

## Re: Haven't done anything real with OOP yet.

**Source:** <http://coding.derkeiler.com/Archive/General/comp.object/2004-10/1556.html>

---

**From:** Gregory L. Hansen ([glhansen\\_at\\_steel.ucs.indiana.edu](mailto:glhansen_at_steel.ucs.indiana.edu))

**Date:** 10/29/04

Date: Fri, 29 Oct 2004 19:22:55 +0000 (UTC)

In article <ZQugd.23\$a24.18@trndny07>,

H. S. Lahman <[hsl@pathfindermda.com](mailto:hsl@pathfindermda.com)> wrote:

>Responding to Hansen...

>

>>>Actually, the temperature controller is probably a good candidate for  
>>>OO. It is more akin to R-T/E development than something like operations  
>>>research algorithms. Let's look at a couple of the objects you  
>>>identified in your blitz...

>>

>>

>> Every peice would have their own set of parameters and state variables,  
>> which could in principle all be contained in a structure whose pointer is  
>> passed around. As objects I don't think they'd have a lot of member  
>> functions, just a transfer function that returns some output when given an  
>> input. Plus the usual maintenance like initializers, destructors, and  
>> things that set the parameters like the setpoint or heat capacity.

>

>True, as I eventually pointed out.

>

>>

>>

>>>Plant. I assume this is the hardware that actually provides heating and  
>>>cooling (e.g., an HVAC installation). Seems reasonable.

>>

>>

>> It's the thing whose behavior is to be controlled. In my case it's a  
>> target cell filled with liquid helium-3, and we want to control the  
>> temperature. The heating is done by an electrical resistor epoxied to the  
>> back. But that doesn't really matter. Heat goes in, heat comes out, a  
>> temperature is measured. The transfer function is an exponential.

>

>OK, then Plant is more like what I assumed for Building and the Resistor  
>is what I was thinking of as the HVAC hardware. I think it might be  
>clearer, though, when one gets to control actions to think in terms of  
>Cell and Resistor as separate entities. Plant is a bit ambiguous about  
>whether it includes the resistor or not.

comp.object: Re: Haven't done anything real with OOP yet.

Sorry, I was using some traditional terms from control theory without thinking that they should be defined if the culture isn't mutually understood and assumed.

>>> *Thermometer. Somebody has to track the actual temperature, so this seems reasonable also. [I suspect the problem would only be interesting if there were multiple Thermometers in, say, a building and one had to get an average or something. That computation would probably be a responsibility of the container, such as a Building entity.]*

>>>

>>> *Electronics. This sounds like a surrogate for hardware associated with the Thermometer. One communicates with it by reading/writing registers but otherwise one really doesn't care about it. Could the register read/writes be encapsulated in the Thermometer methods? Probably.*

>>

>>

>> *The thermometer is a germanium resistor, it has an  $R(T)$ . That's converted to a voltage by a lock-in amplifier reading a bridge circuit, the lock-in has a low-pass filter, so it's a little more than just a multiplying factor. And the output is passed along to the computer. I've thought of the whole set as a logical unit.*

>

> *OK. But that just says that Thermometer has to do something more complicated than returning a memory location's value when `getTemperature` is invoked. One of the nice things about abstraction is that it allows one to hide such details via encapsulation (in this case method implementation) from the Big Picture. So the electronics is important to the way the temperature is computed in Thermometer but not to how the temperature is controlled in Plant.*

>

>>

>>

>>> *Control Loop. For an OO reviewer the klaxons would be sounding here. This sounds very procedural or functional. That is, it sounds like a behavior responsibility rather than a entity. For example, it might be a logical behavior of something else, like Building.*

>>

>>

>> *The temperature controller. Implemented in a digital computer, in this case. It calculates a heater output based on a temperature input. Or really, a deviation of the temperature from the setpoint. An input of zero means everything is fine.*

>>

>> *It's really not treated any differently from the other components. An error signal goes in, and a voltage comes out. Analyzing it would start with a diagram like*

>>

>> \_\_\_\_\_

>> *setpoint ->( )-->| controller |--->| plant |---+-> temperature*

>> ^ || |(target)| |

>> | | \_\_\_\_\_ | | \_\_\_\_\_ | |

Re: Haven't done anything real with OOP yet.

comp.object: Re: Haven't done anything real with OOP yet.

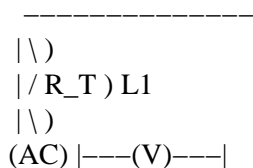
```
>> | _____ |
>> -----<----| transducer |----<-----
>> |(thermometer)|
>> | _____ |
>>
>> And each block could be given more detail. The controller has a
>> proportional term, integral term, derivative term, and a digital filter in
>> it.
>
>Is this controller hardware or software? If software, is it part of the
>proposed application or already realized (e.g., provided by the vendor)?
> This is important because if there is a physical entity in the problem
>space, it should be abstracted to be responsible for the algorithm if
>you have to write it. However, if you don't have to provide the
>algorithm, then all one has is two message transfers to the outside
>world: the measured temperature goes out and the computed voltage comes in.
```

I didn't expect to talk about this in such detail, although I sure don't mind. Maybe I'll take a step back and describe what we have.

It's a radiometer that measures neutron flux by the heat of reaction of neutrons being absorbed in a lithium-6 or helium-3 target, operating at about 1.8 kelvin. A weak thermal link (a strip of copper foil) connects the target to a heatsink, and the temperatures of the target and heatsink are separately controlled, so a constant amount of heat flows from one to the other. The beam is turned on and off, which changes the amount of electrical power that is needed to keep the target temperature constant. We measure the electrical power in each state to high precision, and (P\_off - P\_on) is the heat of reaction from the beam. We know the energy per neutron, so we get the flux.

The temperature measurement is done with germanium resistance thermometers in an AC bridge. The thermometer and a reference resistor are cold and form one leg, and a variable ratio transformer at room temperature forms the other leg. An AC power supply is shown on the left below, and the voltage is measured across the middle of the bridge by a lock-in amplifier. The zero point is set by the transformer, so a voltage of zero means the target is at the desired temperature, and I think a negative signal means it's hot while a positive signal means it's cold. That's digitized and read by a computer. We have an old Quadra still pulling its weight on this experiment. There's a thermometer, a ratio transformer, bridge circuit, power supply, and lock-in for each of the target and heatsink.

AC Bridge



Re: Haven't done anything real with OOP yet.

comp.object: Re: Haven't done anything real with OOP yet.

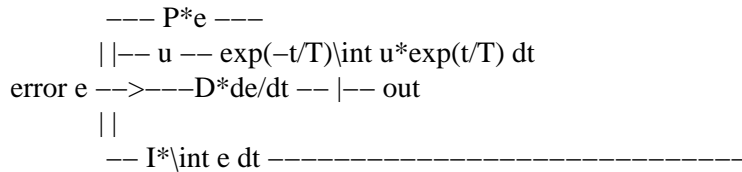
|/)  
|\ R\_r ) L2  
|/)

-----

The computer reads the error signals every 7/60 second, and calculates a power to be applied to each block. That's sent as a voltage to the lock-ins, which conveniently also have digital-to-analog outputs. The lock-ins then provide DC voltages to resistance heaters, one each for the target and heatsink.

The computer also records current and voltage of the control signal, counts accumulating in timers, and other stuff, but that has nothing to do with the temperature control.

The control system is a PID controller with a digital filter and an offset added, running in the Quadra along with everything else.



where that long part on the right is the analog expression of a low-pass filter, which I haven't put in discretized form since the terms on the left aren't discretized. Then take the square root of out since power goes as V^2, which linearizes the controller output in power. And an offset is added depending on the state of beam, on or off. That's given to the D/As on the lock-ins.

Often, in an industrial setting, they'd let the output fluctuate as it pleases, and let the thermal inertia of the block smooth it out. But the signal we measure is the output, not the temperature, so the filter is important in giving us a meaningful number.

So I hope that clears up where the physical entities are.

- >
- >Let's assume it is software you have to provide as part of the
- >application, even if it is embedded. It is still basically a single
- >algorithm that computes a voltage from a temperature deviation. If
- >there is physical controller hardware where the algorithm lives, then I
- >agree that needs to be abstracted as an entity and it will have a method
- >to compute the voltage.
- >
- >OTOH, if there is no such obvious entity in the problem space (i.e., it
- >is just conceptually a convenient place to put the computation in a
- >block diagram like yours above as you think about the design), then I
- >would probably put it in Cell (or Resistor). Without that behavior
- >there are no obvious responsibilities for Cell, yet it seems to be

Re: Haven't done anything real with OOP yet.

comp.object: Re: Haven't done anything real with OOP yet.

>pretty central to thinking about the problem.

I think it's more meaningful to treat the target cell separately from what we do to it. Sure, there may be a resistor glued to the back of it. But fundamentally we're interested in the temperature of the cell. The temperature is controlled by applying heat. Voltage means nothing except for its heating effect. Just as if it were a propane heated bath, the torch wouldn't be considered part of the bath.

The purpose of the target cell is to sit there, getting hotter or colder, acting as its own low-pass heat-to-temperature filter due to the heat capacity.

>

>Doing this sort of thing is known as anthropomorphizing. It is common  
>in all software development but has been raised to a semi-artform in OO  
>development. Since one is usually abstracting inanimate entities, it is  
>kind of a stretch to attribute them with doing things. At the same time  
>software is replacing operations that would be performed by people less  
>efficiently. So we map those control functions that people would  
>normally perform onto the inanimate entities as if the entities were  
>people. Therefore one tends to look for entities one has already  
>abstracted when seeking to assign behavior responsibilities rather than  
>creating an abstraction that really just represents a person's activities.

>

>>

>>

>>>Resistor. I'm not sure where this comes in. Part of the heating  
>>>mechanism or part of the temperature sensing mechanism? Either way, it  
>>>seems like it is a bit too detailed for the level of abstraction of a  
>>>temperature controller. To control temperature one extracts data from  
>>>sensors, processes it, and, if necessary, sends a message to the Plant  
>>>hardware. One should be able to abstract that at the level of Plant,  
>>>Thermometer, and <something else> that owns the control loop.

>>

>>

>> In this case, the target is heated by current through a resistor. The  
>> controller plus heater resistor should probably be considered a logical  
>> unit, with the output being heat, even if the resistor is physically  
>> closer to the target.

>

>OK, this is what I was envisioning with HVAC. It is a surrogate for the  
>hardware that actually provides heat.

>

>Given the nature of the simulation, I would probably put the computation  
>as a new temperature in here. (More precisely, the computation of the  
>hardware value the Thermometer electronics provides that needs to be  
>massaged into a temperature value by Thermometer per the above.) The  
>resistor is doing the heating so it is logical that power and heat  
>transfer calculates would be here.

>

Re: Haven't done anything real with OOP yet.

comp.object: Re: Haven't done anything real with OOP yet.

>>  
>>  
>>> *Timer. Because of delays ("elapsed time") in the HVAC you need to have  
>>> some sort of sampling policy for monitoring temperature. The  
>>> conventional way to deal with this is an external timer (e.g., in the  
>>> OS) that issues an event when the time has elapsed). This object would  
>>> be a surrogate for that external facility.*  
>>  
>>  
>> *Or in a simulation, each iteration represents some delta\_t. The actual  
>> control system has things happening on a lot of different time scales, and  
>> there's a loop that checks the time, then checks whether the temperature  
>> should be sampled, whether counters should be read and cleared, etc., and  
>> calculates when is the next time each action should be taken. It's a  
>> rather nice peice of procedural programming.*  
>  
> *OK, that's a bit more complicated. Note, though, that this just changes  
> the responsibilities of Timer. It now encapsulates a polling loop and  
> the rest of the algorithm. It generates an event to trigger temperature  
> sampling, etc. for an iteration. Eventually a Reset event comes back  
> that tells it the adjustments have been made and it starts polling  
> again. And...*  
>  
>>  
>> *For a simulation, I thought it would be nice to have two time scales; a  
>> "fast" time for the evolution of analog systems which are inherently  
>> continuous, and then the sampling time of the control system. That would  
>> allow exploring things like sampling rate versus stability. E.g. 0.01  
>> seconds and 0.1 seconds.*  
>  
> *The simulation adds some additional requirements. I suspect you will  
> need a separate Timer instance for each time scale. One simulates the  
> actual control sampling while the other simulates the hardware reaction  
> times. Fortunately both periods can be trivially parameterized for What  
> If scenarios.*

What I'd thought is the timer ticking away, and with each tick an input and a time interval would be given to each object. Or maybe I should think in terms of a time interval being given to each object, and let the objects themselves request an input from wherever it should come. Externally passing an output from one object to the input of another seems more procedural. Each object would need pointers to each other object it's connected to, then.

The analog objects would do their thing with each tick— they have no discrete interval in the real world, so give them whatever time steps are available. The controller would simply ignore the timer (its state, the voltage output, won't change) until `timer >= time_of_next_action`. Then it would request an error signal, recalculate the voltage, and calculate the next time of action. And then I suppose maybe a graph object would read the state of each of the other objects at the intervals of its

Re: Haven't done anything real with OOP yet.

comp.object: Re: Haven't done anything real with OOP yet.

choice, and collect data to be saved to disk. And then the beam power (I suppose there'd be a beam object that outputs to the target object) changes at a particular time so the response can be seen.

I'm not sure if that's what you mean by an event. But I like the thought of telling each object "Do something", and they'll know what to do. Member functions, besides maintenance (parameter setting, etc.) would include "report state" and "do something", and probably that's about it.

Either way,

```
if (timer >= time_of_next_action)
{
    do_something();
    time_of_next_action = time_of_next_action + time_step;
}
```

or something like that. And if the fundamental timer were incommensurate with time\_step, e.g. 0.1 seconds and 7/60 seconds, an action might be done at 0.2 seconds rather than 0.1167 seconds. Oh, well. The next action would still be scheduled for 0.2334 seconds, not 0.3167.

>

*>[Note that simulation probably introduces some other complexities. It  
>would probably be very helpful if the processing were event-based, which  
>means object state machines and at least one EventQueue object lurking  
>around somewhere. To deal with the two time scales, you will probably  
>need at least some concurrency because both Timers are probably going to  
>have to poll. None of these things is relevant for OOA solutions but  
>they would be introduced in OOD elaboration. (I am a translationist, do  
>I only do OOA models. B-)]*

So you're thinking like one type of timer attached to the analog elements, another type of timer attached to the discrete elements, each timer polled by an übertimer? And I was thinking of timers built in to each object, in the sense of determining whether it's time to "do something" yet.

I know events in the context of the Macintosh ToolBox; "MouseDown", "MouseMove", "MenuItem", etc. (Or whatever the official names are.) I'm not sure what role they're playing in your simulation.

I'm starting to feel like I'm monopolizing your time, so I'll cut it short. Except...

*>However, the complexity you are attributing to the controller algorithm  
>gets back to the point I made awhile back that OO isn't very helpful for  
>pure algorithmic stuff. Note that almost everything in the algorithm  
>will be contained in two methods: one in Resistor to compute the new  
>temperature as  $F(\text{last temperature, voltage, elapsed time})$  and one in  
>Cell to compute a new voltage as  $F(\text{temperature delta, elapsed time})$ .*

Re: Haven't done anything real with OOP yet.

comp.object: Re: Haven't done anything real with OOP yet.

No matter how OOP you are, eventually you'll have to go algorithmic. In the controller I outlined above, you could, in principle, define at least four objects; a proportional term object, an integral term object, a derivative term object, and a filter object. Throw in a square root object and an offset object, which weren't included in the diagram. Each one would receive a number, perform a simple mathematical operation, and return a result (in the integral and filter cases, a result that depends on the previous history). Or perhaps the output would request something from the square root and add the offset. The square root would request numbers from the integral and filter, and find the square root of the sum, and so on. But that would be kind of silly. And anyway, eventually you'll have to get to something like

```
define object 1
define object 2
tell object 2 to get input from object 1
```

That's algorithmic.

That makes me think of something. If object 1 requests something from object 2, object 2 requests something from object 3, and object 3 requests something from object 1 (feedback loop), you could get yourself into trouble. But I don't think the timer method I gave above would suffer from that, since "do something" is controlled externally by a timer, and each object, when requested, would return its current state without querying further back in the line.

```
--
"for every problem there is a solution which is simple, clean and wrong."
-- Henry Louis Mencken
```