Sidebar: Useful PN Actors

- **OrderedMerge** (`MoreLibraries->Esoteric`): Merge two monotonically increasing sequences of tokens into one monotonically increasing sequence of tokens.
- **Sequencer** (`Actors->FlowControl->SequenceControl`): Take a stream of input tokens and stream of sequence numbers and re-order the input token according to the sequence numbers. On each iteration, this actor reads an input token and a sequence number. The sequence numbers are integers starting with zero. If the sequence number is the next one in the sequence, then the token read from the *input* port is produced on the *output* port. Otherwise, it is saved until its sequence number is the next one in the sequence.
- **Stop** (`Actors->FlowControl->ExecutionControl`): Stop execution of a model when a true token is received on any input channel.
- **Synchronizer** (`Actors->FlowControl->Aggregators`): Synchronize multiple streams so that they produce tokens at the same rate. That is, when at least one new token exists on every input channel, exactly one token is consumed from each input channel, and the tokens are output on the corresponding output channels.

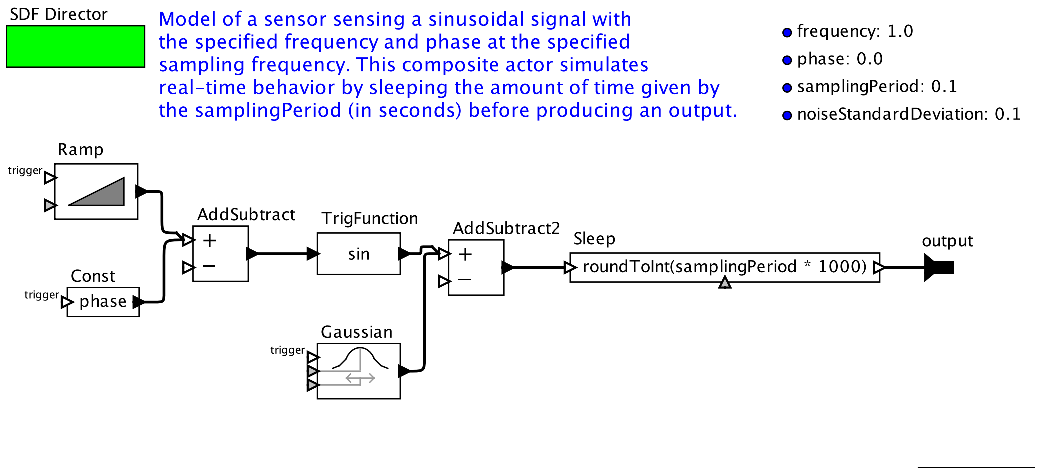
SDF Director
Model of a sensor sensing a sinusoidal signal with the specified frequency and phase at the specified sampling frequency. This composite actor simulates real-time behavior by sleeping the amount of time given by the samplingPeriod (in seconds) before producing an output.
frequency: 1.0
phase: 0.0
samplingPeriod: 0.1
noiseStandardDeviation: 0.1
Ramp
trigger
Const
trigger
phase
AddSubtract
TrigFunction
sin
AddSubtract2
Sleep
roundToInt(samplingPeriod * 1000)
output
Gaussian
trigger

Figure 4.2: Model of a real-time sensor.

5. A common scenario in embedded systems is that multiple sensors provide data at different rates, and the data must be combined to form a coherent view of the physical world. In general, this problem is called **sensor fusion**. The signal processing involved in forming a coherent view from noisy sensor data can be quite sophisticated, but in this exercise we will focus not on the signal processing, but rather on the concurrency and logical control flow. At a low level, sensors are connected to embedded processors by hardware that will typically trigger processor interrupts, and interrupt service routines will read the sensor data and store it in buffers in memory. The difficulties arise when the rates at which the data are provided are different (they may not even be related by a rational multiple, or may vary over time, or may even be highly irregular).

   Assume we have two sensors, SensorA and SensorB, both making measurements of the same physical phenomenon that happens to be a sinusiodal function of time, as follows:

$$\forall\, t \in \mathbb{R}, \quad x(t) = sin(2\pi t/10)$$

   Assuming time t is in seconds, this has a frequency of 0.1 Hertz. Assume further that the two sensors sample the signal with distinct sampling intervals to yield the following measurements:

$$\forall\, n \in \mathbb{Z}, \quad x_A(n) = x(nT_A) = sin(2\pi n T_A/10),$$

where T_A is sampling interval of SensorA. A similar set of measurements is taken by SensorB, which samples with period T_B.

A model of such a sensor for use with the PN director of Ptolemy II is shown in figure 4.2. You can create an instance of that sensor in Vergil by invoking the `[Graph->Instantiate Entity]` menu command, and filling in the boxes as follows:

```
class: SensorModel
location (URL): http://embedded.eecs.berkeley.edu/
                concurrency/models/SensorModel.xml
```

Create two instances of the sensor in a Ptolemy II model with a PN director.

The sensor has some parameters. The *frequency* you should set to 0.1 to match the equations above. The *samplingPeriod* you should set to 0.5 seconds for one of the sensor instances, and 0.75 seconds for the other. You are to perform the following experiments.[3]

(a) Connect each sensor instance to its own instance of the SequencePlotter. Execute the model. You will likely want to change the parameters of the SequencePlotter so that *fillOnWrapup* is `false`, and you will want to set the *X Range* of the plot to, say, "0.0, 50.0" (do this by clicking on the second button from the right at the upper right of each plot). Describe what you see. Do the two sensors accurately reflect the sinusoidal signal? Why do they appear to have different frequencies?

(b) A simple technique for sensor fusion is to simply average sensor data. Construct a model that averages the data from the two sensors by simply adding the samples together and multiplying by 0.5. Plot the resulting signal. Is this signal an improved measurement of the signal? Why or why not? Will this model be able to run forever with bounded memory? Why or why not?

(c) The sensor fusion strategy of averaging the samples can be improved by normalizing the sample rates. For the sample periods given, 0.5 and 0.75, find a way to do this in PN. Comment about whether this technique would work

[3]You may find the actors described in the sidebars on pages 85, 86, 95, and 107 useful.

effectively if the sample periods did not bear such a simple relationship to one another. For example, suppose that instead of 0.5 seconds, the period on the first sensor was 0.500001.

(d) When sensor data is collected at different rates without a simple relationship, one technique that can prove useful is to create **time stamps** for the data and to use those time stamps to improve the measurements. Construct a model that does this, with the objective simply of creating a plot that combines the data from the two sensors in a sensible way.