Verification of Real Time Systems - CS5270
2nd lecture

Hugh Anderson

National University of Singapore
School of Computing

January, 2007

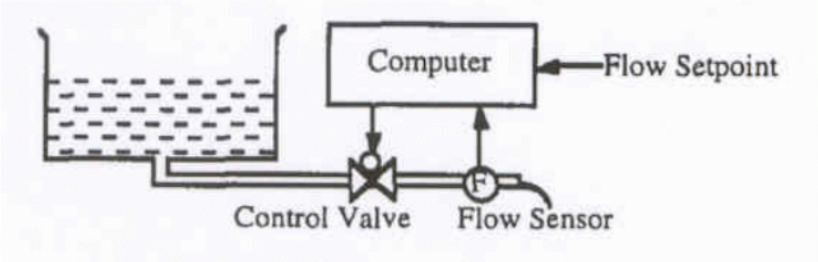
A warning...

Outline

1. Examples of RT systems (Kopetz)
   - Flow in a pipe
   - Engine control
2. The hard real-time environment
   - Requirements for hard real-time systems
   - Time triggered architecture example
   - Clocks and Synchronization
3. The design challenge
   - Notions of the challenge

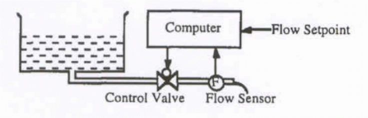
Controlling pipe flow

A simple system (From Kopetz):



Controlling pipe flow

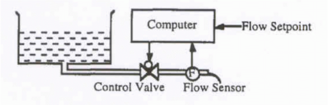
The system is supposed to work as follows:



- Maintain a given flow set point (rate of flow) despite changing environmental conditions - note:
  - Varying level of the liquid in the vessel.
  - temperature of the fluid (affecting its viscosity)
- The computer controls the plant by setting the position of the control valve.
- Flow sensor is used to determine the effect of the control.

Controlling pipe flow

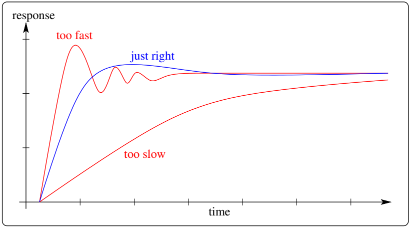
But note that:



- Actuators also have sensors to monitor the effect of control actions:
  - The position of the control valve
  - Two limit switches
    - completely open
    - completely closed
- Often 3-7 sensors for every actuator (not just single sensor/actuator).

Controlling pipe flow

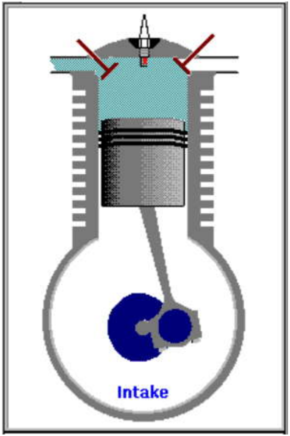
Stability of control is a main issue (Separate topic):



- Output action by the controller will affect the environment after a delay. Observing the effect on the environment will involve a delay introduced by the sensor.
- Measure or derive these delays to implement the temporal control structure...

(Car) Engine control

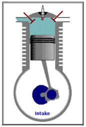
The internal combustion engine



(Car) Engine control

The system is supposed to work as follows:

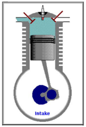
- Calculate the amount of fuel and the moment at which this fuel must be injected into the combustion chamber.



- Fuel amount and injection time depend on:
  - Intentions of the driver (position of the accelerator pedal)
  - Current load on the engine
  - Temperature of the engine
  - The position of the piston in the cylinder...

(Car) Engine control

The dynamics of this system:



- The position of the piston indicated by the measured angular position of the crankshaft.
  - Precision required: 0.1 degree
- At 6000 rpm, 10 msecs for each 360 degree rotation.
- Temporal accuracy (sensing when the crankshaft has passed a particular position):
  - Precision required:3 microseconds

(Car) Engine control

The dynamics:

- Fuel injection by opening a solenoid valve:
  - Delay from the time “open” command issued by the computing system and the time at which valve opens:
    - hundreds of microseconds!
    - Changes depending on environment (temperature...)
  - This delay is measured each cycle and used to compute when the next “open” command to be issued so that fuel is injected at the right time.
- Extremely precise temporal control is required.
  - Incorrect control can damage the engine!
- Up to 100 concurrently executing software tasks must run in tight synchronization.

A first cut

Three areas for overall requirements:

- **Functional**
  - Data collection and signal conditioning
  - Alarms and monitoring
  - Control algorithms
  - User interface
- **Temporal**
  - Sampling rates and accuracy
  - Dead time, jitter, latency
- **Dependability/safety**

Functional requirements

Data collection terms and concepts:

Real time entity: A significant named state variable $\langle \text{Name}, \text{Value} \rangle$.

Continuous RT entity: Can be observed at any point in time (pressure)

Discrete RT entity: Can be observed only between specified occurrences of interesting events (rotation time)

- If $\langle N, v \rangle$ is observed at time t and used at time t', then maximum error $(v' - v)$ depends on temporal accuracy (Δ) and maximum gradient of N during this interval.
- If the gradient is high then Δ must be small and tasks using N must be scheduled often!

Functional requirements

Data collection terms:

- **RT Image:**
  - Current picture of an RT entity.
  - $\langle \mathrm{Name}, \mathrm{time}-\mathrm{of}-\mathrm{observation}, \mathrm{Value} \rangle$
- **Accuracy:**
  - Value ($\mathrm{v}-\mathrm{accuracy}$)
  - Temporal ($\Delta-\mathrm{accuracy}$)

Functional requirements

Data collection terms:

- An RT image is temporally accurate only for a limited time interval.
  - Fast-changing RT entity implies short accuracy for the RT image.
- Only temporally accurate T images must be used in computations.
- Real time data base: All RT entities.
  - This DB must be updated periodically (time-triggered) or immediately after a state change of the RT entity (event-triggered).

Functional requirements

Data collection, Temporal accuracy definition:

$\langle N, t, v \rangle$ is Δ-accurate if the value of N was v at some time in the interval $(t - \Delta, t)$.

RT image	**Maximum change**	**V-accuracy**	**Δ-accuracy**
Piston Position	6000rpm	0.1degrees	3μsec
Accelerator pedal	100%/sec	1%	10msec
Engine load	50%/sec	1%	20msec
Oil temperature	10%/min	1%	6sec

Functional requirements

Signal conditioning:

- The processing steps needed to convert sensor measurements to RT images.
- Sensor produces a raw signal value: (voltage, pressure, ?)
- Collect a sequence of raw signal values and apply an averaging algorithm to reduce measurement error.
- Calibrate and transform to standard measurement units.
- Check for plausibility (sensor error).

Functional requirements

Alarm monitoring:

- Continuously monitor RT entities to detect abnormal process behaviors.
- When an RT entity's value crosses a pre-set alarm threshold: alarm
- Malfunctioning usually produces an alarm shower.
  - Rupture of a pipe
    - pressure, temperature, liquid levels..
- Must identify primary event.

Functional requirements

Alarm monitoring:

- Alarms must be recorded in an alarm log with the time of occurrence of the alarms.
- Time order useful for eliminating secondary alarms.
- Complex plants use knowledge-based systems to assist in alarm analysis.
- Predictable behavior during peak-load alarm situations is vital!
- Performance in rare-event situations is hard to validate in real time systems:
  - Meltdown in nuclear power plant!
- Formal verification!

Functional requirements

Control algorithms:

- Design (and implement) control algorithms to calculate set points for the actuators (to enforce control).
  - Sample the values of RT entities.
  - Execute the control algorithm to calculate the new set points.
  - Output the set point signals to the actuators.
  - Take into account delays, and compensate for random disturbances perturbing the plant.
- Warning: Fuzzy controllers not OK for hard RT

Functional requirements

Man-machine interface:

- Inform the operator of the current state of the controlled object.
- Critical sub-system:
  - Quality, quantity and format of the information presented requires careful engineering. (Therac-25)
  - Protocols for the interface especially in alarm situations are crucial.
- Many computer-related disasters in safety-critical real time systems have been traced to faults at the man-machine interface.
  - Separate topic!

Functional requirements:

Temporal:

- Stringent requirements come from the control loop:
  - The delay between change in the state of the plant (from the desired values) and the correction action should be less than Δ.
- Man-machine interface timing requirements are less stringent.
- The sampling rate must be high enough and the execution of the control loop fast enough to minimize Δ.

Temporal requirements

Deadtime:

Definition: The delay between the observation of the RT entity and the start of the reaction (control action) of the plant.

- Dead time = delay(computer) + delay(plant)
- delay(computer) = execution time of the control loop.
- delay(plant) = the inertial delay; time of arrival of the actuating signal and the change in the state.

Temporal requirements:

The important things are to:

- Minimize dead time!
- Minimize latency jitter:
  - max(delay(computer)) - min(delay(computer))
- Minimize error detection latency:
  - loss or corruption of a message, failure of a node etc. should be detected within a short time with high probability.

Dependability requirements

Terms:

Reliability:

- Failure rate λ = failures/hour, $\frac{1}{\lambda}$ = MTTF: Mean Time To Failure. 10^{-9} failures/hour: ultrahigh reliability requirement

Maintainability:

- Time required to repair a system after a benign failure.
- For maintainability one needs a number of Smallest Replaceable Units connected by Serviceable interfaces.
- Plug is serviceable but less reliable than solder connection.

Reliability and maintainability are in conflict. Mass consumer products focus on reliability at the cost of maintainability.

Dependability requirements

Terms:

Availability

- The fraction of the time the system is ready to provide the service.

Security

- prevent unauthorized access to information and services.

Before TTA there was the CAN bus

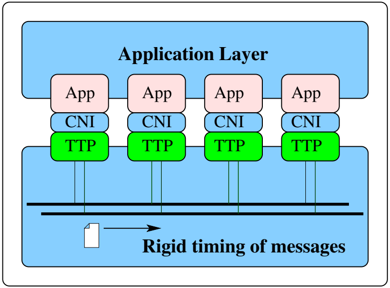
Controller Area Network (CAN) from Wikipedia:

- CAN: a broadcast, differential serial bus standard,
- Developed in the 1980s by Bosch, (Cars!)
- Designed to be robust in electromagnetically noisy environments (Cars)
- The messages it sends are 8 data bytes max, protected by a CRC-15 (polynomial 0x62CC) that guarantees a Hamming bit length of 6 (so up to 5 bits in a row corrupted will be detected by any node on the bus).
- Bit rates up to 1 Mbit/s are possible at networks length below 40 m.

TTA systems are intended to have much larger upper bounds (Also Bosch!).

(As seen before) TTA architecture...

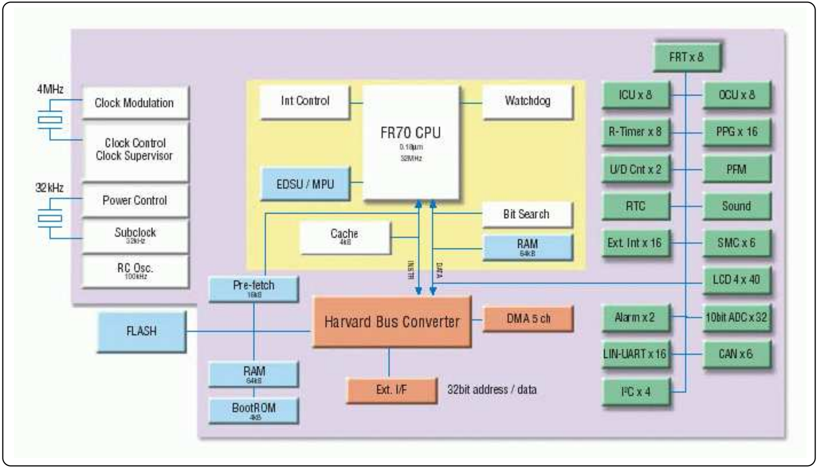
Nodes communicate using CNI, using a TTP



- All the TTPs in a cluster know this schedule.
- All nodes of a cluster have the same notion of global time.
- Fault-tolerant clock synchronization.

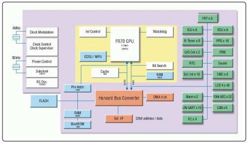
MCU for FlexRay

Sample MicroController Unit:



MCU for FlexRay

Short technical specs:



- 32 bit pipelined RISC CPU, single cycle instruction execution, 512KB flash
- Lots of I/O ... even 10-bit A/D channels
- Lots of timers
- Sample software in development kit includes production quality TT protocol stack, sample code and scheduler...

The main idea

The distributed RT computing system performs a multitude of functions concurrently:

- Monitoring RT entities
  - values and rate of change of values.
- Detecting alarm conditions
- Execution of the control algorithms
- Driving the man-machine interface.

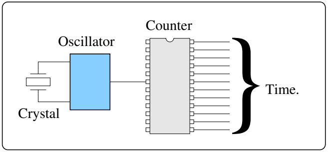
The main idea

Constraint on the behaviour of these nodes:

- Different nodes execute different functions.
- But all nodes must process all events in the same consistent order.
- More generally, all must have the same view of the times at which interesting events have happened.
- A global time base is needed.

Clocks in computers

The hardware:



- Clocks in computers contain a counter - a physical oscillation mechanism that periodically generates an event (microtick) that increments the counter.
- The duration between two consecutive microticks is the granularity of the clock.

They are not perfect 10^{-2} to 10^{-7} secs/sec...

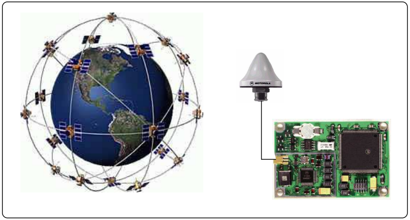
A clock drift disaster: Feb. 25, 1991:

- Accumulated drift over a 100 hour continuous operation (never before experienced) was nearly 343 msecs.
- This lead to a tracking error of 687 meters causing an incoming Scud missile to be declared a false alarm.
- 29 dead and 79 injured. Bug was fixed the next day.

Clocks

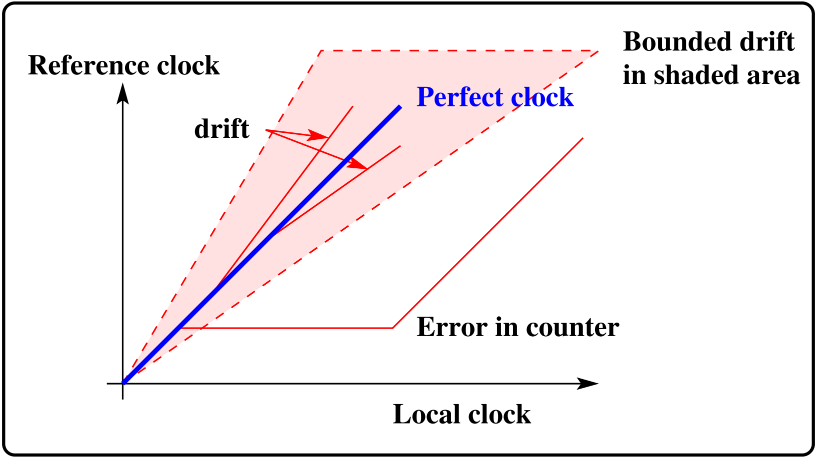
All sorts:

- Global (universal) standard reference clock. (UTC/GMT)
- Have clocks for the nodes, and ensure that the local physical clocks stay locally and globally synchronized.
- NTP? Marzullo's algorithm - smallest interval consistent with largest number of sources (200?μsec accuracy)
- GPS time:



Clocks

When not synchronized, clocks drift...



Clocks

Imagine a (perfect) reference clock:

- In perfect agreement with UTC (!)
- f frequency and hence $g = \frac{1}{f}$ granularity
- If f is large (10^{15}) then digitization error is small.

Time stamps

- Whenever an event e occurs, an omniscient observer (assume!) records the current reference clock time (i.e. the value of its counter) and generate this value as the time stamp of e.
- $t(e)$: the time stamp of the event e.

Clocks

Clock definitions

drift: the frequency ratio between a clock and a reference clock (over a particular time segment). The perfect value is 1.

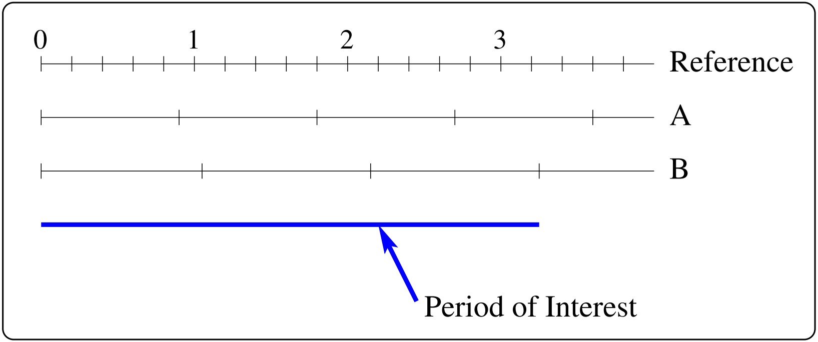
offset: The term offset refers to the time difference between the microticks of two clocks measured in terms of the microticks of the reference clock.

precision: the maximum offset found between a set of clocks measured in terms of the microticks of a reference clock.

accuracy: refers to the maximum offset between a clock and the reference over a particular period of interest.

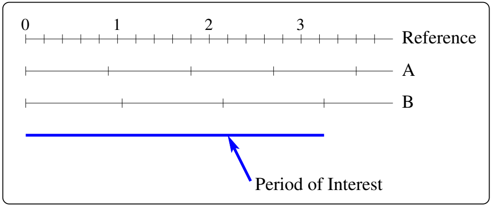
Clocks

Maximum offset occurs at tick 3. Precision is 3 microticks



Clocks

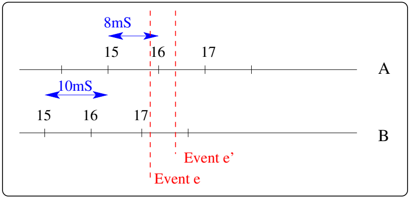
Clock A is running fast, clock B slow:



Maximum offset of A occurs at A's tick 3, which occurs at 13 microticks of the reference clock (an offset of 2 microticks). The maximum offset of B occurs at B's tick 3, which occurs at 16 microticks of the reference clock (an offset of 1 microticks). The accuracy of the collection with respect to the reference is 2 microticks.

Clocks

The term granularity refers to the time between two ticks of the clock:



Assume we have a collection of two clocks A and B whose precision is 10 msecs and whose global time granularity is 8 msecs. It is possible for B to report that events e and e' occur at the same time, although A reports that the events occured one after the other.

The Design Challenge

Two approaches:

- Derive a model of the closed system:
  - Specification/requirements
  - Timing
  - Notion of physical time
- Design and implement:
  - a distributed, fault-tolerant, optimal - real time computing system so that the closed system meets the specification/requirements.

The Design Challenge

Structural elements:

- Each computing node will be assigned a set of tasks to perform the intended functions.
- Task :
  - Execution of a (simple) sequential program.
    - Read the input data
    - The internal state of the task (include RT profiles)
    - Terminate with production of results and updating internal state of the task.
- The (real time) operating system provides the control signal for each initiation of the task.

The Design Challenge

Properties for simple tasks:

- No synchronization point within the task.
- Does not block due to lack of progress by other tasks in the system.
- But can get interrupted (preempted) by the operating system.
- Total execution time can be computed in isolation.
- The Worst Case Execution Time of task over all possible relevant inputs.
  - Correct estimate of WCET is crucial for guaranteeing real time constraints will be met.

The Design Challenge

Properties for complex tasks:

- Contains blocking synchronization statement(s):
  - **wait** semaphore operation.
  - **receive** message operation.
- Must wait till another task has updated a common data structure:
  - Data dependency
  - Sharing
- Must wait for input to arrive.
- WCET of a complex task can not be computed in isolation.

The Design Challenge

For all tasks:

- There will be tasks that are triggered by exceptions, interrupts and alarms.
- There will be tasks that need to be executed periodically.
- These tasks may have precedence relationships.
- These tasks may have deadlines.
- These tasks may share data structures.
- They may have to execute on the same processor.
- We must schedule!