Network Control Systems

Vincenzo Liberatore[1]

December 9, 2002

[1]Electrical Engineering and Computer Science Department, Case Western Reserve University. 10900 Euclid Avenue, Cleveland, Ohio 44106-7071, USA. Fax: (216) 368 6039. E-mail: vxl11@po.cwru.edu. URL: http://vorlon.cwru.edu/~vxl11/.

Chapter 1

Networked Control Systems

The ncs (Networked Control System) extension to ns makes it possible to simulate physical systems that are attached to a network. The basic ncs agent is the "plant", a historical name used in the industrial automation community. The plant agent represent the interface between the physical and the network dynamics. The plant can take the role of a controller interface, sensor, or actuator depending on its usage in the simulation script. Pairs of plants are connected to each other and exchange sample data and control instructions. The sample data are obtained by the simulation of a physical system and the control instructions are used as the input for the simulation of that system. On the whole, the ncs extension can be used to implement the abstraction shown in Figure 1.1.

As an example, the simulation of a physical system with a remote controller can be implemented by defining two simulated systems: the controlled system and the controller. Plants are then attached to the controlled system (to represent sensors and actuators) and each controller plant is attached to a corresponding system plant. Section 1.3 details this example.

1.1 Tcl Linkage

The plant agents expect the following two functions to be defined in the tcl code.

1.1.1 sysphy

The function sysphy simulates the physical dynamics of the system to which the plant is attached. The function sysphy must maintain global variables t_0, $\mathbf{x}_0$, and $\mathbf{u}_0$ that represent the initial time, the initial system state, and the input that is applied to the system at time t_0 (in general, $\mathbf{x}_0$ and $\mathbf{u}_0$ are vectors). When sysphy is invoked at (simulation) time $t_1 \geq t_0$, sysphy must calculate the new system state at time t_1 as a function of $\mathbf{x}_0$, $\mathbf{u}_0$, and $t_1 - t_0$, update the value of $\mathbf{x}_0$, $\mathbf{u}_0$, set $t_0 \leftarrow t_1$, and return the system output at time t_1. The function sysphy can be invoked either when the plant receives a packet from its peer or when the plant samples the system output. The function signature is:

```
Agent/Plant sysphy {plant-description send-time { system-input } }
```

where plant-description is an alphanumeric description of the plant that invokes sysphy, send-time is the time when a packet was sent by the peer (if sysphy is invoked upon receiving a packet) or the current time (if sysphy is invoked to sample the output), and system-input is the input that must be applied to the system (the system-input list is empty if the system input does not change). The

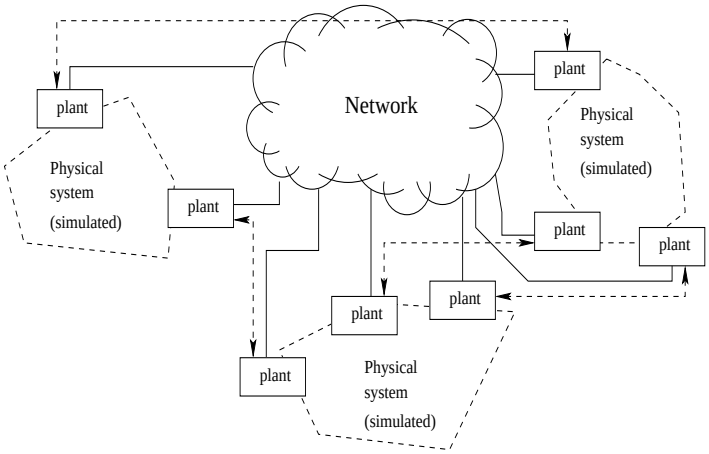

Figure 1.1: Usage of the ncs extension for the co-simulation of network and physical systems.

packet creation time is available in the simulation, but this information is in general impossible to obtain in reality. The return value is the vector $\mathbf{y}_1$ that represent the system output at time t_1. Its format is a list whose first element is the size of $\mathbf{y}_1$ followed by the entries of $\mathbf{y}_1$.

1.1.2 smplschd

The function smplschd can schedule future invocations of system output sampling (if any). The function is especially helpful in the case of sensors to implement a fixed or variable sampling rate. It is also used at the controller site to trigger control messages. The signature is:

```
Agent/Plant smplschd { plant-description origin }
```

where plant-description is an alphanumeric description of the plant agent that invokes sysphy and origin is 1 if smplschd is invoked upon receiving a packet or 0 if smplschd is invoked upon sampling the system output. The function smplschd does not return any value.

1.2 The C++ PlantAgent class

Output samples and control signals are sent within packets of type ncs:

```
struct hdr_ncs {
  int seqno_;
  double send_time;      // timestamp: sending time
  int input_dim;
  double *system_input;  // input for the receiving agent
  static int hdr_ncs::offset_;     // offset for this header
  inline static hdr_ncs* access(const Packet* p) {
                         return (hdr_ncs*) p->access(offset_);
  }
  int& seqno() { return (seqno_); }
};
```

The `PlantAgent` class is derived from `Agent`:

```
class PlantAgent : public Agent {
 public:
  PlantAgent();
  int command(int argc, const char*const* argv);
  void recv(Packet*, Handler*);
 protected:
  int off_plant_;  // ncs header access
  int sent_seqno_; // the seqno of the last packet sent
  int recv_seqno_; // the seqno of the next expected packet
  void sample();   // output sample
  char plant_description_[36];
};
```

The two fundamental methods are `recv` and `sample`. Additionally, `set-description` sets the alphanumeric description of the plant.

1.2.1 recv

The `recv` function invokes the sysphy function to compute the system state and to set the new system input. Then, `recv` invokes smplschd to schedule a future time at which the plant will sample the system output. The `recv` function procedure drops out-of-order packets.

1.2.2 sample

The `sample` command invokes sysphy to compute the current system output, sends it to the plant peer, and then invokes smplschd to schedule future invocations of `sample`.

1.2.3 set-description

The `set-description` command sets an alphanumeric description of the plant. The description is any string of length 32 or less without spaces, tabs, line feeds, or carriage returns. The description is passed as an argument to sysphy and smplschd and can be used to differentiate among different plants.

1.3 Example: Control of a Scalar Plant

The file $\sim ns$/tcl/ex/plant.tcl contains an example with the plant agent. It simulates the topology shown in Figure 1.2. In this case, the three nodes marked as plants are connected to independent and indentical scalar systems. Since the systems are identical, this scripts only simulates the physical dynamics of the first one (but it injects packets for all three plants). The simulation output an ns trace file, a nam file, and a file with the system output over time. Upon exit, it spawns nam as well as xgraph to plot the system output.

1.4 Commands at a Glance

```
set p0 [new Agent/Plant]
```

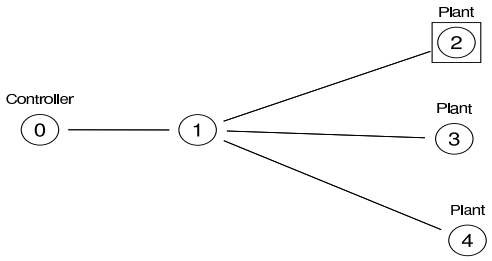

Figure 1.2: Network topology.

Creates a new plant agent.

```
$p0 set-description <description string>
```

Attaches an alphanumeric description to the plant.

```
$ns attach-agent $n0 $p0
```

Attaches plant $p0 to node $n0.

```
$ns connect $c0 $p0
```

Connects the two plant $c0 and $p0.

```
$p0 sample
```

Samples the output of the (simulated) physical system.