BabySRE,
Squeak Reverse Engineering

Trygve Reenskaug

Abstract

SRE (Squeak Reverse Engineering) are three tools making Squeak objects visible and tangible.
1) *SRE collaboration* lets me diagram a Squeak objects with the links between them. With this tool, I find that I can master larger systems than I could without it
2) *SRE Object Browser* gives me a complete description of a single object with its identity, state, and behavior. I can edit its state as in an Inspector and its behavior as in a class Browser. The class hierarchy is flattened so that I see the object as a whole.
3) *Object>>traceRM:* is a method that dumps the stack on to the Transcript.

Taken together, these tools put me in closer contact with the real objects of the image, thereby giving me better understanding of existing systems and better contol over new ones.

Document History

2004-12-19: First release: BabySRE-TRee.11
2005-01-13: Updated for release: BabySRE-TRee.34

1 USING REVERSE ENGINEERING TO EXPLORE EXISTING OBJECTS

The Squeak inspector is a powerful tool for investigating an object. It is easy to open new inspectors on its linked collaborator objects and thereby build a picture of the system. The downside is that the screen (and my brain) quickly get cluttered with inspectors. I might even have several inspectors on the same object without realizing it.

This last problem is somewhat remedied by adding identity information to Object>>printOn: so that it prints, e.g., **[1622] world** *: PasteUpMorph.* (This notation is somewhat UMLish. It denotes an object with oop=**1622** and name **world**. It is an instance of the *PasteUpMorph* class).

But I still have the problem of many inspectors. A new tool, *SRE Collaboration*, lets me diagram a system of interlinked objects. I find that it helps me master larger systems than I could master without it.

Real Squeak objects are very rich and I find it illuminating to distinguish between different projections of a system of objects. In the documentation on the following pages, I distinguish between three projections that express a particular separation of concerns:

1. *Domain collaboration.* An SRE Collaboration that shows the objects that together realize my domain.
2. *Object descriptor.* An SRE Collaboration that shows how the features of an object are determined by the object's class and the superclasses of that class.
3. *Instantiation.* An SRE Collaboration that shows how every object is an instance of a class. In this projection I plot the object, its class, the class of this class, etc.

Each of these projections shows the truth and nothing but the truth. None of them shows the whole truth.

The second projection is interesting in that it highlights the nature of a Squeak object. It is a cliché to say that an object has identity, state and behavior. The *SRE Object Browser*, shows an object's identity, state and behavior in a single window - very illuminating. The top row in this browser shows the object's attributes together with the value of the selected one (as in the Inspector). The second row thows the object's methods together with the code of the selected one. The third row shows two multiple select lists; a superclass list and a list of method categories. Both filter what is shown in the attribute and method lists.

The *SRE Collaboration* tool helps me understand the static structure of my system of objects. The *SRE Object Browser* tool helps me understand the nature of a particular object. A third tool, *traceRM,* helps me understand the dynamic interaction. The most important is Object>>traceRM:levels:. It gives me a dump of the stack in the Transcript. I hope it can be replaced with more sophisticated tools in the future. (For example by putting an ObjectTracer around every object in a given collaboration).

In the following, I will describe the reverse engineering of a simple example. You may, as I do, find it boring reading. But is does illustrate what you can expect if you install the SRE tools in your own image. I find I frequently use them when I wander into uncharted parts of a system. It is so much easier to read the code when I understand the relationships between the objects.

The development of the SRE package is part of the *BabyUML* project where I look for higher level constructs for object oriented programming (as opposed to class oriented programming). [ECOOP-04]The vision is that the SRE tools may evolve into being projections of the program itself where the current class code is just one of several projections.

I believe the SRE tools are quite usable as they are. Their main purpose is to experiment with what it takes to work with objects. I plan to continue this experimentation, and do not contemplate to maintain the SRE tools as regular Squeak products.

2 THE DEMO PROGRAM

For the purposes of this study, I created a small class that defines a colored ellipse on the screen. This ellipse cycles through a sequence of different colors. The class definition is as follows:

```
EllipseMorph subclass: #DemoEllipseSRE
        instanceVariableNames: 'colorIndex'
        .....
```

There are two methods:

```
step
	self color: self nextColor.

nextColor
	| colors |
	colors := {Color red. Color green. Color blue. Color magenta. Color yellow.}.
	colorIndex ifNil: [colorIndex := 1].
	colorIndex := colorIndex + 1 \\ colors size + 1.
	^colors at: colorIndex.
```

The object under study is opened with the statement

```
DemoEllipseSRE new openInWorld.
```

and I observe a small ellipse that regularly changes its color.

3 DOMAIN COLLABORATION SHOWING DOMAIN OBJECTS AND LINKS

Appendix 2: on page 10 describes how I created the domain collaboration shown overleaf. The focus of attention is the ellipse object I created when I executed

```
DemoEllipseSRE new openInWorld.
```

This is an example of the simplest kind of morph. It is owned by [1622] world, a morph that controls the whole Squeak screen and has all the visible elements as submorphs. (world is an instance of PasteUpMorph. Its Balloon help says that it is a morph whose submorphs comprise a paste-up of rectangular subparts which "show through").

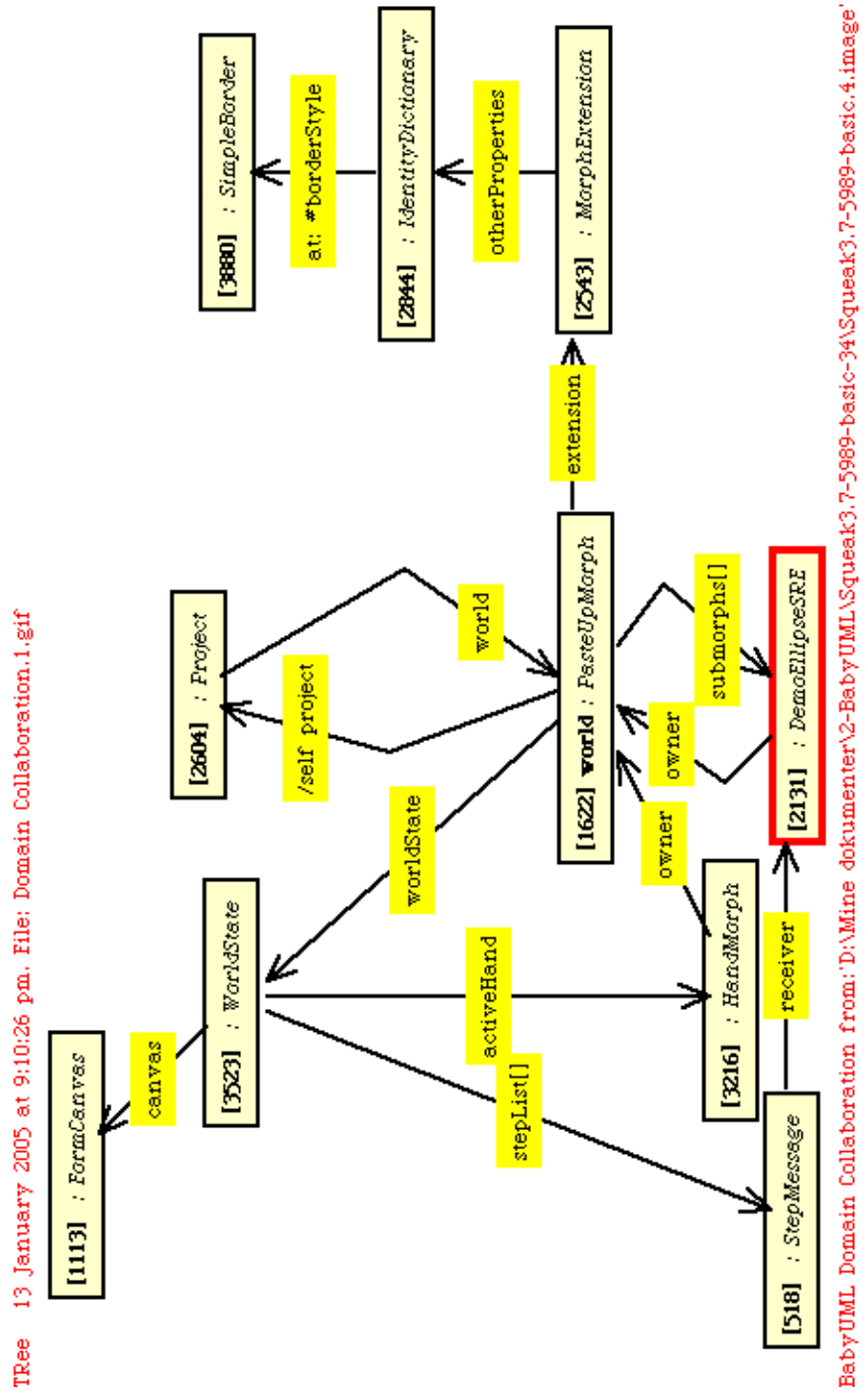
I see that world has a link named extension holding a SimpleBorder. The extension appears to be private to world, I think of it as being encapsulated by the world object.

The world also has a link named worldState that holds an activeHand for input, a damageRecorder for selective display, a stepList for stepping animated morphs, and a canvas that is a representation of the display screen.

The world also has a project method (called a derived attribute in UML). This project has a link back to its world.

The WorldState's activeHand is a morph that represents the cursor within the world. The hand is a submorph of world. (The hand's submorphs hold anything being carried by dragging. The owner of the dragged morph is temporarily set to the hand and returned to the original owner when the dragging is completed. This is very dynamic and hard to diagram at the instance level. It could be described in a role model, the same kind of diagram where the objects are represented by the role they play in the context).

This gave me the inspiration to seek out some other objects shown in the diagram: :



After having drawn most of the diagram, I wanted to find out how and from where the *step* method was called. I augmented the step method with a *stack trace:*

```
DemoEllipseSRE>>step
        self doOnlyOnce: [self traceRM: self levels: 50].
        self color: self nextColor.
```

The result was the following trace of the stack in the Transcript:

```
1  [2131] : DemoEllipseSRE >> step {[2131]a DemoEllipseSRE(2131)}
2  [2131] : DemoEllipseSRE >> doOnlyOnce:
3  [2131] : DemoEllipseSRE >> step
4  [2131] : DemoEllipseSRE >> stepAt:
5  [518] : StepMessage >> value:
6  [3523] : WorldState >> runLocalStepMethodsIn:
7  [3523] : WorldState >> runStepMethodsIn:
8  [1622] : PasteUpMorph >> runStepMethods
9  [3523] : WorldState >> doOneCycleNowFor:
10  [3523] : WorldState >> doOneCycleFor:
11  [1622] : PasteUpMorph >> doOneCycle
12  [2445] : Project class >> spawnNewProcess
13  [3684] : BlockContext >> newProcess
```

I find that find that the step method is called from [518] StepMessage. This object is not in the diagram, so I right-click the diagram background and select add role from expression... . I then type

```
StepMessage allInstances detect: [:obj | obj asOop = 518]
```

Wonders! I now get this object in the diagram. I put the receiver link from this object to [2131]:DemoEllipse. The above stack trace shows that the StepMessage was called from [3523]:WorldState. I try the add link from variable... command and find that the link is in the stepList collection attribute. Further reading of code divulges that the stepList is a collection holding a StepMessage for every morph that is stepping. This causes the step methods to be called at regular intervals, causing my [2131] : DemoEllipseSRE to cycle to the its next color.

My next quest was to find out how the ellipse is being redrawn. All I have done myself is to set self color: self nextColor in the step method. I added a trace statement to the draw method:

```
DemoEllipseSRE>>drawOn: aCanvas
        self doOnlyOnce: [self traceRM: self levels: 50].
        super drawOn: aCanvas.
```

The Transcript dump of the stack was as follows:

```
1  [2131] : DemoEllipseSRE >> drawOn: {[2131]a DemoEllipseSRE(2131)}
2  [2131] : DemoEllipseSRE >> doOnlyOnce:
3  [2131] : DemoEllipseSRE >> drawOn:
4  [2261] : FormCanvas >> draw:
5  [2261] : FormCanvas >> drawMorph:
6  [2131] : DemoEllipseSRE >> fullDrawOn:
7  [2261] : FormCanvas >> roundCornersOf:in:during:
8  [2261] : FormCanvas >> roundCornersOf:during:
9  [2131] : DemoEllipseSRE >> fullDrawOn:
10  [2261] : FormCanvas >> fullDraw:
11  [2261] : FormCanvas >> fullDrawMorph:
```

12 [3523] : WorldState >> drawWorld:submorphs:invalidAreasOn:
13 [3332] : Rectangle >> allAreasOutsideList:startingAt:do:
14 [3332] : Rectangle >> allAreasOutsideList:do:
15 [3523] : WorldState >> drawWorld:submorphs:invalidAreasOn:
16 [1125] : Array >> do:
17 [3523] : WorldState >> drawWorld:submorphs:invalidAreasOn:
18 [3523] : WorldState >> displayWorld:submorphs:
19 [1113] : FormCanvas >> roundCornersOf:in:during:
20 [1113] : FormCanvas >> roundCornersOf:during:
21 [3523] : WorldState >> displayWorld:submorphs:
22 [1622] : PasteUpMorph >> privateOuterDisplayWorld
23 [1622] : PasteUpMorph >> displayWorld
24 [3523] : WorldState >> displayWorldSafely:
25 [392] : BlockContext >> on:do:
26 [392] : BlockContext >> ifError:
27 [3523] : WorldState >> displayWorldSafely:
28 [3523] : WorldState >> doOneCycleNowFor:
29 [3523] : WorldState >> doOneCycleFor:
30 [1622] : PasteUpMorph >> doOneCycle
31 [2445] : Project class >> spawnNewProcess
32 [3684] : BlockContext >> newProcess

I still haven't discovered how the change of color causes the display of the area covered by the ellipse. It is probably something involving [3523]:worldState>>damageRecorder, but I ended my quest at this point.

4 OBJECT DESCRIPTOR COLLABORATION AND BROWSER

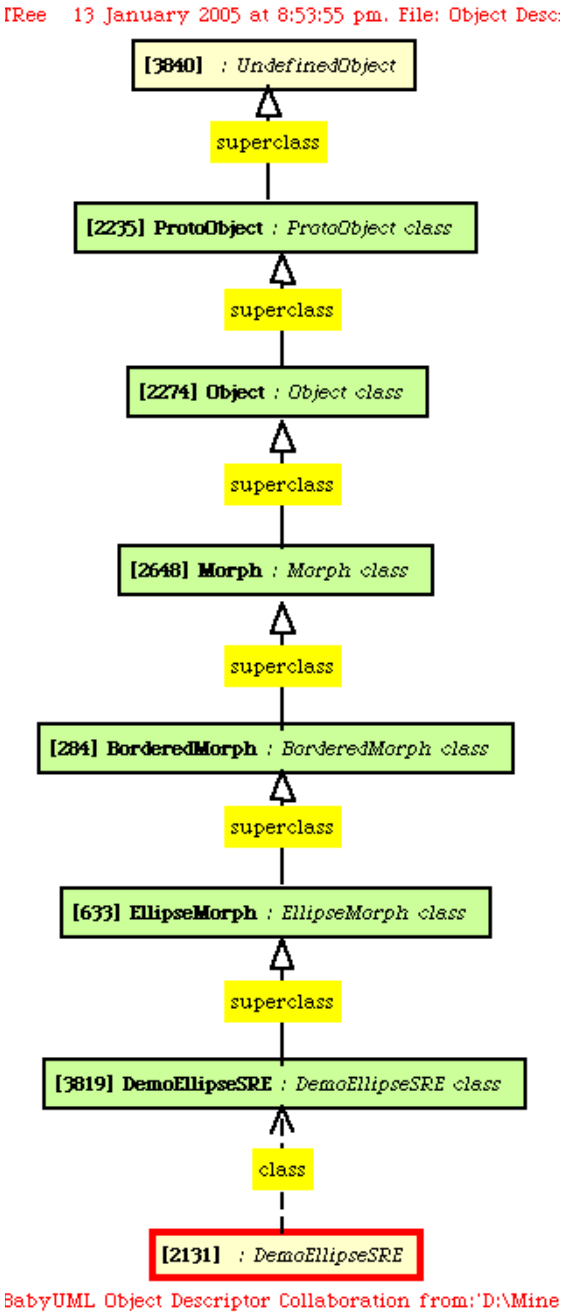
The properties of an object are given by its class and the superclasses of that class. The relevant objects are shown in the diagram on the left.

[2131] : DemoEllipseSRE is an instance of
[3819] DemoEllipseSRE is a subclass of
[633] EllipseMorph etc.

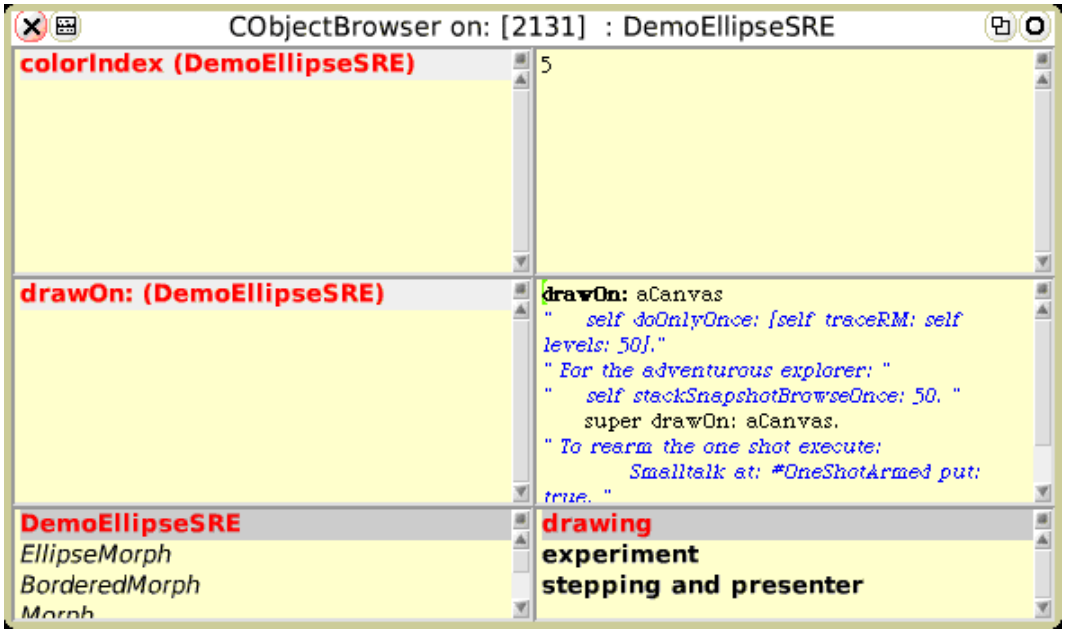
It is interesting to note that the objects shown in this diagram taken as a whole exactly describe this particular object. It is also interesting to note that the defining classes in all probability belong to several packages.

It is tempting to create an editor that at a glance will give a complete descriptor of an object. An early attempt is is the SRE object browser shown below.

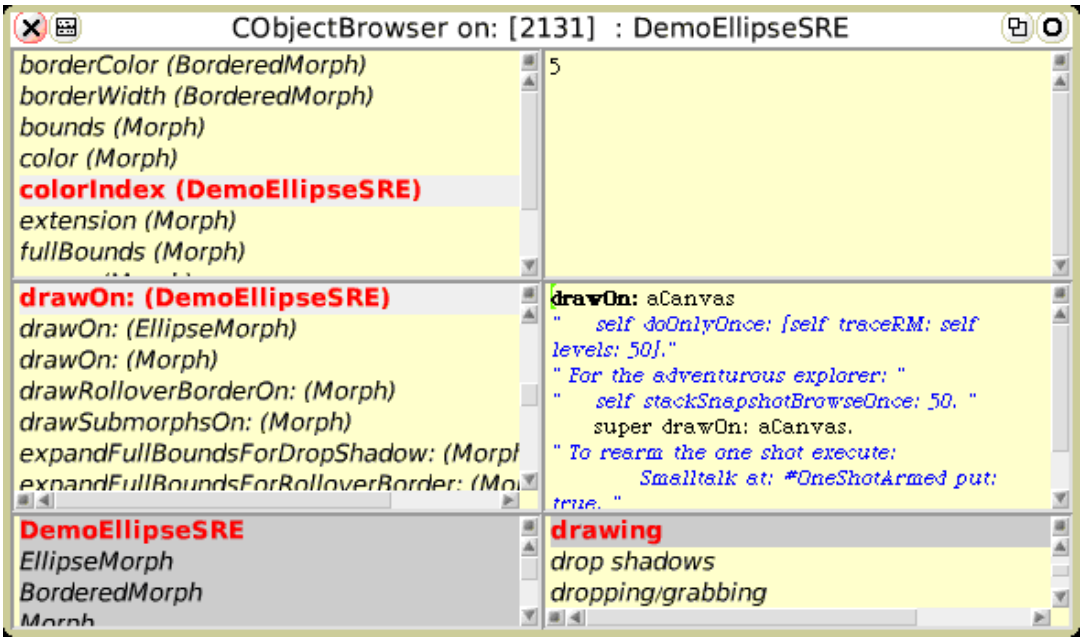
Note: Sometimes, it is useful to extend the name of the object with the name of the class. At other times, this can be very confusing as illustrated in this diagram. May be the default should be that the class name should only be shown for anonymous objects without a name on their own.



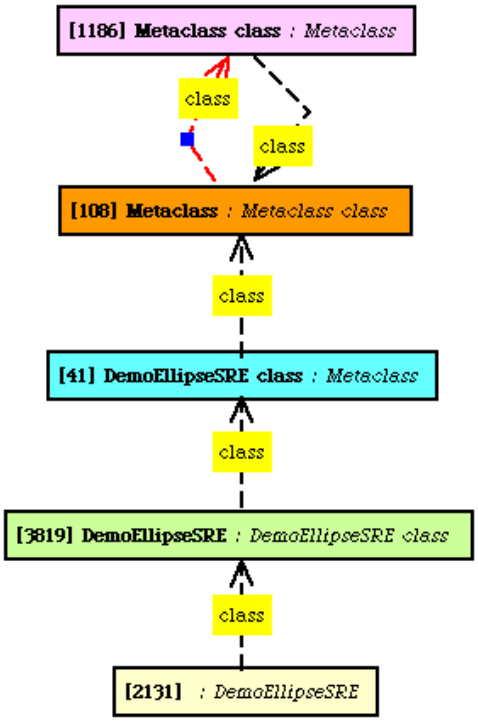
The top row shows the object's instance variables; the value of the selected variable is shown and can be edited on the right. The middle row shows the object's behavior. Selectors on the left; the code of the selected method to the right. The bottom row contains two multiple selection list filters (PluggableListMorphOfMany), superclasses on the left, method categories on the right..



This browser gives a complete picture of the object. In the snapshot shown below, all superclasses and one category have been selected. Notice the three drawOn: methods in the superclass chain.



5 INSTANTIATION COLLABORATION



byUML Instantiation Collaboration from:'D:\Mine dc

Every object is an instance of a class. A class object is likewise the instance of a class. (Classes whose instances are classes are often called metaclasses). The metaclass is also an instance of a class -- does it ever stop? The Smalltalk architecture is given in some complex diagrams in the [Blue book] (on pages 270-272 in my copy). Subclassing and instantiation are independent mechanisms and can best be studied in separate projections. The chain of instantiation for the [2131] : DemoEllipseSRE object is shown on the left.

Quite simple, but one can easily get lost by confusing instantiation with inheritance. It can be instructive to create a collaboration showing instantiation and inheritance in the same diagram. It gets wonderfully complicated without adding any interesting insights.

In particular, the metaclass [41] DemoEllipseSRE class, is not part of the ellipse object's descriptor. The squeak-dev mailing list sometimes shows questions arising from a confusion of a class with its metaclass. I am speculating that the metaclass object could be described as a regular object in the system design (sometimes called a *factory object*). May be we don't need one for every class; several classes could share a common factory object.

6 CONCLUSION

The diagrams and stack dumps shown here are all created with the help of the BabySRE tools. They augment the traditional browsers and inspectors when we need to understand and possibly document an existing system The object browser can also be used to modify the system; its user can change the values of instance variables, add new instance variables, define and redefine methods in the object's immediate class. (Modifications of the superclasses has been blocked because this seems too dangerous to be contemplated in the context of a single object).

It is tempting to go a step further and to explore if system projections that resemble the BabySRE diagrams could become part of the program itself. In particular, anything that has to do with object interaction could be moved out of the class browser and into some kind of collaboration browser. The system description would then put constraints on what could be specified in the regular class methods.

My focus on objects opens a vista of structuring objects by enclosing them in *components*, that is objects with encapsulated objects that are invisible from the outside of the component.

In the meantime, I have on many occations found that the BabySRE tools help me better understand programs written by myself and by others.

7 ACKNOWLEDGEMENTS

Many thanks to Ned Konz for the Connectors package. Also for his permission to copy/rename the classes I use from this package so that the evoplution of BabySRE becomes independent of the evolution of Conntectors the package.

Sincere thanks to Milan Zimmermann and Chris Muller for excellent and very useful suggestions that have been realized in he second version of BabySRE.

Appendix 1: Installation

1. The image I started from was Squeak3.7-5989-basic.
2. I gave World Menu command open>>SqueakMap Package Loader.
3. I used this loader to load the latest version of BabySRE.
   - 'The package has no published release for your Squeak version, try releases for any Squeak version?' answer yes.
   - 'The package has no published release at all, take the latest of the unpublished releases?' answer yes.
4. To reproduce the example described above, I executed

   ```
   DemoEllipseSRE new openInWorld.
   ```

   and observed that the ellipse with cycling colors appeared in the top left corner.

NOTE: An earlier version required that Connectors were installed before BabySRE. This is no longer necessary because a copy of the required parts of Connectors is now included in the BabySRE package.

Appendix 2: How I created the domain collaboration

1. I pointed to the ellipse and pressed the left button with the ALT-button (Windows) down to open the halo. I opened the debug menu in the white button on the right hand edge. I selected SRE collaboration to open a collaboration diagram on the morph.
2. I placed the object [[2131]] a DemoEllipseSRE in the diagram.
3. I right-clicked on the diagram background (the 'playfield') and selected the menu item Rename diagram... . I typed Domain Collaboration.
4. I right-clicked on the role symbol marked [2131] : DemoEllipseSRE and selected the menu item add link for variable.... This gave a new menu. I selected the owner item. I placed the resulting [1622] world.
5. Right-clicking [1622] world, I added the link named submorphs[2131].
6. Again right-clicking [1622] world, I placed extension, then otherProperties, #borderStyle, and #worldState.
7. Right-clicking [3523] : WorldState, I placed #canvas and a#ctiveHand.

8. Right-clicking [1622] world, I selected add role from expression., typed 'self project' and placed [2604] a Project. I then linked this project back to [1622] world.
9. I right-clicked [3523] : World State, selected add role from expression, typed
   stepList detect: [:sm | sm receiver asOop = 2131]
   I then placed [518] : StepMessage. (I had to look around a bit to discover this statement. An alternative way for finding this object is given in the body of this note.)
10. I right-clicked [518] : StepMessage, selected add link for variable and chose receiver. This created the link from stepMessage to [2131] : DemoEllipseSRE.

Appendix 3: How I created the object descriptor collaboration

1. I pointed to the ellipse and pressed the left button with the ALT-button (Windows) down to open the halo. I opened the debug menu in the white button on the right hand edge. I selected SRE collaboration and was given the choice of using one of the existing diagrams or a new one. I selected new.
   I placed the object [2131] : DemoEllipseSRE in the diagram.
2. I right-clicked on the diagram background (the 'playfield') and selected the menu item Rename diagram... . I typed Demo descriptor.
3. I right-clicked [2131] : DemoEllipseSRE and selected the menu item add link for variable' -> class. I placed **[3819] DemoEllipseSRE** *: DemoEllipseSRE class*
4. I right-clicked [3819], selected add link for variable -> superclass. Placed [633] EllipseMorph.
5. Continued up the superclass chain to [22359] ProtoObject and [3840] : UndefinedObject.

Appendix 4: How I created the instantiation collaboration

1. I pointed to the ellipse and pressed the left button with the ALT-button (Windows) down to open the halo. I opened the debug menu in the white button on the right hand edge. I selected SRE collaboration and was given the choice of using one of the existing diagrams or a new one. I selected new.
   I placed the object [2131] : *DemoEllipseSRE* in the diagram.
2. I right-clicked on the diagram background (the 'playfield') and selected the menu item Rename diagram... . I typed Demo instantiation.
3. I right-clicked [2131] : *DemoEllipseSRE* and selected the menu item add link for variable'-> class. I placed **[3819] DemoEllipseSRE** *: DemoEllispeSRE class* in the diagram.
4. I right-clicked [3819], selected add link for variable -> class and placed **[41] DemoEllipseSRE class** *: Metaclass*.
5. I continued up to the root classes, [108] Metaclass and [1186] Metaclass class.

References

[ECOOP-04] Trygve Reenskaug: *Empowering People with BabyUML: A sixth Generation Programming Language*. Opening talk, ECOOP 2004, Oslo. http://heim.ifi.uio.no/~trygver/2004/ECOOP-04/EcoopHandout.pdf

[Blue book] Goldberg and Robson: Smalltalk-80. The language and implementation. Addison-Wesley 1983. ISBN 0-201-11371-6

[Kay77] Kay, Alan, *Microelectronics and the Personal Computer,* Scientific American, September 1977, p. 244.

[Pawson04] Richard Pawson: *Naked Objects*. PhD thesis, Trinity College, Dublin, 2004.

[Norman88] Donald A. Norman: *The Design of Everyday Things.* Doubleday, 1988

About the author

Trygve Reenskaug , prof.em.,
Department of informatics,
University of Oslo, Norway.

mailto: trygver <at> ifi.uio.no
http://heim.ifi.uio.no/~trygver