

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2005-05/msg00254.html>

- *From:* "H. S. Lahman" <h.lahman@xxxxxxxxxxx>
 - *Date:* Thu, 19 May 2005 20:00:01 GMT
-

Responding to Hansen...

Let's see if I can hang with this game. What I'd assumed before was that parts should be combined to reduce unnecessary units. So, for instance, I'd combined the germanium resistor, bridge circuit, AC supply, lock-in, analog to digital converter, and GPIB into a single Thermometer. Input is temperature, output is a voltage difference (V=0 at the setpoint).

Fascinating. I assumed it was the other way around; all that hardware you describe produced a voltage in a register that the Thermometer driver software converted to a temperature for the feedback (controller) processing.

Is there any software at all involved in producing the delta V? If not, then this doesn't seem interesting to the simulation because it will all be replaced by your heat flow calculation that produces the delta V. (I had that heat flow calculation living in Target and responding to the E1 event.)

No software at all in producing delta V, it's just something read out over the GPIB.

So when you say, "Input is temperature, output is a voltage difference" you mean the hardware is observing a temperature and presenting its value as a delta-V, right? Does the feedback control processing deal directly

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

in delta-V values or is a software driver converting the delta-V to a temperature as it is sampled and before it is processed?

First, a definition, just to be sure we are on the same page. A device driver is software that is dedicated to talking to a particular hardware unit. The main purpose of a device driver is to shield other software from the details of the hardware unit. The most common way that is manifested is through mapping HW (delta-V) and SW (temperature) semantics.

What I am pushing on here is that what the hardware is doing is producing a value in a register. Nothing about how that value gets there is relevant to how the controller software deals with it once it is there. I would also bet that the actual control processing in software doesn't begin until one has a temperature sample. So one has:

```

                delta-V                temperature
HW Register -----> SW driver -----> SW controller
```

For the simulation you will replace the hardware producing the delta-V with heat flow computation software that will probably be encapsulated in an object that abstracts whatever thing has a temperature. As it happens, that simulation computation is naturally going to produce a temperature value. It would be silly to convert it to a delta-V just to have the SW device driver convert it back to a temperature, especially since the SW driver needs to be modified to "read" the value from somewhere other than a hardware register anyway. So I would change the SW driver to simply get the temperature from the thing being heated and pass it back out.

The point here is that the SW device driver provides a low-level buffer to isolate the software from the hardware. (GPIB is actually another, even lower level that can isolate the single delta-V value from things like split register values.) That allows one to draw a very clear line in the sand between hardware and software. So when you replace the hardware with simulation software you will have a very robust buffer to encapsulate any "glue" needed to map the different views.

The corollary, which is relevant to the discussion below, is that all the processing on the software side should be the same as in the real controller except for the implementation of SW driver, which gets data from a different place. However, things are not the same on the hardware side, especially the synchronization -- both between hardware elements and, consequently, at the "touch" points between HW and SW. That's why it may not be possible to synchronize the overall simulation with exactly the same triggering scheme as provided with the present HW.

And I had the control logic, digital to analog converter, an attenuator, and

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

heater resistor as Controller, with input being a voltage difference and output a heat. Technically everything after the DAC is analog, but it faithfully changes only when the controller tells it to change, there was no independent time evolution, and combining them let me get rid of a square root calculation, so it was as good as part of the controller. And the target itself was just a chunk of metal.

This sounds like a mix of software (control logic) and hardware (everything else). I was putting the control logic in Controller. I assumed the rest of the hardware is what the control logic controls. The hardware elements, like DACs and ADCs, would not appear in the simulation at all unless you had to explicitly substitute simulation computations (e.g., computing ΔV) to emulate them.

Besides my algorithmic background, I was coming from a control theory background, which wants to abstract away such things as the implementation of the controller. And from a control theory point of view, what the controller does is adjust the heat delivered to the target.

One difference is that the control theory view is looking at the whole system and doesn't distinguish between HW and SW. However, from the software design view of the real controller, the hardware is an external black-box agency. (In use case terms, an Actor.) This is especially important when you simulate the whole system in software because the hardware side isn't the same. Viewing HW and SW as separate, interacting modules allows one to provide the HW simulation seamlessly.

However, apropos of the point above, as separate modules the synchronization of the interactions across the HW/SW boundary may be affected by "doing" the HW a different way. If one already sees them as independent modules in the collaboration, then it will be easier to analyze synchronizing the touch points between them.

Another difference is that one manages complexity in SW through modularization. While conceptually the view

read

write

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

[Registers] -----> [Controller] -----> [registers]

may be accurate at a megathinker level, it is not very useful for writing the software for the controller. That's especially true when we have already identified several different tasks that need to be coordinated in time and with the HW. From a software view the granularity of the modularization must be at least fine enough to support collaborating synchronization messages.

It gets worse for synchronization across touch points between HW and SW when one completely replaces the HW. You can use the same synchronization scheme for sequencing /within/ the Controller, but you may have to rethink synchronization between HW elements and the Controller elements.

However, I get the sense below that Controller is limited to just talking to the stimulus hardware. That is, it is really Chart that is determining what the stimulus should do and Controller just knows how to relay that to hardware registers.

Eh, Chart is just collecting data to save to disk. There's not much point in operating the system without it, but from the point of view of control or simulation it doesn't play an active role, it's just something that's there.

Then it doesn't really give a hoot about the synchronization mechanism. Any mechanism that ensures that data is consistent and written at the right time will do. That is, writing data doesn't depend on it being the 600th tick; it depends on there being a proper collection of data available and the rest of the system happens to ensure that it is there and consistent on the 600th tick. So any announcement by any simulation artifact would do so long as that artifact reasonably knows the data collection is complete and consistent.

However, that orthogonal responsibility of Chart just leaves open my other issue about who is doing the processing between observing temperature and ordering more heat. The processing seems to be complex enough to warrant breaking down Controller further, regardless of what control theory might say.

[Though I think that is really an issue of scale, which one can cast in

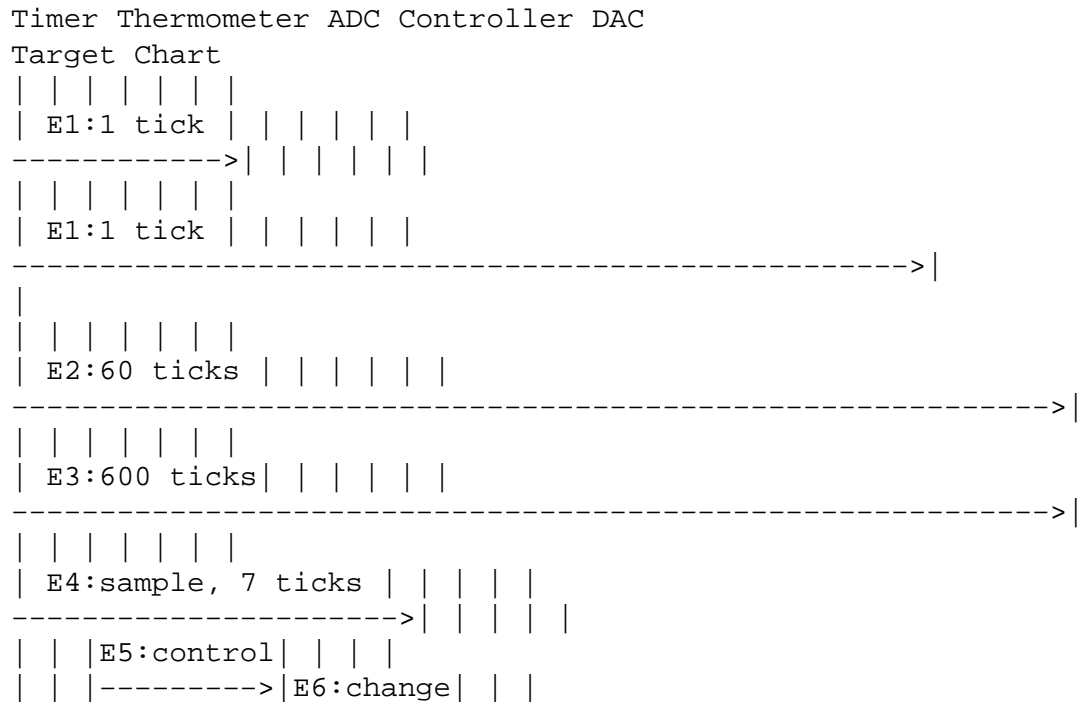
Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

OO terms as level of abstraction. If computational elements /within/ the controller need to be synchronized with elements of the hardware, then I have to think the control theory view is going to subdivide Controller at least until that synchronization can be unambiguously expressed.]

We could factor the ADC out of Thermometer (I believe that would be your Sampler), and factor the DAC out of Controller. Also, the controller gets a temperature every 7 ticks, a chart gets a temperature every 60 ticks. And the controller calculates a control action on a schedule, not on the command of the thermometer or ADC. In the non-OOP real program there's a big loop that includes a section that checks to see if it's time to do a control action. If so, it read a temperature and then calls the controller function. That's a lot like calling the Sampler on a schedule, and then the Sampler kicking Controller.

So let's see...



Re: Lahman, how ya doing?

```
| | | |----->| | |
|E7:1 tick | | | | |
|----- | | | | |
| | | | | | | | |<----- | | | | |
```

No events are going to Target or Thermometer other than E1, "analog time evolution", or being generated by them, because they have the role of passive elements that always do what they do, and always have their outputs ready for whoever wants to read them.

Confusion reigns. B-)

Why would a DAC or ADC be getting events? They aren't software in your simulation; they are pure hardware elements. At best some software element writes a value to a DAC register and you can't even do that for an ADC because the software can only talk to digital registers.

I guess confusion does reign. I gave them events by way of telling the DAC to change its output and the ADC to give a number, but I guess I don't really know how to represent them. For what it's worth, when it's time to control the target there's a section of code like

Generally one treats talking to HW as reading/writing knowledge. The attribute just maps to a hardware register. Typically one provides surrogate objects for hardware that appear to be dumb data holders; when one needs to talk to the HW one just accesses the attributes. The surrogate object's getters and setters encapsulate the actual register access. Since you seem to have low level GPIB third-party drivers for the specific instruments, the getter/setter implementation would invoke the relevant GPIB driver API read or write. (Example below.)

Since it is perceived as knowledge access access, it is inherently synchronous and one would not use events for that. One only uses events for asynchronous behavior messages when object state machines are interacting.

In your diagram, the E4 event should be triggering some object state machine action that needs a specific value from the hardware (which happens to be in an ADC output register). That entity would invoke a

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

getter to the object acting as the relevant HW surrogate. IOW, E5 should be a getter. So the action that E4 triggers is really going to be in Controller and the ADC surrogate would not even appear in the diagram. (While one can include knowledge accesses on a UML Collaboration Diagram, nobody ever does so because the clutter tends to be enormous and it obscures the main use of the diagram for tracking behavioral flow of control.)

```
/*
  read target x
*/
r->runData[targInChan] = SR530ReadX(r->l[targLoop].ibnum);
if (fabs(r->runData[targInChan]) > r->maxX[targLoop])
    r->maxX[targLoop] = fabs(r->runData[targInChan]);
/*
  read target y
*/
r->runData[targSinChan] = SR530ReadY(r->l[targLoop].ibnum);
if (fabs(r->runData[targSinChan]) > r->maxY[targLoop])
    r->maxY[targLoop] = fabs(r->runData[targSinChan]);
/*
  control target
*/
if (r->loopMode[targLoop] == controlOn)
{
    /* calculate new output */
    TempControl(r,targLoop);
    SR530Dac5Out(r->l[targLoop].ibnum, r->runData[targOutChan]);
}
```

The SR530 functions are vendor supplied, r is a big data structure that holds all the parameters. TempControl() calculates a control action and places it in r. target x is the usual output from a lock-in, target y is the output with the reference signal shifted 90 degrees, $x^2+y^2=V^2$, and we trim the bridge to keep y near zero.

Right; this is actually doing pretty close to what I suggest above. The SR530 is the low (GPIB) level driver API I mentioned. However, in an OO design one would have something like:

```
class Target
{
public:
    float getX(int i) {return SR530ReadX(i);};
    float getY(int i) {return SR530ReadY(i);};
    void setZ (int i, float f) {SR530Dac5Out(i, f);}
    ...
}
```

Re: Lahman, how ya doing?

where Target is a surrogate for the HW, probably at a higher level of abstraction than an individual ADC. When creating HW surrogates for this sort of thing one often abstracts to a pretty high black-box level so the getters/setters for a single surrogate might actually access multiple instrument APIs at the GPIB level. [I think I mentioned we provided a C API with 300+ functions for our DTU driver. The real hardware could have dozens of different channel cards, each with several logical units and ~10**3 registers from a HW perspective. But from the test program developer's viewpoint it was just one big black box with 300 getters and setters.]

The computations in your examples are part of the business rules and policies of the problem semantics so they would be in the caller of getX. That uses the Target surrogate to isolate the problem solution semantics from the data access mechanisms. The price is indirection. The value lies in being able to swap instruments while only touching Target so one is 100% confident that problem solution logic still Just Works.

[Modularization and layered models were pioneered in R-T/E because the HW Guys keep changing the bit sheets during card revisions and the driver developers wanted to minimize the impact of those changes when the basic functionality did not change. It only took a decade or so to figure out that the same divide-and-conquer notions apply to any sort of maintenance. B-)]

Also, I thought the 7/60th ticks were used to filter out 60 Hz pick-up. I assumed that was done in software but this suggests the hardware is making that adjustment. If so, that could be completely ignored because your simulation won't have any 60-cycle pickup in the delta-Vs because they are computed in the simulation.

Is that what it suggests. Well, it's software. It could be any number, we've just traditionally ran with 7/60.

Sending the 7/60th tick event to a hardware element rather than a software element is what suggested that to me.

However, if that processing is about eliminating 60-cycle harmonics, it might not be necessary to your simulation. OTOH, if it is deemed necessary, then you have to insert the noise in your simulation of the

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

temperature samples. B-) Also, I suspect a prime number that is not a divisor of 60 would be better than others.

By the way, no FFTs, but there is a low-pass filter in the controller.

Hard to keep the threads separate. I was recently involved in one about ATC software on another forum and that might have been where I got it (i.e., processing radar data streams).

Chart appeared a couple of messages ago, but I have no idea what it does. (You may have explained it the first time around but I've long forgotten most of that.) I assume it is whatever is doing FFTs and whatnot on the temperature samples. (More precisely, one a set of multiple statistical samples from each group of 60 temperature samples.) I also assume that once it accumulates 10 such samples Chart does something with them.

But I would expect that 'something' that Chart does would be to determine what the hardware does. So why isn't Chart talking to Controller? Similarly, why is Controller talking to a DAC rather than the Target? If that's just because the software digital view needs to be converted to analog for the hardware, that stuff isn't irrelevant to your simulation. You won't be writing to registers all; you'll be writing attributes that your hardware emulation software reads.

Finally, if the Thermometer always produces its temperature value on schedule, where are the skips? This segues to another issue...

The skip is the ADC reading the thermometer, which I didn't think would be represented as an event going from ADC to Thermometer. I haven't looked at the ticks very carefully, but I assume it's just some witching tick thing where the target and heatbath both control, voltages are read and averaged, and counters are read all on the same tick. I've been assuming

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

there's just some times when there's too much processing for a 12 year old computer to complete in 1/60 second, or maybe even 2/60 second.

The ADC reading the thermometer is all hardware, right? So the hardware should be synchronizing the triggering of the thermometer and the conversion. Then the ADC should have the right value for the current thermometer reading -- always. (There could be a race in the HW, but that strikes me as poor HW design.)

As you opine, that means it is whoever in the software is reading the ADC output that isn't getting there in time because there was too much SW software processing on the witching tick. That is going to be tricky to emulate because of the SW processing you are adding to do the simulation. Which brings me back to the point I keep hammering about synchronization being different for some aspects of the simulation.

You may be able to do it with parallel threads at different priorities but it might take a lot of tinkering to get it to work (i.e., determine the right priorities). If that is the only source of skips, then it might be easier and more controllable to explicit emulate the delay for that particular activity.

(1) Insert a "delay" between whoever generates the normally scheduled event and whoever responds with the processing that must be skipped for the current tick. In effect the delay is such that when the incoming event is re-dispatched it is in the next tick. (This was your first feedback loop diagram.) I think finding a reasonable mechanism for the delay would be tricky, given the disparities in processing time. It also gets a bit hoaky because the "delay" may get another tickle in the next tick and it should not double dispatch in that tick. In that situation one is essentially doing (4) below.

Re: Lahman, how ya doing?

What it should do would depend on the details. But if there was a Delay catching sample events, it only needs one vacancy and then can discard any following sample events until the first is dispatched. Other events, like reset, are sort of meta-simulation and should be passed immediately.

By delay mechanism, I mean postponing processing of the event until the next tick. To do that you have to know when the next tick is in order to re-dispatch it at the right time (i.e., put the event addressed to the right target on the queue at the right time).

That's the job I had for the Delay object. If you want to delay a control event for one tick, send a delay event to Delay, and if it catches a control event on that tick it will hold it until the next tick. If you want to delay an event for two ticks, send two delays to Delay and it will count down for two ticks before releasing the control event.

My concern is how Delay knows that a tick has passed and one is "on" the new tick to release the event. It seems to me that Delay needs to get two events: the base tick event and the event that might need to be delayed.

That's fairly easy to do but there is another problem. When a deferred event is pushed, it may need to be in the correct order relative to other events that Timer burst out for the simulation tick. That is, if the deferred event was registered with Timer to be generated in a certain order compared to other events, then how does Delay preserve that order on the next tick?

Given the reason for the delay in the first place, this probably isn't important, but you might want to think about it a bit.

There is nothing wrong with me that could not be cured by a capful of Drano.

H. S. Lahman

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

hsl@xxxxxxxxxxxxxxxxxxxx

Pathfinder Solutions -- Put MDA to Work

<http://www.pathfindermda.com>

blog: <http://pathfinderpeople.blogspot.com/hslahman>

(888)OOA-PATH