Formal Methods for Exquisite Systems

Embedded and Distributed Real-Time

Hugo Macedo

Universidade do Minho

January 24, 2008

Table of contents

Plan

Devices connected to the physical world

- Better described by its world interaction
- Interaction via sensors and actuators
- Embedded systems
- Control programs
- Modes

Try to apply formal methods

Scenario:
ESA will deploy a robot with a drill in the moon. It should drill x centimeters long and stop.

Problem:

- What are the pre and post conditions?
- How could we check them?
- Possible solutions?
- How to model?

Add the Real-Time Dimension

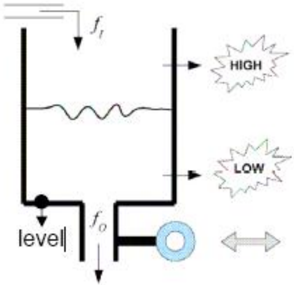
- Scheduling issues
- Time dependability
- Hard/Soft deadlines
- Periodicity

Also the Distributed Dimension

- Synchronous/Asynchronous
- Physical vs Logical Time
- Communication Pattern

Worst Hybrid Systems!

Discrete/Continue Modeling



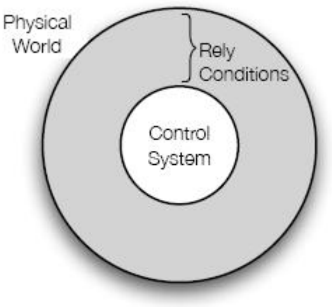
$$\frac{dV}{dt} = f_I - f_O$$

$$f_O = \begin{cases} \frac{\rho \cdot g}{A \cdot R} \cdot V & \text{if valve} = \text{open} \\ 0 & \text{if valve} = \text{closed} \end{cases}$$

Figure: The water tank example

Changes start early

Requirements



Industrially Valid Approach

VDM

- VICE
- CSK successes

Plan

VDM Concurrency

- Concurrency in VDM++ is based on threads
- Threads communicate using shared objects
- Synchronization on shared objects is specified using permission predicates

Threading

Threads

```
thread
(
     dcl id := threadid;
         ...
     while true do
       ...
   )
```

Interpreter Commands

- threads
- curthread (threadid)
- selthreadid

Synchronization

- sync
- mutex
- history counters
- per

History Counters

#req op	The number of times that op has been requested
#act op	The number of times that op has been activated
#fin op	The number of times that op has been finalized
#active op	The number of active executions of op

Synchronization Examples

A buffer with Put and Get operations

```
sync

per Put => #active(Get) = 0

per Get => #active(Put) = 0

per Get => buffer <> []
```

Synchronization Examples

A buffer with Put and Get operations

```
sync

per Put => #active(Get) = 0

per Get => #active(Put) = 0

per Get => buffer <> []
```

Or

```
sync

   mutex(Put, Get)
```

Plan

VDM VICE Extension

- New primitives
- Methodology
- Toolbox extension
- Validation support

Language Extensions

Concurrency

- periodic (threads)
- async

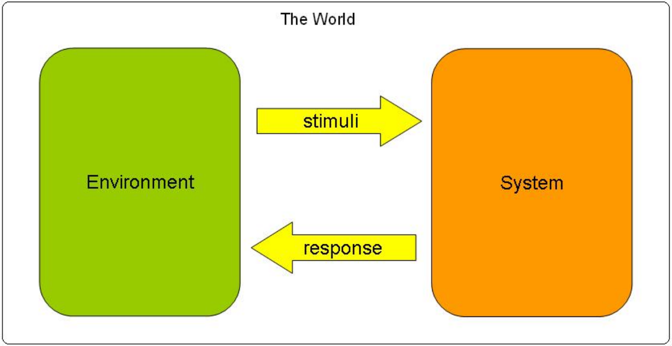
Real-Time

- time
- duration
- cycles

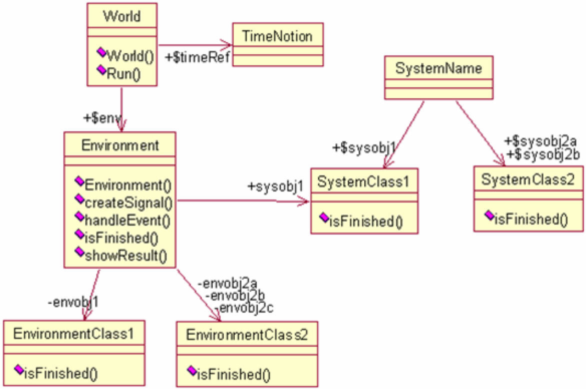
Distribution

- system
- CPU
- BUS

Model paradigm



Model paradigm

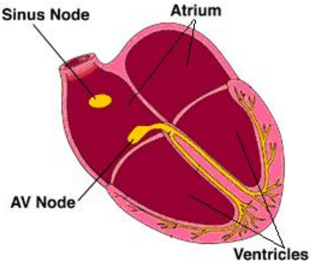


Incremental Development

- VDM-SL
- Sequential
- Concurrent
- Distributed Real-Time

Plan

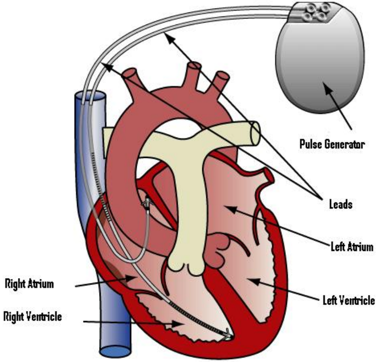
Problem Domain



- I'm not a physician!
- Bradycardia

Problem Domain

- Operating modes
- External unit
- Accelerometer



DOO operating mode

D: Pace Atria and Ventricle

O: Ignore Atria senses

O: Ignore Ventricle senses

Requirements

FixedAV: There must be a 1500 ms period between an atrial event and a ventricular pace.

LRL: The number of pulses per minute delivered in atria must be at least 60.

URL: The number of pulses per minute delivered in ventricle must be at most 120.

Abstract

- Eliciting requirements
- Single function
- DC

Example: DOO Mode

Example: DOO Mode

```
Chamber = <ATRIA> | <VENTRICLE>;

SensedTimeline = set of (Chamber * Time);

ReactionTimeline = set of (Chamber * Time);
```

Example: DOO Mode

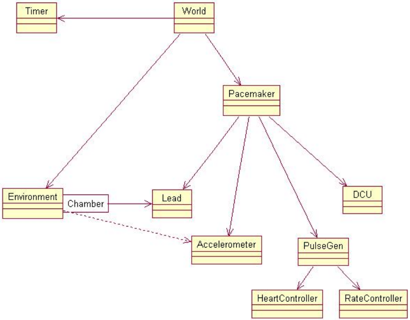
```
Chamber = <ATRIA> | <VENTRICLE>;

SensedTimeline = set of (Chamber * Time);

ReactionTimeline = set of (Chamber * Time);

Pacemaker (mk_(inp,n) : SensedTimeline * Time) r : ReactionTimeline
post let nPulsesAtria = card {i | i in set r & i.#1 = <ATRIA>},
         nPulsesVentricle = card {i | i in set r & i.#1 = <VENTRICLE>}

     in  nPulsesAtria / n >= (LRL / 60) / 1000
         and
         nPulsesVentricle / n <= (URL / 120) / 1000

         and

         forall mk_(<ATRIA>,ta) in set r &
             (exists mk_(<VENTRICLE>,tv) in set r & tv = ta + FixedAV) ;
```

Sequential

- OO model
- Sequential DR-T
- Env/System
- CCS

Sequential

- OO model
- Sequential DR-T
- Env/System
- CCS



Example: Environment

```
public
Run: () ==> ()
Run () ==
   (
    while not (isFinished())
    do
      (
       createSignal();
       Pacemaker‘rateController.Step();
       Pacemaker‘heartController.Step();
       World‘timerRef.StepTime();
      );
    );
```

Concurrent

- Free the concurrency
- Wait/Notify
- Permission predicates

Example: Environment

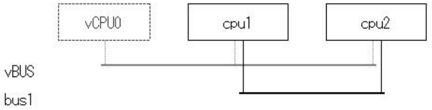
```
thread
  (
   start(new ClockTick(threadid));

   while not finished() do
     ( if busy
       then createSignal();

       World`timerRef.NotifyAndIncTime();
       World`timerRef.WaitRelative(0);
     );

  );
```

Example: Environment

```
thread
  (
   start(new ClockTick(threadid));

   while not finished() do
     ( if busy
       then createSignal();

       World`timerRef.NotifyAndIncTime();
       World`timerRef.WaitRelative(0);
     );

  );

sync
mutex (handleEvent,showResult);
mutex (createSignal);

per isFinished => not busy and time >= simtime;
```

Distributed Real-Time

- Time modelling
- Physical architecture
- VDMTools time



Examples

Environment

```
thread
  periodic (1000,10,900,0) (createSignal);
```

Examples

Environment

```
thread
  periodic (1000,10,900,0) (createSignal);
```

Lead

Requirement: Pulses must have 4 ms width.

```
private
dischargePulse : Pulse * Chamber ==> ()
dischargePulse (p,c) ==
   duration(4)
   World‘env.handleEvent(p,c,time);
```

Examples

System

```
instance variables

 cpu1 : CPU := new CPU(<FCFS>,1E6);
 cpu2 : CPU := new CPU(<FCFS>,1E6);
 cpu3 : CPU := new CPU(<FCFS>,1E6);
 cpu4 : CPU := new CPU(<FP>,1E6);
```

Examples

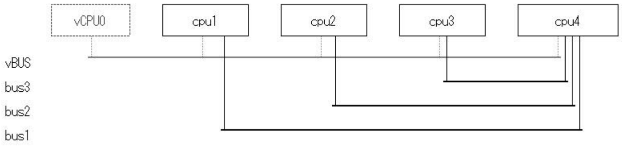
System

```
instance variables

 cpu1 : CPU := new CPU(<FCFS>,1E6);
 cpu2 : CPU := new CPU(<FCFS>,1E6);
 cpu3 : CPU := new CPU(<FCFS>,1E6);
 cpu4 : CPU := new CPU(<FP>,1E6);
```

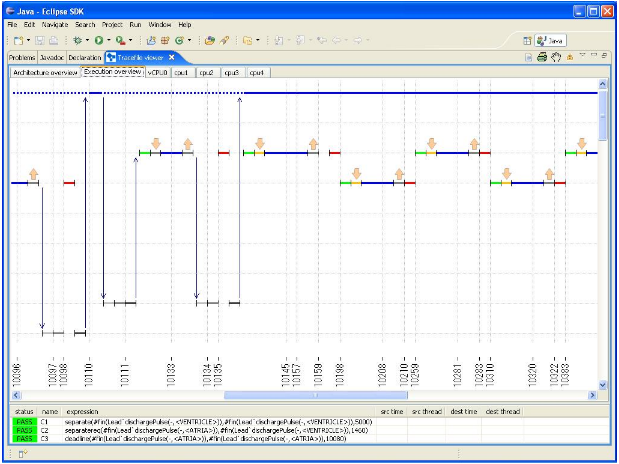
Connecting CPU's

```
 -- Lead (artia) <-> HeartController
 bus1 : BUS := new BUS(<FCFS>,1E6,{cpu1,cpu4});

 -- Lead (ventricle) <-> HeartController
 bus2 : BUS := new BUS(<FCFS>,1E6,{cpu2,cpu4});

 -- Accelerometer <-> RateController
 bus3 : BUS := new BUS(<FCFS>,1E6,{cpu3,cpu4});
```

Architecture Visualization



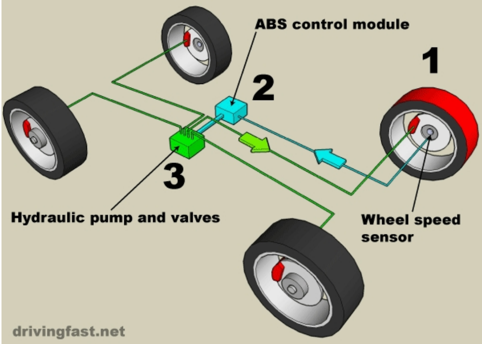
Validation


Java - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Java
Problems Javadoc Declaration Tracefile viewer
Architecture overview Execution overview vCPU0 cpu1 cpu2 cpu3 cpu4
10096 10097 10098 10110 10111 10133 10134 10135 10145 10157 10159 10198 10208 10210 10259 10281 10283 10310 10320 10322 10383
status name expression src time src thread dest time dest thread
PASS C1 separate(#fin(Lead`dischargePulse(-, <VENTRICLE>)),#fin(Lead`dischargePulse(-, <VENTRICLE>)),5000)
PASS C2 separatereq(#fin(Lead`dischargePulse(-, <ATRIA>)),#fin(Lead`dischargePulse(-, <VENTRICLE>)),1460)
PASS C3 deadline(#fin(Lead`dischargePulse(-, <ATRIA>)),#fin(Lead`dischargePulse(-, <ATRIA>)),10080)

Plan

A Simple ABS System

A typical ABS is composed of a central electronic unit, four speed sensors (one for each wheel), and two or more hydraulic valves on the brake circuit. The electronic unit constantly monitors the rotation speed of each wheel. When it senses that any number of wheels are rotating considerably slower than the others it moves the valves to decrease the pressure on the braking circuit, effectively reducing the braking force on that wheel. This process is repeated continuously, and this causes the characteristic pulsing feel through the brake pedal. A typical anti-lock system can apply and release braking pressure up to 20 times a second.


ABS control module
1
2
3
Hydraulic pump and valves
Wheel speed
sensor
drivingfast.net