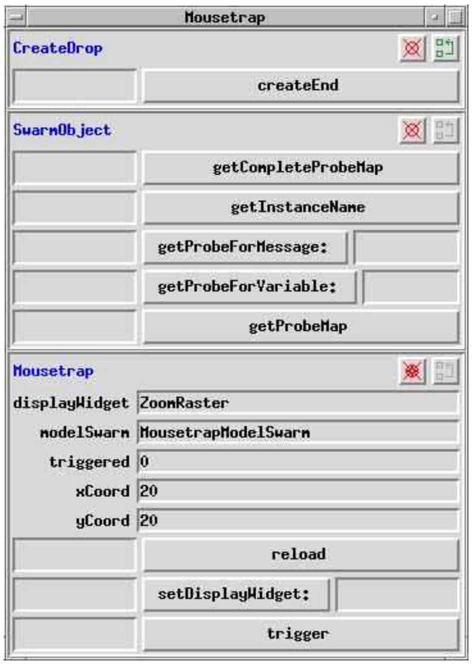
Probing and Displaying the Contents of a SwarmObject

Contents:

Overview



The Framework

Probes, ProbeMaps and ProbeDisplays allow the user to dynamically interact with the objects in their simulation. By *interaction* we mean reading/setting instance variables as well as dynamically generating method calls. The main point being that these interactions are not hardwired into the program code, but occur due to user-generated requests, mainly through the provided GUI.

The key to this capability is the Probe: in general a probe takes an object and either extracts the value of a specific variable, or calls a specific method. For this purpose we provide two subclasses: VarProbe and MessageProbe.

There are two main uses for probes: they can be fed into data-collection objects and serve as interfaces to the objects about which data is being collected (thus keeping the data-collection objects as general as possible) - the Averager class, for example, directly subclasses MessageProbe. Or, they can be used in order to generate a GUI to the individual objects in the simulation (the more common usage).

In order to generate a graphical version of a given probe, the programmer must place it inside a ProbeDisplay, which will automagically generate a window with the appropriate interface. Since, more often than not, the programmer will want to generate windows with more than one variable, we have designed the ProbeDisplay to deal with ProbeMaps rather than individual probes. Thus, by generating a ProbeMap containing exactly the right probes, a user is able to customise the window generated by the ProbeDisplay.

In order to facilitate the creation of all these different objects, the Swarm kernel provides some alternative methods for Probe/Map/Display generation:

- Direct Generation - which leaves the user in charge of all the details.
- Object Generation (Probes/ProbeMaps) - where we use method calls (defined in SwarmObject) on a given target object to generate the Probe/ProbeMap for that object. This allows the object to be aware that it is being probed and to control directly the contents of the Probe/ProbeMap being requested...
- Library Generation (Probes/ProbeMaps) - where the programmer "checks out" a unique, shared copy of a Probe/ProbeMap from the probeLibrary object (of class ProbeLibrary) provided by the kernel. By shared we mean that a similar request made at a different point in the code, will return a reference to the very same probe instance.
  Note that, by default, Object Generation is equivalent to Library Generation since the default behaviour of SwarmObjects, when requested to create a ProbeMap, is to return a reference to the probeLibrary's unique copy!!!
- ProbeDisplayManager Generation (ProbeDisplays) - where the programmer generates a ProbeDisplay directly, by requesting it from the probeDisplayManager object (of class ProbeDisplay) provided by the kernel (in graphics mode). The probeDisplayManager will create the ProbeDisplay based on a ProbeMap given to it by the probed object. Note: since the probeDisplayManager is only created in graphics mode, it is documented in the simtools area of the documentation. Here we emphasise simply that in order to generate a ProbeDisplay onto an object the programmer need only write the following line of code:

```
[probeDisplayManager createProbeDisplayFor: anObject] ;
```

And that the contents of the returned ProbeDisplay will be based on a ProbeMap obtained from the probed object itself.

Probe support resident in the objectbase
(objectbase.html)
library is documented with that library. Extra description, above and beyond the bare interface description for precision probing and SwarmObject is below:

Precision setting for floating-point numbers with probes

Global setting of precision

There are two types of global precision setting via ProbeLibrary (objectbase/ProbeLibrary.html)
:

-setDisplayPrecision: (int) nSigFigsSaved
: Sets the number of significant figures for floating point (and double-floating) numbers diplayed on a GUI widget. Currently this is only implemented for VarProbes. The display uses the %*g sprintf-style formatting, which can vary slightly from implementation to implementation. If you set the number of significant figures to 3, then a float of value of 0.6344346 is displayed as 0.634 on the GUI widget. Note that this in no way affects the underlying stored value of the floating point number.

-setSavedPrecision: (int) nSigFigsSaved
: Sets the global default for the saving of floats through ObjectSaver. All objects with floats and doubles as instance variables are saved with the precision specified by this method. This is independent of the displayed precision of the same instance variable on a GUI widget.

To actually initialise these defaults: in the top level swarm, you should add the calls to the global ProbeLibrary instance (which is actually created by the initSwarm call in main) during the - createBegin method (this sets the precision in the global instance, before any probes are checked out of the instance. If neither method is called on probeLibrary, then the precision defaults to six significant figures in both cases.

Example: Here's how it's done in HeatbugObserverSwarm.m:

```
+ createBegin: aZone
{
  HeatbugObserverSwarm * obj;
  id <ProbeMap> probeMap;
```

```
  [...]

  probeMap = [EmptyProbeMap createBegin: aZone];
  [probeMap setProbedClass: [self class]];
  probeMap = [probeMap createEnd];
  // set the display defaults
  [probeLibrary setDisplayPrecision: 3];
  // typically saved precision would be higher than displayed precision
  // for statistical and data analysis purposes
  [probeLibrary setSavedPrecision: 10];

  // Add in a bunch of variables, one per simulation parameters
  [probeMap addProbe: [probeLibrary getProbeForVariable: "displayFrequency"
    inClass: [self class]]];
  [...]

  // Now install our custom probeMap into the probeLibrary.
  [probeLibrary setProbeMap: probeMap For: [self class]];
  return obj;
}
```

Setting precision for individual probes

The formatting for an individual probe can be set directly, using a sprintf-style formatting string. Typically, `customProbeMaps` are created in the `+ createBegin` factory method for a Swarm or a SwarmObject. To set the formatting for a floating point probe, the method from `VarProbe`

`(objectbase/VarProbe.html)`
is used:

```
-setFormatFloat:  (const char *)floatFormat
     is applied to the instance of the VarProbe "checked-out" of the global probe
     library instance.  The sprintf-formatting string can "over-ride" the "%g"
     format set by the global precision (as above) (Typically "%g" chooses between
     the "%f" and "%e", depending on the size of the exponent - which is implementation-
     - this method allows you to explicitly set the type of display).
```

In the following example, it is desired that the number of significant figures for the floating point variable `randomMoveProbability` is three (3). Currently (Swarm 1.0.5) this is only works for `VarProbes` and not `MessageProbes`, as yet.

Example Here's how it's done in `HeatbugModelSwarm.m`:

```
+ createBegin: aZone
{
  HeatbugModelSwarm * obj;
  id <ProbeMap> probeMap;

  id floatProbe;

  [...]
  // the -setFloatFormat is applied to the probe which is
  // "returned" from the call to probeLibrary
  floatProbe = [[probeLibrary getProbeForVariable: "randomMoveProbability"
    inClass: [self class]]
                                  setFloatFormat: ".3f"];

  // now we have the probe - put it back into the customMap
  [probeMap addProbe: floatProbe];

  [...]
  return obj;
}
```

Or, more compactly:

```
+ createBegin: aZone
{
  HeatbugModelSwarm *obj;
  id <ProbeMap> probeMap;

  [...]

  [probeMap addProbe: [[probeLibrary getProbeForVari-
able: "randomMoveProbability"
    inClass: [self class]]
                            setFloatFormat: "%.3f"]];
  [...]
  return obj;
}
```

SwarmObject Support for Probing

As mentioned earlier, all SwarmObjects can generate Probes/ProbeMaps. By default, they achieve this by "checking out" the appropriate references from the probeLibrary.

```
Interface definition
(objectbase/SwarmObject.html)
from objectbase
(objectbase.html)
library.

  - (const char*)getInstanceName

     If you want to put a specific title on the ProbeDisplay window associated
     with your object, simply override this method.  By default it returns the
     class name of the object.  This is why the default ProbeDisplay window title
     is always the class name of the probed object.


Note:  SwarmObjects do not actually have an instance variable called "instanceName"
- so if you want to store a name for your object, you will need to provide your own
storage at the subclass level and rewrite getInstanceName to refer to this variable.


  - getProbeForVariable:  (const char *)aVariable


  - getProbeForMessage:  (const char *)aMessage


  - getProbeMap


  - getCompleteProbeMap
```

Probe support resident in the *simtools* library:

ProbeDisplay

The `ProbeDisplay` class is meant to serve a dual role, represented by the two example windows on this page:

On the one hand if an object to be probed is specified *without* any particular ProbeMap being specified, then the ProbeDisplay generated will provide a window of class DefaultProb which displays all the variables resident in that class structure.

On the other hand, if a ProbeMap *is* specified then the ProbeDisplay follows exactly the specification as represented by the contents of the ProbeMap (above). When used in this manner, ProbeDisplays can generate tailored interfaces to objects (so for example, we have purposefully hidden certain instance variables in the HeatbugModelSwarm class, and have shown only one of the methods which the class understands).

Graphical Interface Usage

Common to both the standard ProbeDisplay and the CompleteProbeDisplay:

- The different fields in the ProbeDisplay can be updated by typing in new values and pressing Return. However, certain fields (containing pointers or ids, for example) cannot be modified and will generate a beep if such a modification is attempted.
- If an instance variable/argument slot is defined to hold an object, then that object can be drag&dropped into another variable/argument slot by clicking on it with the first mouse button (a small rectangle with the name of the object will appear - simply drag it to another object-typed variable/argument

```
slot and release the mouse button).
```

- Also, if an instance variable/argument slot is defined to hold an object, then that object can be inspected by clicking the entry for that variable/argument slot with the third mouse button(a ProbeDisplay for that object will be generated).

Available only on the standard ProbeDisplay:

- Note that the sunken label at the top of the ProbeDisplay is also active. By clicking on it with the first mouse button you get a drag&drop'able representat of *self*. By clicking on it with the third mouse button you get a CompleteProbeDi to *self*.

Available only on the ProbeDisplay:

- The green "superclass" button can be used to display the succesive superclasses of the object being probed.
- The red "hide" button can be used to hide classes which are irrelevant thus reducing clutter.
- The hide button on the lowest class in the hierarchy has a special meaning since clicking on it dismisses the entire ProbeDisplay.

Interface definition

(simtools/ProbeDisplay.html)
from simtools
(simtools.html)
library.

ProbeDisplayManager

The normal Swarm simulation will probably only ever contain one instance of this class, namely the probeDisplayManager. This object is used for automatic generation of ProbeDisplays: given an object it will attempt to get a ProbeMap from the object. If the object does not respond to the method - getProbeMap (i.e. it does not inherit from SwarmObject), the probeDisplayManager will query the probeLibrary for a ProbeMap. If there is not a *ProbeMap* installed in the `probeLibrary` for the class of that object, it will creat a *DefaultProbeMap*. It will then create a ProbeDisplay to the target object, and from then on, until the Probe Display is removed it will update the ProbeDisplay when receiving an update message.

Interface definition
(simtools/ProbeDisplayManager.html)
from simtools
(simtools.html)
library.

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08