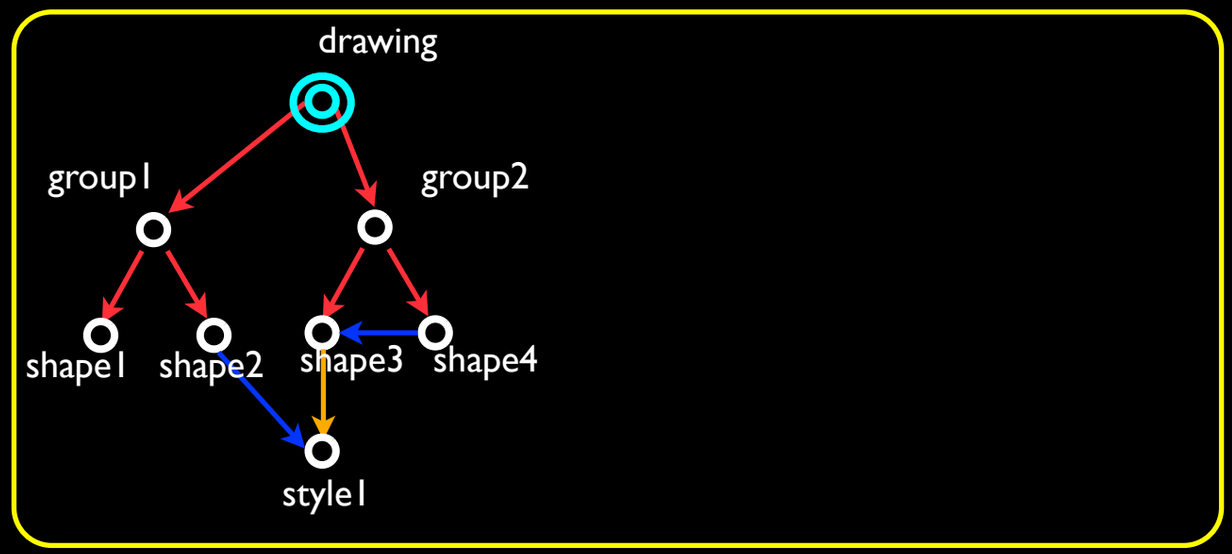
style1 is aliased in the copy of drawing.

This may seem counterintuitive, but if we want it to be copied, we need something in drawing to have a composite reference to it.

- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

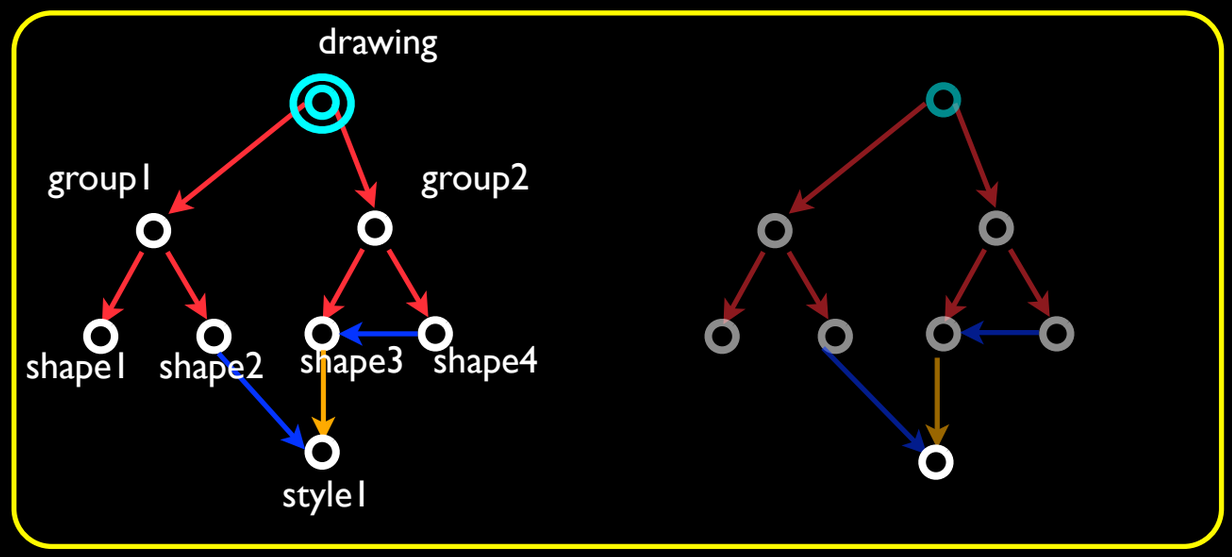
2.b copy drawing, “copy” reference



- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

2.b copy drawing, "copy" reference



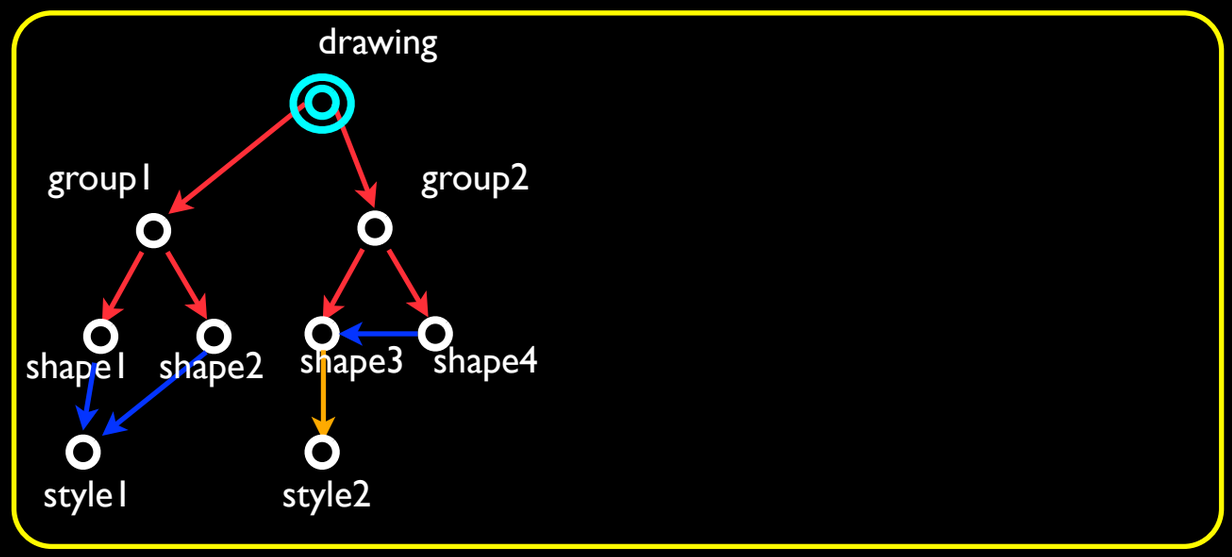
style1 is aliased in the copy of drawing.

This may seem counterintuitive, but if we want it to be copied, we need something in drawing to have a composite reference to it.

- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

2.c copy drawing, “isShared” reference

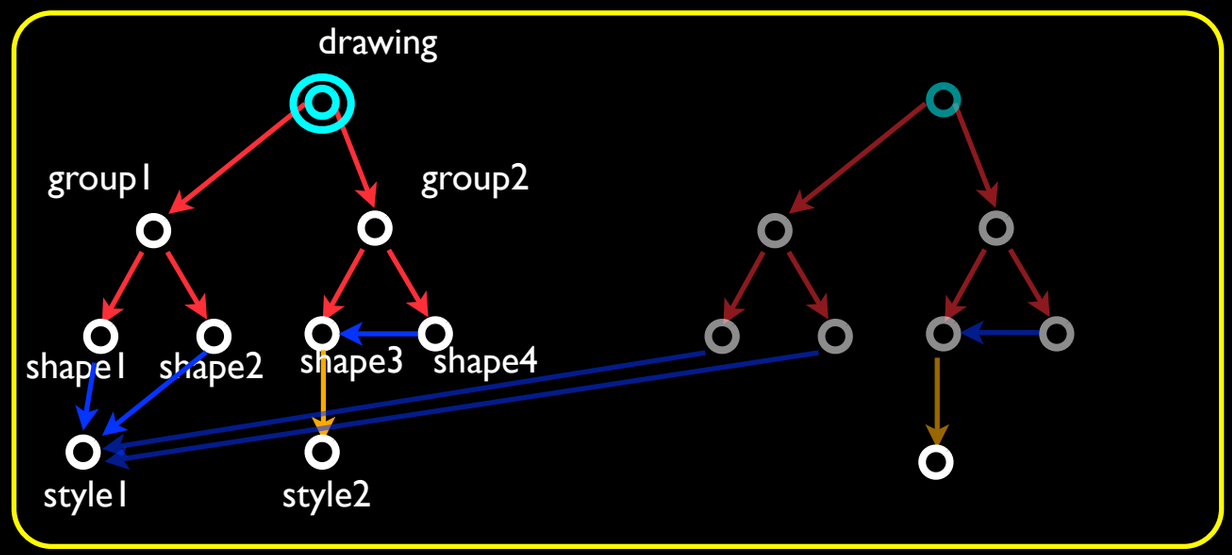


style1 returns YES to -isShared
style2 returns NO to -isShared

- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

2.c copy drawing, “isShared” reference



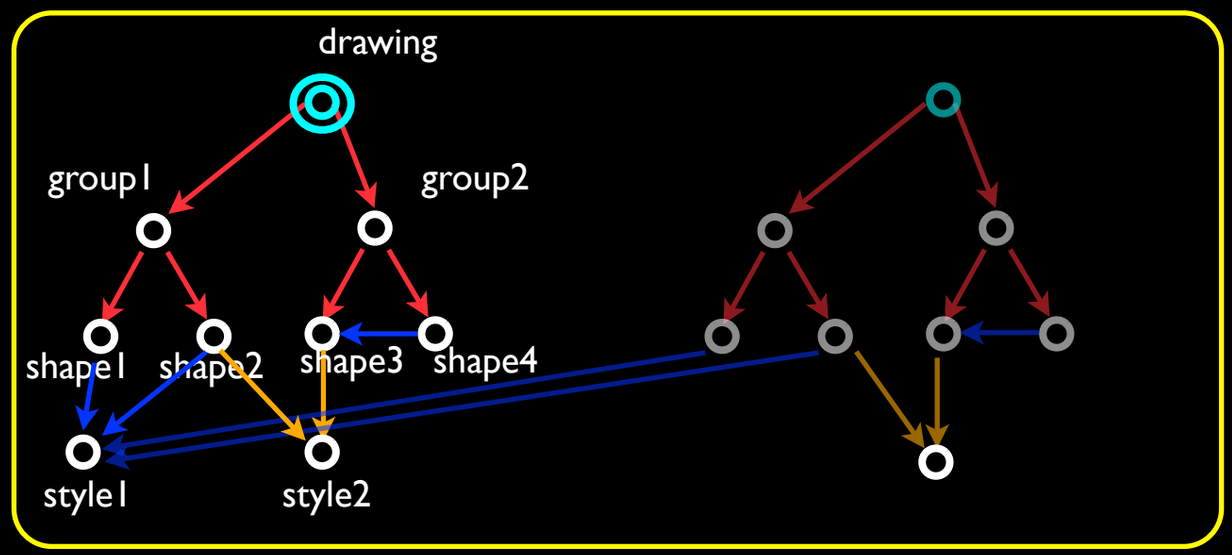
The reference type/color is governed by the object pointed by the arrow, and not by the relationship kind as described in the metamodel.
This makes possible to change whether an object is copied or aliased at runtime per instance.

This could be expressed in the metamodel by adding -[ETEntityDescription isShared] or something similar that describes whether the entity is copied or aliased when present in a relationship.

- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

2.c copy drawing, “isShared” reference



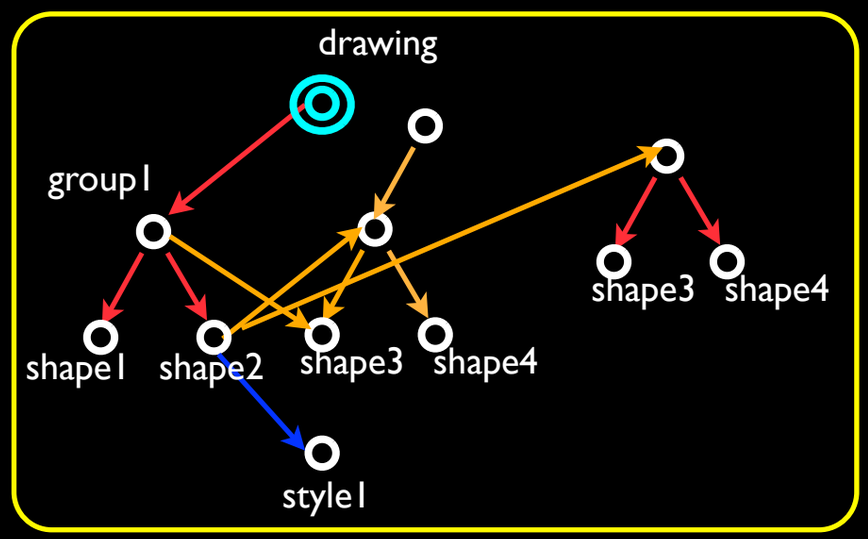
Having two arrows point on style2 is supported in EtoileUI, but almost never occur in practice.

For this border case, the ideal result is shown on the right (not what EtoileUI does currently).

- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

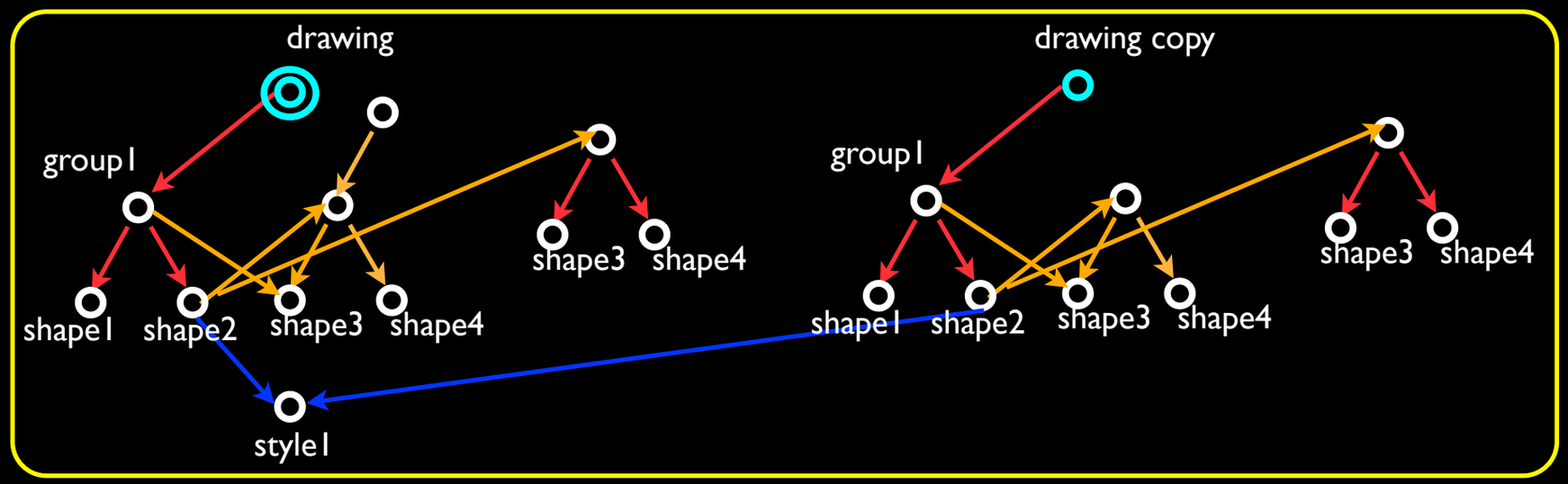
2.c double tree



- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

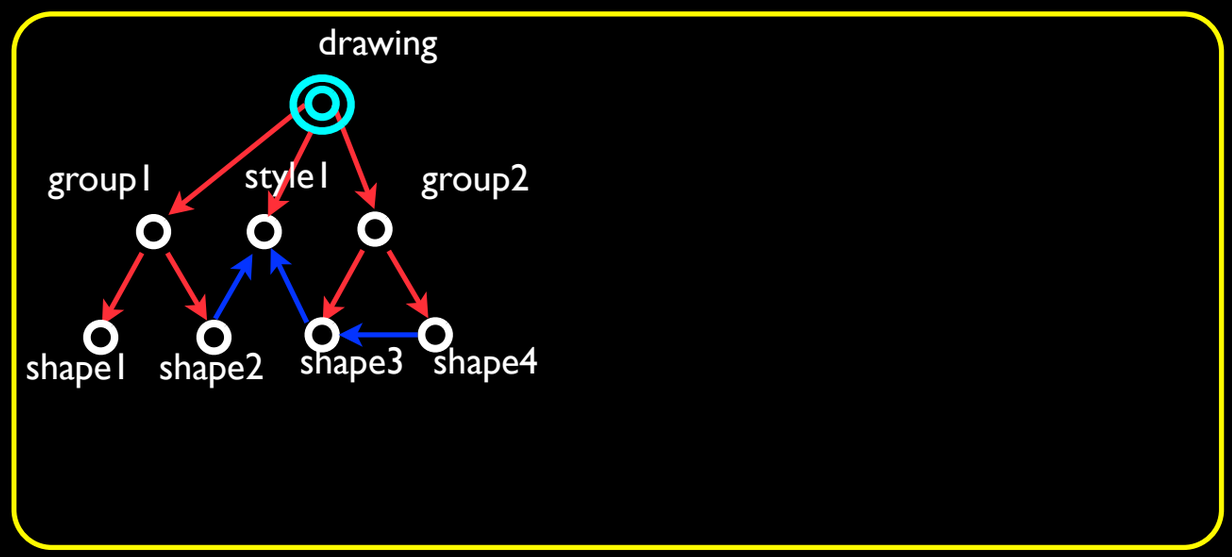
2.c double tree



- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

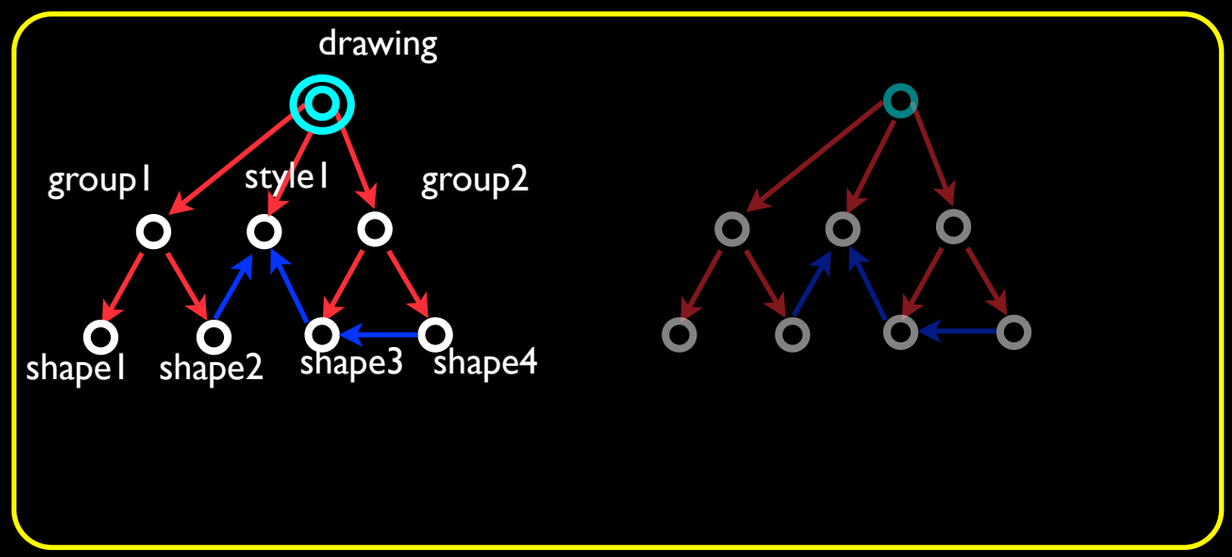
2. copy drawing, including style1



- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

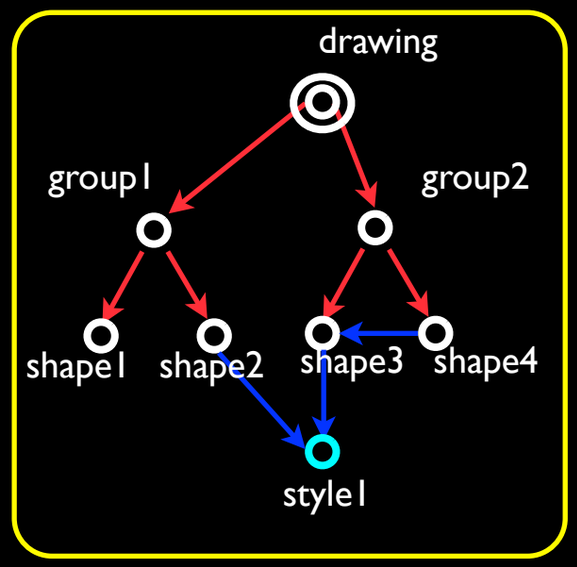
2. copy drawing, including style1



- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

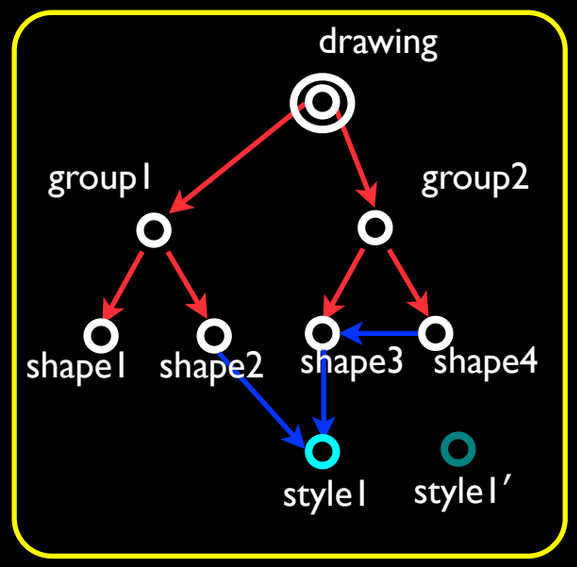
3. copy style1



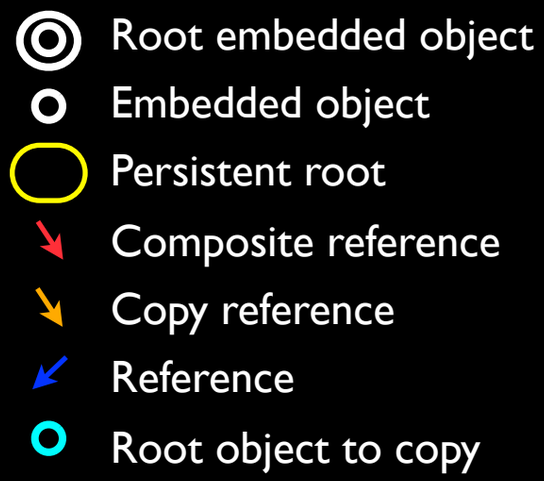
- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying within a context/ persistent root

3. copy style1



References to style1 are not affected by copying it



Copying within a context/ persistent root

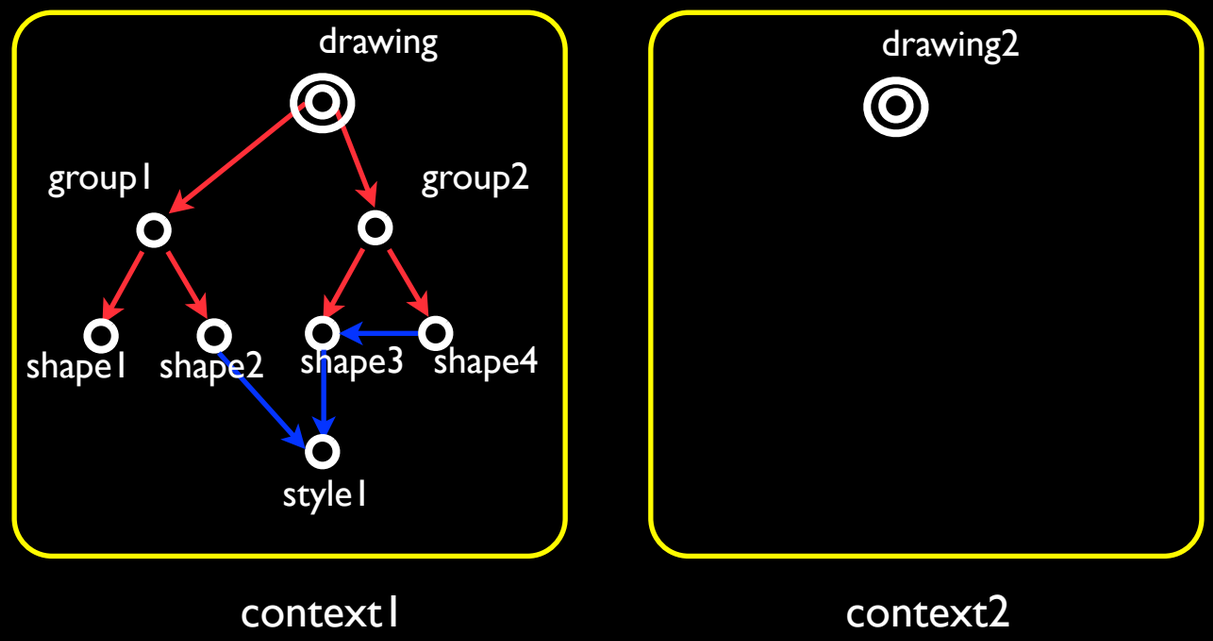
As far as I can see, this copy algorithm is optimal for copying within an editing context/persistent root - it's as simple as possible but also handles every case I can think of.

However, copying between contexts is not as straightforward and a more complex procedure may be needed with custom copy code.

- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying across contexts

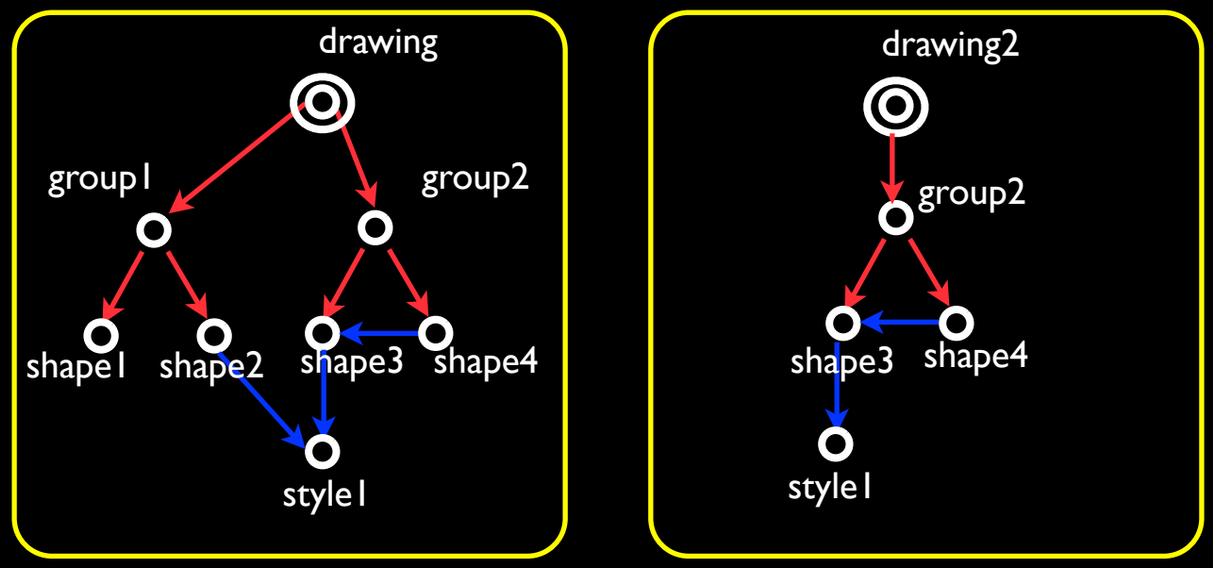
4. copy group2 into context2



- Root embedded object
- Embedded object
- Persistent root
- Composite reference
- Copy reference
- Reference
- Root object to copy

Copying across contexts

4. copy group2 into context2



(desired result?)

There are a few complications with cross-context copying:

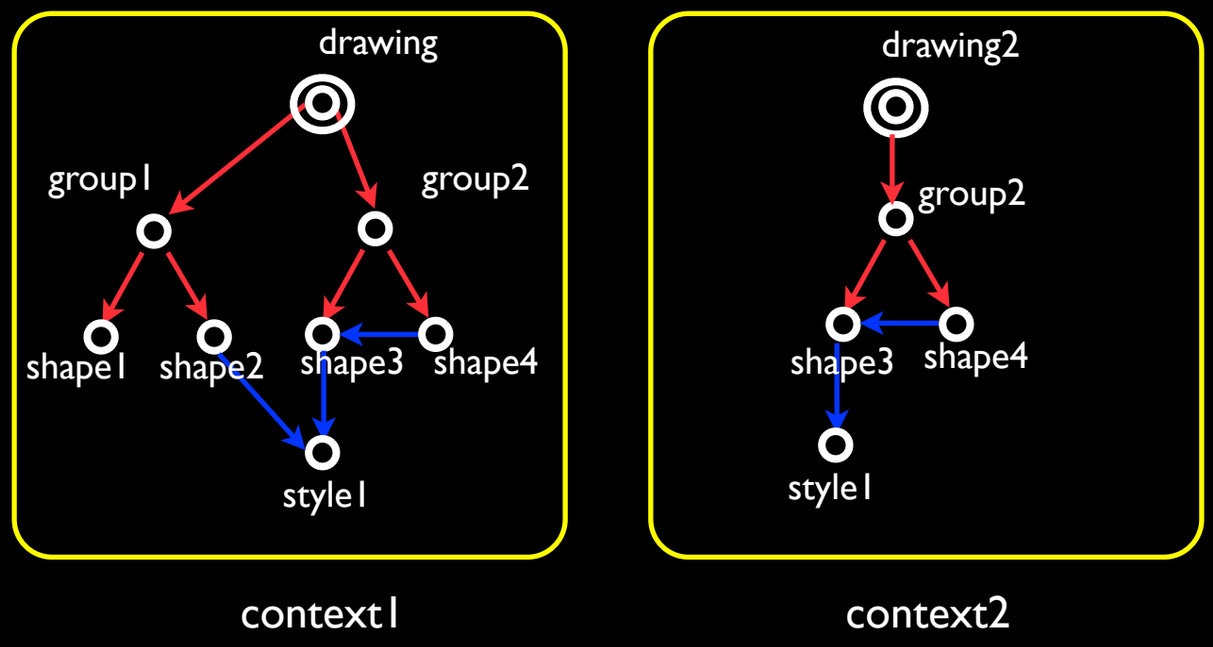
1. we don't necessarily need to rename the copied objects, since embedded object UUIDs don't need to be unique across contexts.

2. If you used the initial algorithm I described, you would end up not copying style1

Root embedded object
Embedded object
Persistent root
Composite reference
Copy reference
Reference
Root object to copy

Copying across contexts

4. copy group2 into context2



Proposed modified algorithm:
- When copying to a different context, continue to rename the copied objects
even though it's not strictly necessary, for the sake of consistency with regular embedded object copying
- Unlike when pasting a copy into the same context from which it was copied,
when you know that the non-composite referenced objects will be there, you don't
know if they will be there when pasting into another context.
- Consider 2 cases:
    1. when pasting group2 into drawing2, style1 may have been already there
    2. style1 was not there.
- Let's say that having a broken reference to style1 in the copy is unacceptable,
because we require the copy to be fully working.
-We _could_ create a cross-persistent root reference back to the original context's style1.
- My preference is, when gathering the set of objects to copy, for each reference, check if the referenced object
is in the destination context (usually won't be, unless the dest context was a copy of the original). If it isn't,
include the referenced object in the set of objects to copy, even though it's not a composite reference
~~- Or, simpler: when copying across contexts, always copy all composite referenced as well as regular referenced objects~~
~~i.e. don't make an attempt to automatically join the object graph being copied with the existing objects in the destination context,~~
~~whereas we would when copying within the same context.~~