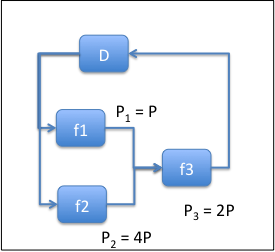
# Simulink Semantics to ESMoL Semantics

Start with a conceptual multi-rate functional dataflow as shown here, where

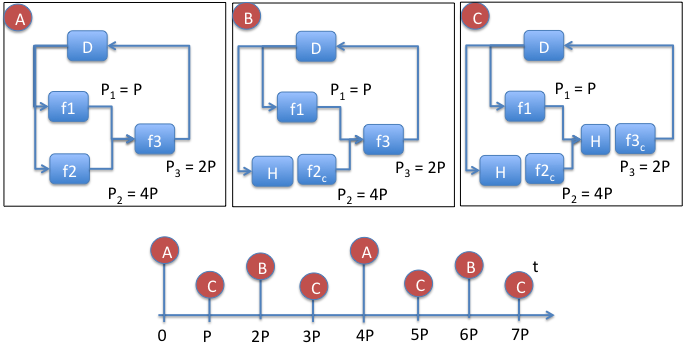
*D* = Dynamics block function (cont. time)

*fi* = controller function (discrete time)

*Pi* = period of *fi*

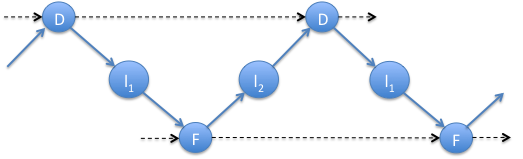
Assume the entire simulation runs at a period of *P*

Then the Simulink semantics result in an execution model that works something like this:



Here the H blocks simply hold the value computed most recently, and the fic block represents the output value of function fi at its last firing. Effectively a different dataflow exists at each of the sub-steps. That dataflow is evaluated instantaneously at each tick, and its results are held in the appropriate places until the next tick.

The ESMoL deployed step semantics are a bit different. The plant simulation has duration (though small), the data transfers between the plant and the controllers have duration (much larger), and the controllers themselves have duration. In the ideal case we have something like the following:



Here the dataflow dependencies are satisfied, and each invocation of the dynamics simulation and the controller sequentially follow one another in perfect synchrony. Ideally all of the runtimes involved in the cycle sum to less than the cycle length (in this case *P*).

The reality of the matter is different. Actually, the clocks of *D* and *F* are not synchronized. Rather they run at the same rate with a small mismatch and some drift. Further, execution of F may be larger than the cycle, so that multiple cycles are involved between the receipt of sensor data and the return of corresponding actuation data.

The questions to answer are:

1. How well does the deterministic Simulink model approximate the possibly non-deterministic data flow of the actual system?
2. How can we model the non-deterministic data flow and dynamics?
3. Is our setup deadlock-free?
4. What can we say about timing? What do we need to say about timing?

Ideas:

1. We are not far off from Kahn semantics, but we are not buffering. How is our model related, and can we use some of Kahn’s ideas to guarantee deadlock-freedom?
2. Look at drift-based LTTA:

[1] [**Approximation, Sampling and Voting in Hybrid Computing Systems**](http://www-verimag.imag.fr/TR/TR-2005-19.pdf)  
Ch. Kossentini and P. Caspi  
*Hybrid Systems Computation and Control, HSCC06*, Santa Barbara, 2006  
Volume 3927 in Lecture Notes in Computer Science

[2] [**Time-Robust discrete control over networked Loosely Time-Triggered Architectures**](http://www.irisa.fr/distribcom/benveniste/pub/LTTA_CDC08.html)  
 P. Caspi and A. Benveniste   
*Proc. of 2008 IEEE Control and Decision Conference.* Cancun, Dec. 9-11, 2008

It looks like if we can adapt the cases described in the second paper to our semantics, and come up with a robustness argument based on continuity of the dynamics, then we’re doing well.

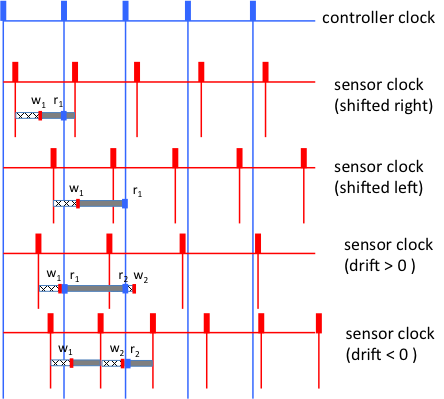
From [2], we have the following system assumptions:

Assumption 1: sampled data overwrite semantics, broadcast data transfers, non-synchronized local clocks, and independent readers/writers

These look good for our configuration.

Case 1: **Reading single signals**

Reworking Figure 2 to match our cases:



If the clock is shifted, then the delay value sensed by the controller changes. For example, in the first two cases above we have respectively no delay and one tick of delay. The other problem (from the paper) is missed or duplicated data. The authors assume that the sampling system doesn’t miss. We do not have that luxury, but we need to make an argument that the continuity of the physical dynamics and the robustness of the controller mitigate data loss and duplication.