

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

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2004-10/1625.html>

From: H. S. Lahman (h.lahman_at_verizon.net)

Date: 10/31/04

Date: Sun, 31 Oct 2004 05:19:52 GMT

Responding to Hansen...

>>>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.
>
>
> 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.

The road to requirements specification is paved with terminology mismatches.

>>>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)) | |
>>> | | _____ | | _____ | |
>>> | _____ |
>>> -----<----| 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

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

>>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.

Fascinating. From a software simulation viewpoint, <so far> I still think this behavior can be encapsulated in Resistor (or Target, maybe), rather than a separate Controller. You still have a basic equation to compute power in terms of voltage and elapsed time.

Similarly, Heatsink becomes an entity that encapsulates a different equation for power. It also sounds like somebody is doing something to "control" the heatsink temperature separately, which may mean another object is missing and/or some more relationships and maybe another Timer instance.

Just out of curiosity, how /is/ the heatsink's temperature controlled? Is there another beam stimulating it? This also implies there is another Thermometer for the heatsink.

>
> 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
>
> -----
> / \)
> / / R_T) LI

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

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

> / \)
> (AC) /----(V)----/
> / /)
> / \ R_r) L2
> / /)
> -----

Things are getting more complicated. How typical of software development in general; it never ends up as simple as it first seemed. The Thermometer is complicated enough so it may be necessary to simulate its innards. That will depend on how the analog side of things is simulated.

I note that Thermometer is not a conventional thermometer in that, rather than producing a temperature value, it is actually transforming the temperature the hardware is measuring into a continuous stream of deviations (the error input to the filter below, if I understand this correctly).

When the target is being heated the temperature will change for the hardware to measure. But there isn't any hardware for the software simulation. Therefore somebody needs to compute a temperature in response to the target stimulation. To a first approximation, that can be calculated directly from the stimulus to the target (adjusted for heatsink dissipation). Then you don't need this much detail and Thermometer just becomes a dumb holder of a current temperature attribute calculated by Resistor.

OTOH, if you are really interested in things like delays and response times, then you might need to model the bridge with individual components (classes) that interact. That is, you may need to model the conversion of temperature to deviations explicitly. (I'm a bit skeptical of that but this scale is a different world, so I thought I'd throw it out as food for thought.)

>
> *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.*

In the simulation you have to make the same calculation that the computer makes in the real system, right? If so, I will back off my inclination to put this calculation in Cell. Since this is a software simulation of the real hardware, then modeling the specific hardware entities literally is quite reasonable. However,...

>
> *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.*

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

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

>
> *The control system is a PID controller with a digital filter and an offset
> added, running in the Quadra along with everything else.*
>
> $P * e$
> $\int u \exp(-t/T) dt$
> $error\ e \rightarrow D * de/dt$
> \int
> $I * \int e dt$
>
> *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.*

...this is providing the input to the Computer. Here is where the actual current temperature values are extracted from the Thermometer. So Thermometer, Controller, and Computer are just decomposing a single T -> V calculation into three parts that are daisy chained. IOW, the need for A/D conversion and whatnot are hardware issues rather than simulation computation issues.

Unless there is something interesting about the filters (delays, etc.) to simulate, I would be inclined to combine Computer and Controller into a single Controller entity. Since there is no hardware for Thermometer in the software simulation, the feedback needs to be calculated somehow. That calculation is presumably deterministic. Depending on the goals of the simulation there may be many ways to abstract the feedback calculation across Thermometer, Controller, and Computer.

For example, the feedback computation that replaces the thermometer hardware could calculate a simple temperature value from the heat flows between Resistor and Cell. Then Computer could go directly to a calculation of voltage based on the temperature value rather than sampling the filter's "out". So long as the calculations were equivalent in accuracy, one wouldn't need a literal simulation of the filter in Controller at all. (Clearly that doesn't work for anything but rather simple simulation goals; it would be useless for doing something like evaluating filters.)

However, if I understand this correctly, the Thermometer output (deviation from 0, e) is a continuous analog signal that is smoothed by the filter. But the voltage computed by Computer seems to be constant for a given ON burst. Does the Computer provide additional smoothing or is the 7/60th sample sufficient?

>
> *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*

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

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

- > *signal we measure is the output, not the temperature, so the filter is*
- > *important in giving us a meaningful number.*

This is kind of what I had in mind when I said that in the simulation Resistor would have to compute whatever the Thermometer used to determine temperature. However, the actual situation is significantly different since temperature is only one input.

How one abstracts Thermometer, Controller, and/or Computer will be depend on how you decide to simulate the analog portion. Ultimately something needs to be computed that reflects the power input and heat transfer equations and then that something needs to be fed back into the voltage and burst time calculation in Computer. Ultimately that will determine how many and how detailed the hardware abstractions need to be form an OO viewpoint.

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

Probably more than I ever wanted to know. B--))

- >
- >
- >>*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*
- >>*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.*

That's quite true, but the software simulation doesn't have any hardware. For example, the Thermometer gets hardware inputs from the

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

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

target cell that it converts into the error stream. In the simulation, you will need to provide those inputs somehow for the feedback. That means that some object needs to do a heat flow calculation that isn't done in the actual system. I am arguing that object is either Resistor or Cell. One can justify Resistor because it naturally knows the power being dissipated and its purpose in life is to raise the temperature of Cell. One can justify Cell because that is where the real heat transfer takes place that results in a temperature rise (which would be more persuasive if there were things like convection and mixing effects).

Where one puts the calculation depends on what feels more natural in the overall simulation design. But it has to go somewhere and that will likely affect the granularity of how one views Plant in the simulation because of separation of concerns and cohesion issues.

BTW, this brings up another issue that goes with some of the points above. The basic problem the software is providing (as I understand it so far) is a hardware simulation. Obviously one needs to abstract the hardware entities. However, some responsibilities and even entities (e.g., timers) may exist strictly because of issues related to the simulation design itself. That, in turn, depends upon what needs to be accomplished in the simulation. IOW, simulation techniques themselves represent a problem space that may also need to be abstracted.

>>>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

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

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

- > *read the state of each of the other objects at the intervals of its*
- > *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.*

It seems to me you still need two timers. One for the beam calculation every 7/60th of a second and one for the computing heat transfers for the feedback loop. In digitizing the feedback loop, it would probably be better (harmonics in the filter) and more convenient (fixed elapsed time) to have a fixed frequency. You can handle the time slicing with separate threads (for hard real time control) or using object state machines with a single event queue manager (for loose time control). That choice, though, gets into pretty low level simulation design.

- >
- > *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.*

I meant event in the sense of state machines. Each relevant object's behaviors would be captured in an object state machine. The timers would then generate events that would trigger the transitions and corresponding actions. As it happens, finite state machines are quite good at enforcing good OO practices. For example, the rules of FSMs prohibit a state from knowing its previous state or what its next state will be; the associated action can only operate on the input alphabet. More important, the collaboration is inherently asynchronous so the sender of an event can't depend on what the receiver does because there may be an arbitrary delay before the event is actually processed. Most important of all, state machines separate message (event) from method (state action), which is fundamental to good OOA/D.

Thinking in terms of imperatives (Do This) during collaborations is a very procedural view of the construction. The OOA/D paradigm prefers messages to be announcements of something the sending method did (I'm Done). This ensures that the sending method's implementation does not depend on what the specific response to the message is. Such hierarchical functional dependencies were commonplace in procedural functional decomposition (i.e., spaghetti code). This is probably the single most important difference between the procedural and OO construction paradigms and it very strongly affects the way applications are built in the two paradigms.

When you employ a Do This view the message sender must know three things: (a) that a specific receiver exists, (b) that the receiver has a specific behavior, and (c) that behavior is the next thing to be done in the overall algorithm. All three break encapsulation but (c) is the most insidious because it invites one to hard-wire sequences of operations in the overall solution into method implementations.

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

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

That's reflected in the way one defines objects and their responsibilities before defining the collaborations in OOA/D. And collaborations are defined (i.e., determining who sends messages, who the messages go to, and when they are sent) at a higher level of abstraction (e.g., a UML Interaction Diagram) than object methods.

[Though experienced developers only do it for tricky situations, one could fully define all the classes and methods without any collaborations. One could then define the collaboration messages and back-fill by inserting the calls at the end of the existing methods in OOP code. One can also use DbC explicitly to do this by matching the preconditions for executing some behavior to the post conditions of another behavior to determine who sends the message. (Note that this is routinely done R-T/E when using interacting state machines.)]

>>*[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.*

Yes to the first part; discrete and analogue would have separate timers.

No to the second. I assumed each timer would poll the OS system clock. Better yet, because the OS would handle the polling time slicing, the OS may provide timer facilities via registering a callback method in the application Timer object. (I have no idea whether the Mac OS provides timers.) But now we are getting into pretty low level implementation details for the Timer class.

>>*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 $F(\text{temperature delta, elapsed time})$.*

>

>

> *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*

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

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

- > *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*

Things like square roots are far too detailed for OO objects. The calculations (e.g., the filter) you describe above are much more complex and will be encapsulated in methods. Consider Computer and Resistor. Assuming it doesn't provide additional smoothing, Computer would sample "out" from the filter and compute values of voltage and burst time as an output sent to Resistor. Resistor takes those inputs and, for the simulation, computes the power. That calculation involves a square root but it will be buried in the Resistor method implementation.

- >
- > *define object 1*
- > *define object 2*
- > *tell object 2 to get input from object 1*
- >
- > *That's algorithmic.*

Yes, but at a much higher level of abstraction. The OO application is about gluing together the simulation elements, not the details of the individual computations.

- >
- > *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.*

Yes and no. B-) First, in OOA/D knowledge access is treated differently than behavior access. It is assumed knowledge can be accessed synchronously on an as-needed basis. (If the knowledge access has an inherent delay, such as being distributed, then OOP must make sure it seems synchronous by making sure the getter doesn't return until the data has been retrieved.) This greatly facilitates data integrity because when a method needs knowledge from other objects it navigates directly to them to get it *_within the scope of that method_*. This makes dealing with things like concurrency during OOP a */lot/* easier. Corollary: in a well-formed OO application there is usually very little data passed in messages because the methods get what they need directly.

OTOH, in OOA/D behavior access is viewed as asynchronous (i.e., there may be an arbitrary delay between when a message is sent and when

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

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

someone actually responds). So one tends to see a lot more "handshaking" messages in OO application than one sees in procedural applications. Also, encapsulation and logical indivisibility force class methods to be cohesive (usually small) and self-contained. Also the I'm Done paradigm I mentioned above makes sure the method can be specified in complete isolation from other methods. That allows one to connect the algorithm dots at a higher level of abstraction than individual objects, also as I mentioned above. (Ain't it great when a Plan comes together?)

I also mentioned DbC for defining collaborations. The preconditions for executing a method are usually two: (a) some other operation in another object has occurred (algorithmic sequence) and (b) the data the method needs to eat is timely (the application state is correct). However, since attributes are always set in object methods, one can enforce (b) by ensuring that the proper methods have already been called by the time (a) has been called. One does that by connecting the dots of those self-contained methods.

A pseudocode example might help:

```
classA::doIt (x)
  tmp = x + 5
  y = myB.doIt (tmp) // message to B
  z = myC.doIt (y) // message to C
  classA.attr1 = z * 3
```

```
classB::doIt (x)
  ... // do something with x
  return x * 2
```

```
classC::doIt (x)
  ... // do something with x
  return x + 1
```

ClassA::doIt is typical of procedural applications but it is awful OO because it is full of dependencies on what other objects do and it also hard-wires a solution sequence between B and C. Note that classA::doIt cannot be unit tested unless correct implementations of the other two methods exist. (Using a test harness to stub them by returning values consist with the input is just an exercise in self-delusion; one is just unit testing the test harness.) So one would do some rewriting without changing the functionality allocated to each object:

```
classA::doIt1 (x)
  tmp = x + 5
  myB.doIt (tmp) // message to B
```

```
classA::doIt2 (x)
  tmp = x * 3
  classA.attr1 = tmp
```

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

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

```
classB::doIt (x)
  ... // do something with x
  myC.doIt (x * 2) // message to C
```

```
classC::doIt (x)
  ... // do something with x
  myA.doIt (x + 1) // message to A
```

Each object has exactly the same responsibilities as before. All that has changed is the way the responsibilities of classA are organized and how the collaborations are done. However, now each method is now fully unit testable without implementing any of the other methods. (At most, all one has to demonstrate is that the message was sent with the right arguments.)

The original solution sequence is also preserved by the sequence in which the methods are invoked, which happened by connecting the dots properly among small, self-contained, logically indivisible methods.

Bottom line: to eliminate dependencies OO applications will have small methods and lots of messages between them.

BTW an analogy I sometimes use is Tinker Toys where one creates shapes by connecting slotted wheels with spokes. One can make the <tenuous> analogy mapping:

Shape = problem solution
wheel = class
wheel slot = method
spoke = message

To get a different problem solution one simply reconnects the existing wheels and there slots with new spokes. Obviously one often has to modify the methods as well to accommodate change. However, it is surprising how much of the change can be handled by simply reorganizing the messages.

```
--
*****
There is nothing wrong with me that could
not be cured by a capful of Drano.
H. S. Lahman
hsl@pathfindermda.com
Pathfinder Solutions -- Put MDA to Work
http://www.pathfindermda.com
blog (under constr): http://pathfinderpeople.blogspot.com/hslahman
(888)-OOA-PATH
```

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