Manufacturing / Printing System Simulator

A Design Fest Proposal

Designers of real-time embedded software are often faced with a serious problem. The physical system that their software will control is being designed concurrently and is therefore not available as a test platform for the software. Add to this the usual dangers of running untested software on systems that have inherent safety and property damage risks, and the arguments in favor of simulation environments become quite strong.

An additional use of hardware simulation is in an academic setting where resources do not permit the availability of the physical system to be controlled. Simulation is already a very popular approach in school-based science and engineering laboratories.

Design Fest teams choosing this assignment will design a general-purpose simulator for a mechanical process. This proposal addresses a specific domain within embedded systems: that where software must control the transport of physical objects through various way points where physical actions are applied to them. Some examples from this domain include

- Manufacturing assembly lines
- Printing systems
- Railroads[1]

Software is needed to simulate the physical system so that the software being designed to control that system can be empirically tested. The control software is not part of this proposal.

[1] A rail system may not be the ideal application since control software does not actually control the trains themselves. On the other hand, model railroading might be an ideal application.

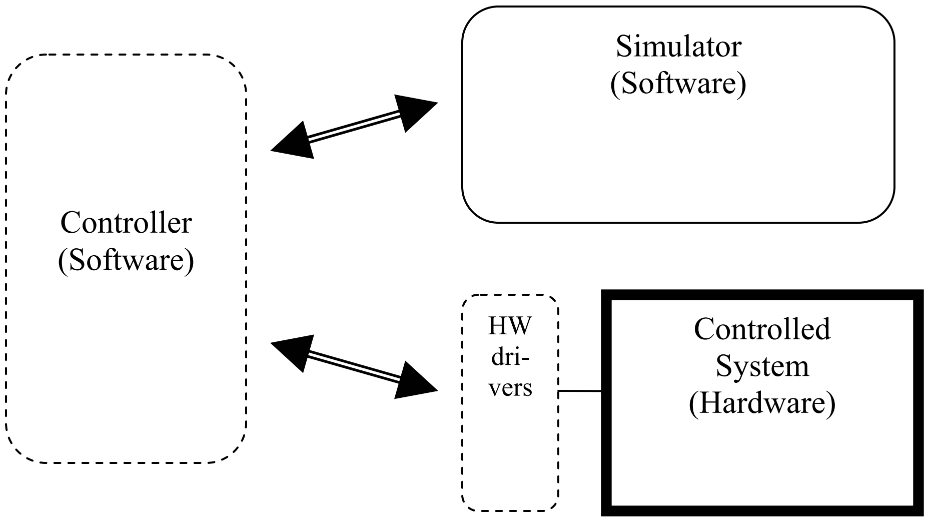

Figure 1. High-Level System Architectures

System Artifact Specification

The environment being simulated is a set of possibly converging and diverging paths on which objects travel. It can be viewed as a graph where the nodes represent way points and the edges represent travel routes (perhaps conveyor belts) between the way points.

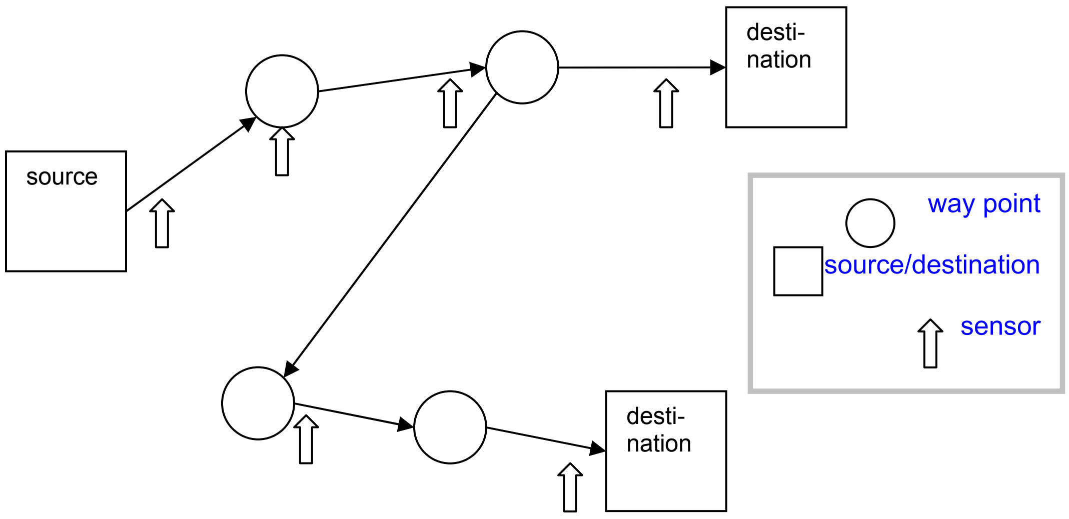

Figure 2. Sample Simulation System

The edges have travel times. The times are probabilistic because the physical devices involved may sometimes slow down or even speed up the rate of travel of objects along them. There is also a probability that a path segment will fail completely, i.e., halt.[2] The way points are control points. Upon commands from the software, objects sitting in way points have their state modified in some way (a lid gets put on a can; ink is printed on a sheet of paper, etc.). Some way points cause their objects to stop and wait there for awhile while the action is being applied. Control software would have to halt and resume the travel of the objects. The time needed for the state change action to be applied is again probabilistic.

Some of the way points act as physical switches, directing arriving objects to one of several outgoing paths. Others passively accept incoming objects from many different paths without external control. However, they must be able to report collisions.

Objects have a *length* expressed as the difference in time between when the leading and the trailing edges of the object arrive at a way station or sensor. Sensors may be placed anywhere in the system – paths or way stations. They report the presence or absence of objects at their locations.

Finally, there are special places at the endpoints of paths. The source points are pre-filled with objects and are capable of sending one object at a time out to the paths to which they are connected. The destination points accumulate objects. Both kinds can report their current stocks. (Source and destination points might actually be represented as special cases of way points.)

Control Interface

In a real system with embedded software control, there are various hardware interfaces that allow information about the system to flow into the control software and allow control signals to go out. For the simulator, that hardware-dependent part of the software (called the HW Drivers in Figure 1) will not be used. Designers must choose an interface more amenable to a concurrently running software system on the other side. This implies some kind of inter-process or intra-process (inter-thread) communication. Network protocol – based communication may or may not be appropriate depending on response time requirements. Many distributed real-time control systems employ special network protocols designed to guarantee, or at least minimize the variation in, response time.

[2] Note that it is not the simulator's job to recover from, or even detect this situation. The simulator simply stops sending sensor reports to the controller; the controller then deduces the problem and recovers.