Events

Procedure Calls Or Demons

Active Databases

To Use Or Not To Use

Are Traditional MILs/PLs Adequate?

Procedure Calls Or Demons

- **Implicit invocation through demons**
  - not direct procedure invocation:
    E.g.,

```
M1

   if event-1 then do
         M3.operation-x (param-1: type-1, param-2: type-2)
         M5.operation-y (param-1: type-1)
         M7.operation-z (param-1: type-1, param-2: type-2)
   end if
```

  - but
    a component announces (or broadcasts) one or more events;
    other components register an interest in an event
    by associating a procedure with the event
    when the event is announced, the demon invokes all the procedures
    that have been registered for the event

    Thus, an event announcement in one module "implicitly"
    causes
    the invocation of procedures in other modules

- **Where Did We See This?**
  -
  -

Active Databases

Employee

emp#	mgr	salary
123	234	20K
124	234	31K
125	235	46K
126	234	66K
127	235	35K
234	239	75K
235	240	80K
239	245	76K
240	246	85K

Ensuring consistency constraints in DBMSs

integrity constraint: sal < mgr.sal

transaction-1: e.salary <- e.salary + increment

transaction-2: e.salary <- e.salary - decrement

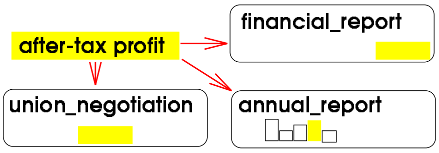
e.g., let e.emp# = 234

Enforcing trigger mechanism

at time t, compute bonus

on receipt of any good performance report, offer promotion

spreadsheet updating, SQL 96



To Use Or Not To

reusability:

a component in system X is used in system Y

as an event announcer: no need to explicitly name other modules

as an event user: no need to explicitly name other modules

+ supported by implicit invocation/demons/actors

evolvability:

replacement of a component by another:

does NOT affect the interfaces of the existing modules

addition of a component:

deletion of a component:

+ demons allow for miminum perturbation of "interfaces"

Performance:

detection of constraint violation, or satisfaction of trigger mechanism:

tend to use more (internal) space for efficient enforcement can be costly

Controllability:

can be difficult to control the order of processing

(e.g., multiple modules reacting to the same event; chaining; exception handling)

Are Traditional MILs/PLs Adequate?

Event Specification

```
Module-1
        declare  Event-1          /* on Event-1, Module-1 should announce it */
                      x: integer;  /* parameter list */
                      y: Module-N.myType
        declare  Event-2          /* parameter-less event */
        when     Event-3 => Method-1 b    /* Event-3 declared in Module-2 */
                                          /* parameter B declared in Event-3 */
        Method-1   ...
```

```
Module-2
        declare  Event-3
                      B: integer;
        when     Event-2 => Method-4
        when     Event-1 => Method-2 x2 y2

        Method-2   ...
        Method-4   ...
```

```
Module-3
        when     Event-2 => Method-3
        when     Event-1 => Method-4 x3 y3
        Method-3
        Method-4   ...
```

Are Traditional MILs/PLs Adequate?

Event Manager

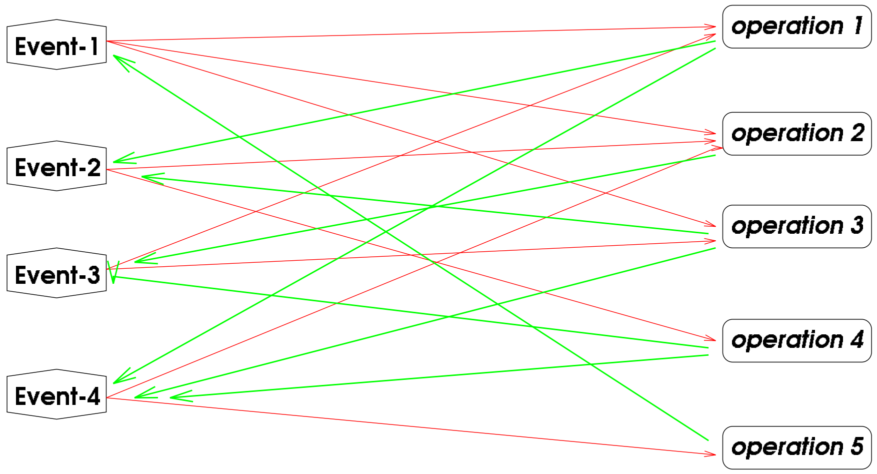
```
Module Event-Manager
        import   Module-1, Module-2, Module-3
        type     Event: (Event-1, Event-2, Event-3)

        case   Event   of
               Event-1:
                         Module-2.Method-2 (param1: integer, param2: ModuleN.myType)
                         Module-3.Method-4 (param1: integer, param2: ModuleN.myType)
               Event-2:
                         Module-2.Method-4;
                         Module-3.Method-3
               Event-3:
                         Module-1.Method-1 (param1: integer)
        end case
```

- for interfacing between event spec. and target language
- specify event-operation relationships
- the compiler has to generate a code for detecting events and for a fair scheduler

Are Traditional MILs/PLs Adequate?

- Fair scheduler:



- Event-1 -> operation-1, operation-2, operation-3

  Event-2 -> operation-2, operation-4

Process Control

Types of Systems

Case Study: Cruise Control

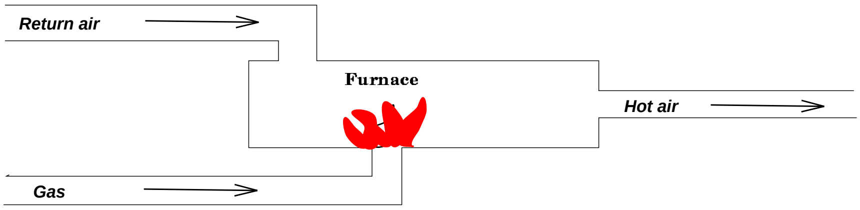
Object View

Process-Control View

Comparison

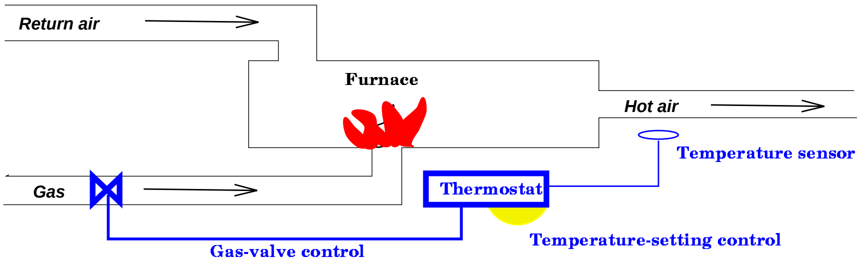
Types of Systems

Open-loop system



- continuous process
- no surveillance

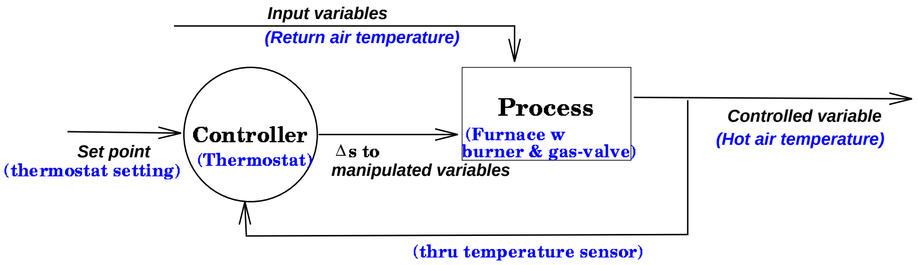
Closed-loop system



- continuous process
- surveillance -> control the process:
  good when the system is subject ot external perturbations

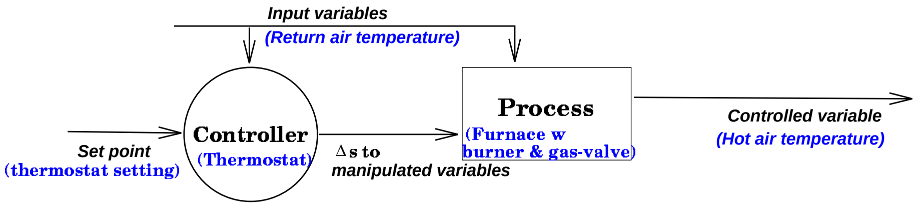
Two Forms of Closed-loop Control

Feedback control



- adjust the process according to measurements of the controlled variable
- simplified (sensor properties, transmission delays, caliberation issues)

Feedforward control

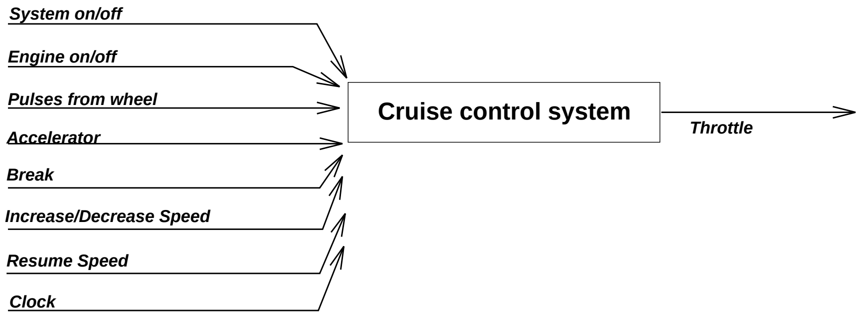


- anticipate future effects on the controlled variable
  valuable when lags in the process delay the effect of control changes
- simplified (sensor properties, transmission delays, caliberation issues, combinations)

Case Study: Cruise Control Architecture

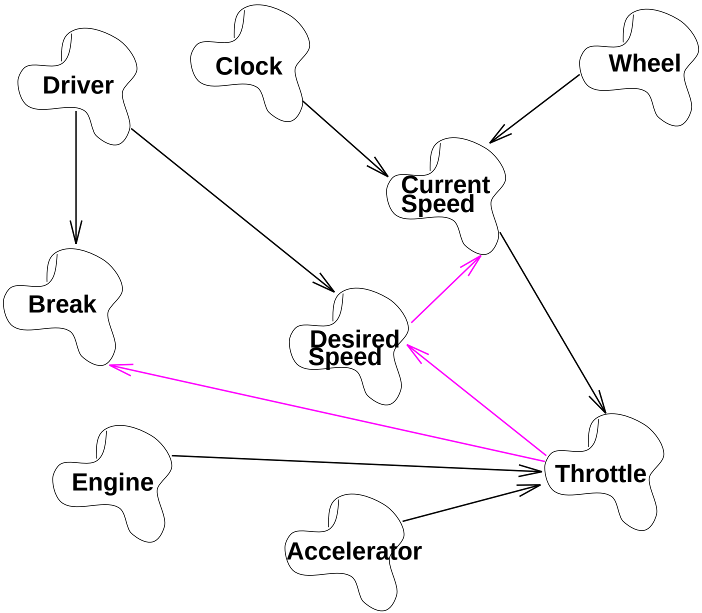
Problem

"A cruise contol system maintains the speed of a car, even over varying terrain. Whenever the system is active,
it determines the desired speed, and controls the engine throttle setting
to maintain that speed."



- ✥ Clock is used only in combination with the wheel pulses (per every revolution) to determine the current speed
- ✣ The system receives a (digital) throttle setting as input & controls the speed
- ⌘ (wheel pulses & clock -> current speed) - (accelerator input, increase/decrease speed -> desired speed)-> change in the throttle setting

Object View of Cruise Control



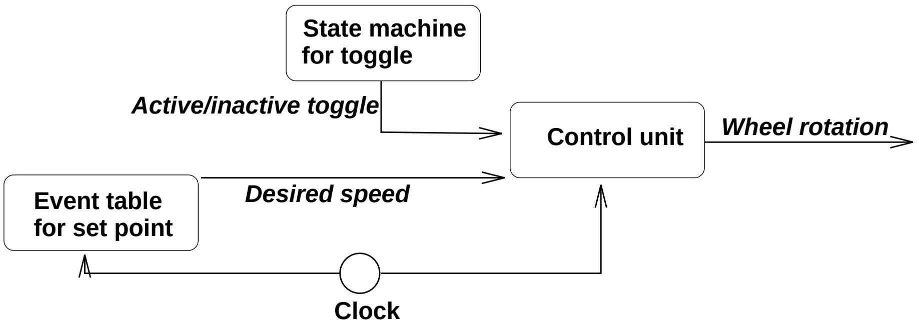
✥ **NFR considerations**

+ identifies important concepts and inter-dependencies

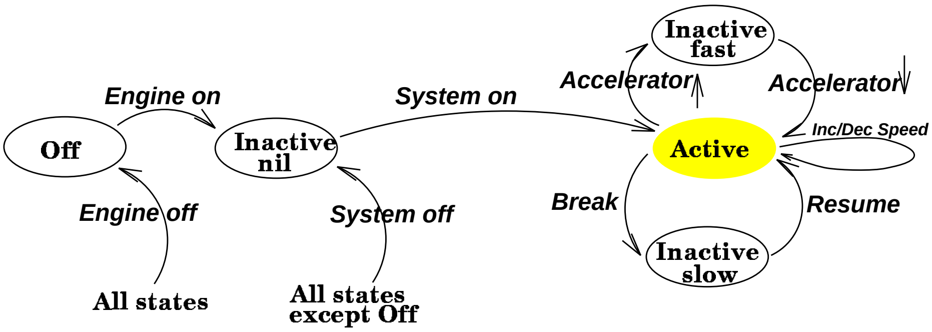
- inability to cope with external perturbations
(variations in terrain, vehicle load, air resistance, fuel quality, etc.)

Process-control View

Architecture

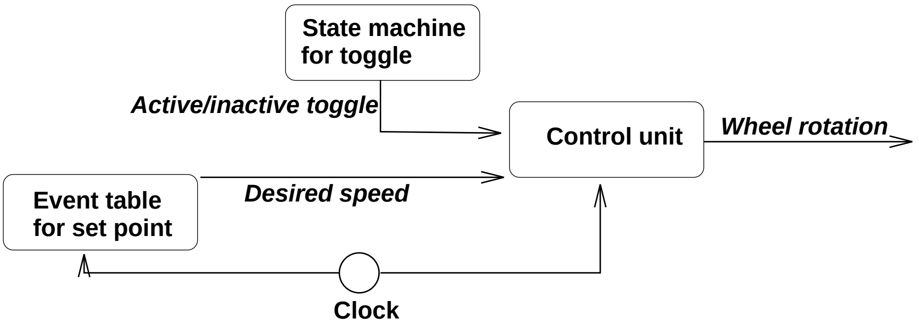


State Machine



Process-control View

Architecture



Event table

Event	Effect on desired speed
Engine off, system off	Set to "undefined"
System on	Set to current speed as estimated from wheel pulses
Increase speed	Increment desired speed by constant
Decrease speed	Decrement desired speed by constant

- **NFR considerations**
  - + separation of concerns: control vs. process
  - + consideration of external perturbation
  - + performance and safety