A note on call-by-visit in the BC-Emerald implementation and break-even point.

Eric Jul, 2018-03-08 – revised 2023-03-09

call-by-visit and call-by-move do NOT work properly in the Emerald implementation that we are using.

Here is a work-around:

As `call-by-move` and `call-by-visit` essentially are optimized versions of what a programmer can program anyway, you can simulate `call-by-move` in the following manner, for example, given the call:

```
        callee.call[move testdata]
```

replace this by the following to achieve the effect of `call-by-move`:

```
        move testdata to callee
        callee.call[testdata]
```

Doing `call-by-visit` is more difficult because we want to ask the called object to move the parameter back, BUT the called object does NOT know from where it was called. We therefore add a parameter, `returnTo`, to the call. The parameter tells where to send the data object back to which is the `caller` object. So replace

```
        callee.call[visit testdata]
```

by

```
        callee.call[testdata, self]
```

And modify the `callee` object, *e.g,*, here in the original form:

```
const callee <- object callee
    export operation f[data: DataType] -> []
        data.update[]
    end call
end callee
```

and here with the "manual" return of the parameter object:

```
const callee <- object callee
    export operation f[data: DataType, returnTo: Any] -> []
        data.update[]
        move data to returnTo
    end call
end callee
```

So find the break even point using the "simulated" `call-by-visit` as outlined above.