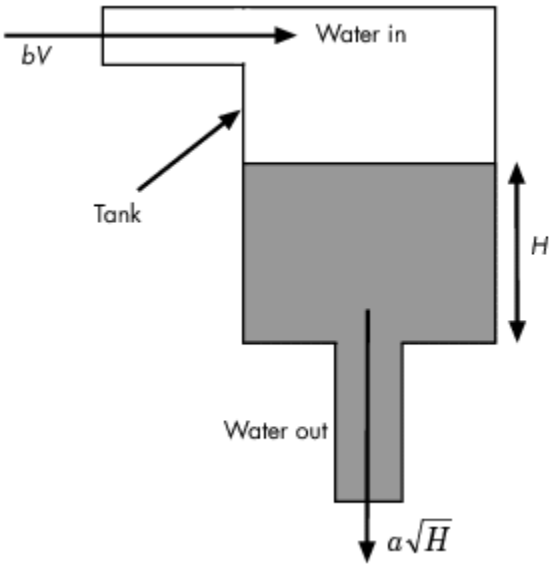
# MARTe Tutorial

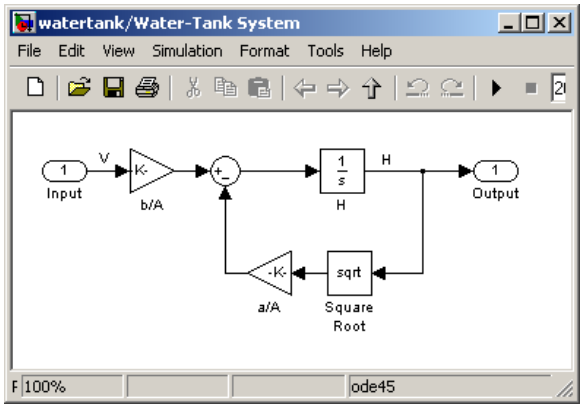
**André Neto***, F. Sartori,
D. Alves, A. Barbalace,
L . Boncagni, G. De Tommasi,
G. Manduchi, F. Piccolo,
R. Vitelli, D.F. Valcárcel,
L. Zabeo and
EFDA-JET PPCC contributors

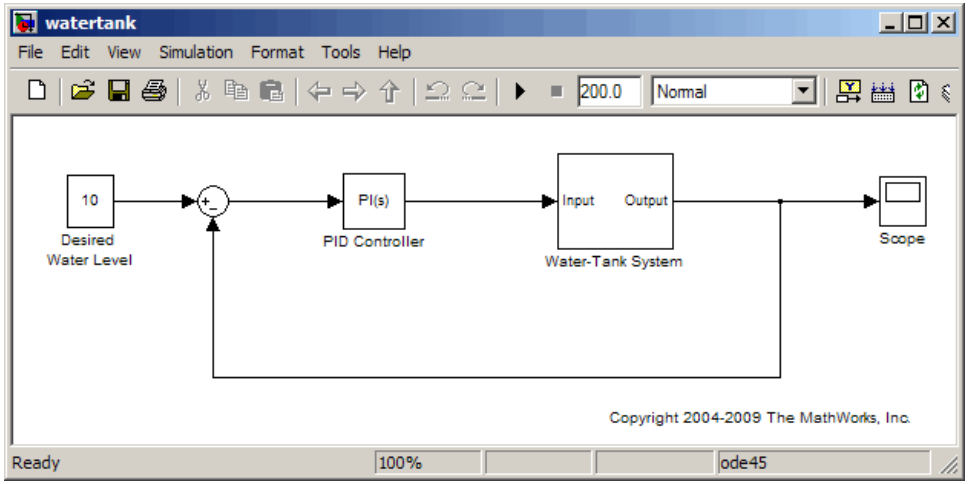
*Instituto de Plasmas e Fusão Nuclear
Instituto Superior Técnico
Lisbon, Portugal
http://www.ipfn.ist.utl.pt

# The water tank



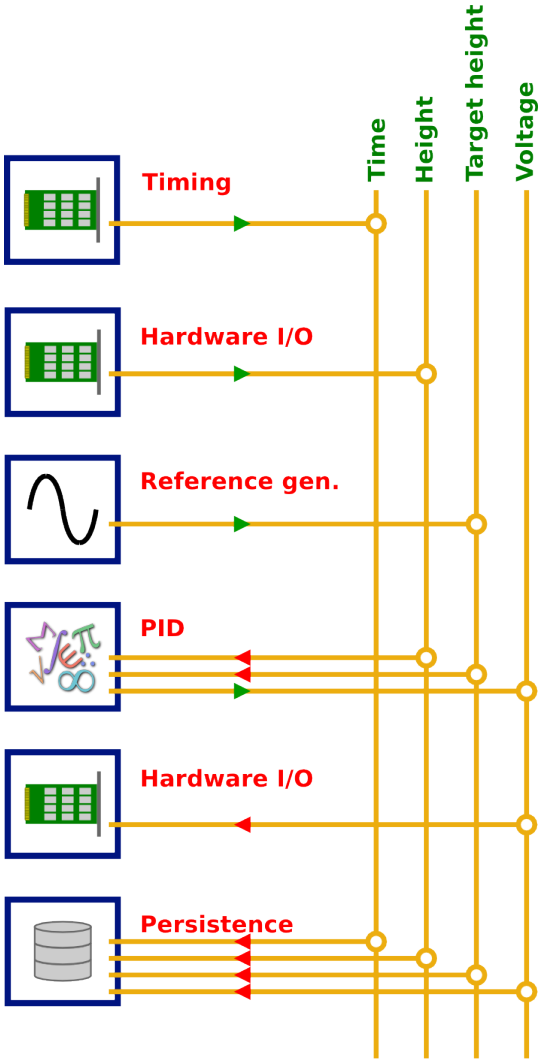

$$\frac{d}{dt}Vol = A\frac{dH}{dt} = bV - a\sqrt{H}$$

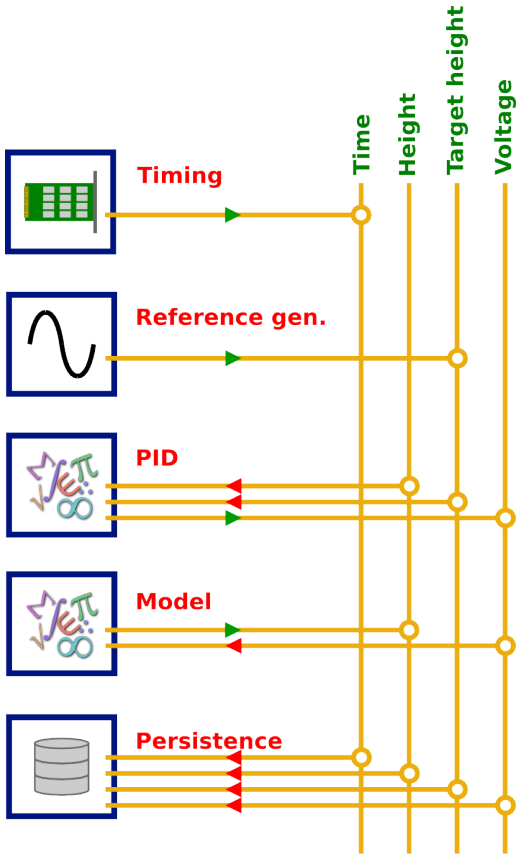



- Vol – volume of water in tank
- A – cross-sectional area of water in tank
- b – constant related to flow rate into the tank
- a – constant related to flow rate out of the tank
- H – height of water

# What GAMs for the water tank?

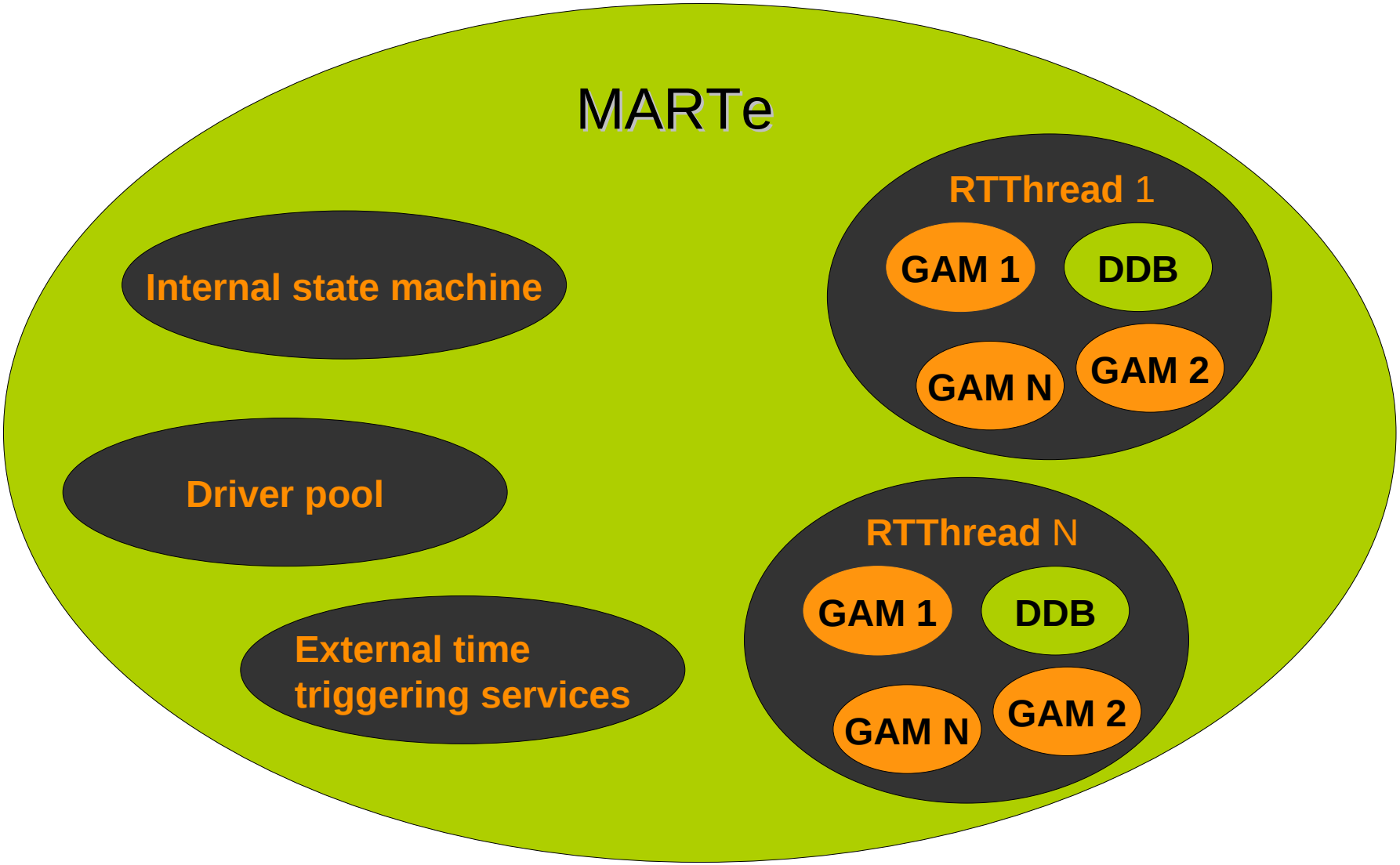


# Development cycle

- What are my needs?
  - Interfaces to hardware
  - Algorithm execution
  - Plant simulation
  - Connection between components (DDB)
  - Interfaces to outside world
- What do I have ready to be used?
  - Recycle hardware interfaces
  - Reuse algorithms

# MARTe World


MARTe
Internal state machine
Driver pool
External time triggering services
RTThread 1
GAM 1
DDB
GAM N
GAM 2
RTThread N
GAM 1
DDB
GAM N
GAM 2

# Requirements for our goal

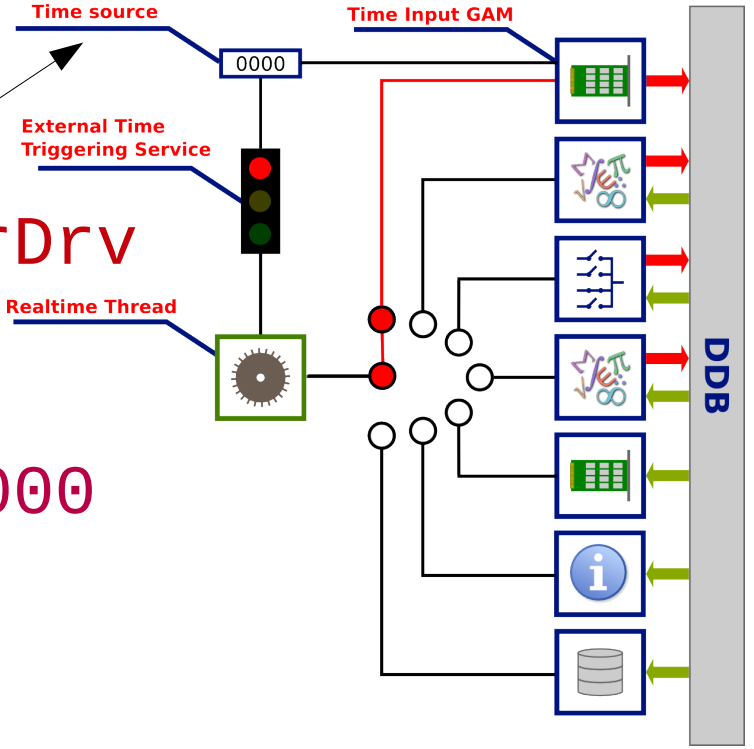
- Development of a water tank simulator
  - Time provider (timer)
  - Reference generation
  - A GAM with the model of the plant
    - water tank
    - pump power supply
  - PID
  - Data downloading
  - External triggering of the state machine

# Skeleton configuration file

```
+MARTe = {
   Class = MARTeContainer
   StateMachineName = StateMachine
   MenuContainerName = MARTe
   +DriverPool = {...}
   +Messages = {...}
   +ExternalTimeTriggeringService = {...}
   +Thread_1 = {...}
}
```

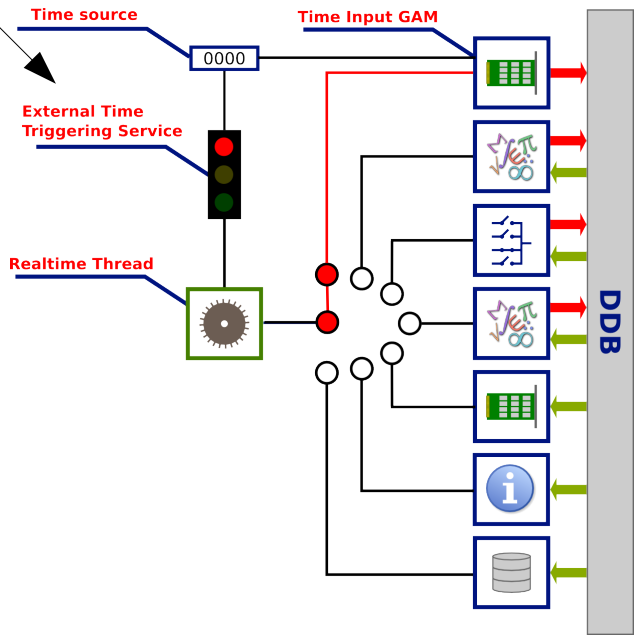
# Generic Timer

```
+MARTe = {
   ...
   +DriverPool = {
      +TimerBoard = {
         Class = GenericTimerDrv
         NumberOfInputs = 2
         NumberOfOutputs = 0
         TimerUsecPeriod = 1000
      }
   }
   ...
}
```



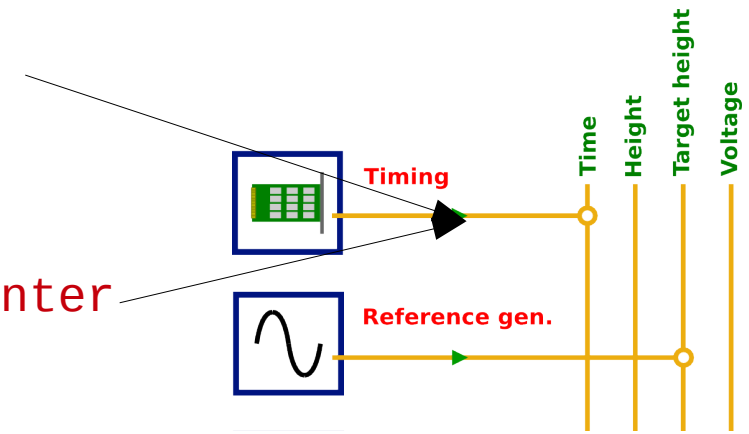
# External Time & Trigger

```
+MARTe = {
   ...
   +ExternalTimeTriggeringService = {
      Class = InterruptDrivenTTS
      TsOnlineUsecPeriod = 250
      TsOnlineUsecPhase = 0
      TsOfflineUsecPeriod = 10000
      TsOfflineUsecPhase = 0
      TimeModule = {
         BoardName = TimerBoard
      }
   }
   ...
}
```



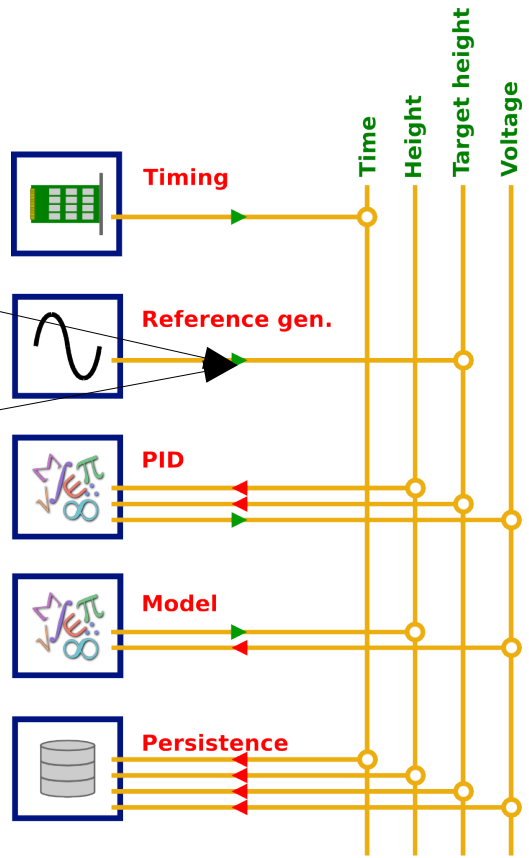
# IOGAM (Timer)

```
+Thread_1 = {
    ...
    +Timer = {
        Class = IOGAMs::TimeInputGAM
        TriggeringServiceName = ExternalTimeTriggeringService
        BoardName = TimerBoard
        Signals = {
            time = {
                SignalName = usecTime
                SignalType = int32
            }
            counter = {
                SignalName = timerCounter
                SignalType = int32
            }
        }
    }
    ...
}
```



# Reference Generator

```
+Thread_1 = {
    ...
    +WaveformGen = {
        Class = WaveformGenerator
        UsecTime = usecTime
        +waterHeightReference = {
            Class = WaveformClassSine
            Frequency = 0.1
            Gain = 1
            Offset = 2.5
        }
        +zeroSignal = {
            Class = WaveformClassPoints
            TimeVector = {0 1}
            ValueVector = {0 0}
            Frequency = 1
        }
    }
    ...
}
```



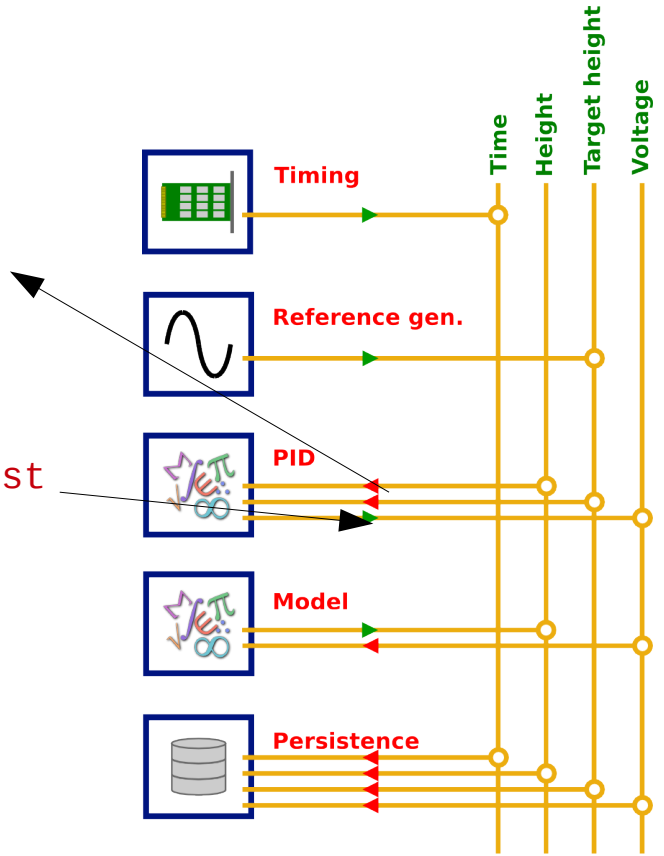
# PID GAM (1)

```
+Thread_1 = {
    ...
    +PIDGAM = {
        Class = PIDGAM
        TStart = 0.0
        TEnd = 10000.0
        InputSignals = {
            PIDInput = {
                SignalName = PIDIn
                SignalType = PIDGAMInputStructure
                FlatNamed = True
            }
        }
        OutputSignals = {
            PIDOutput = {
                SignalName = PIDOut
                SignalType = PIDGAMOutputStructure
                FlatNamed = True
            }
        }
        ...
```

```
struct PIDGAMInputStructure {
    /** Time signal */
    int32                       usecTime;
    /** Reference signal to be      followed */
    float                       reference;
    /** Measurement signal */
    float                       measurement;
    /** Feedforward control */
    float                       feedforward;
};
```

# PID GAM (2)

```
+Thread_1 = {
    +PIDGAM = {
        ...
        Remappings = {
            InputInterface = {
                usecTime = usecTime
                reference = waterHeightReference
                measurament = waterHeight
                feedforward = zeroSignal
            }
            OutputInterface = {
                controlSignal = pumpVoltageRequest
                feedback = pumpVoltageRequest
                error = pidHeightError
                integratorState = pidIntState
            }
        }
        Kp = 3.00
        Ki = 2.00
        Kd = 0.20
        SamplingTime = 0.001
        ControllerOn = On
    }
    ...
```



# Water Tank model

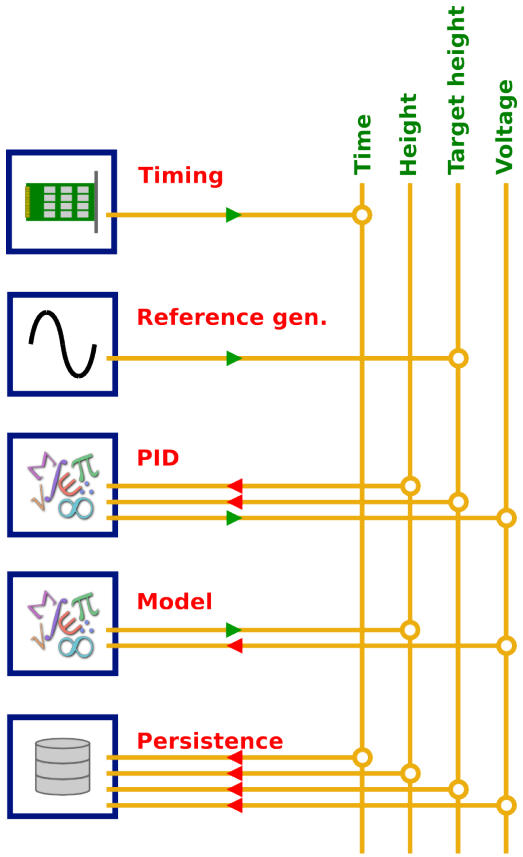
- Only GAM not readily available
- GAM development cycle
  - Design algorithm
    - Piece of paper
    - Software (matlab, octave, ...)
  - Decide inputs and outputs
  - What parameters are configurable?
    - What parameters are compulsory?

# Water Tank variables

```
class WaterTank : public GAM, public HttpInterface {
...
// Parameters
private:
    /** Last usec time (for the integral) */
    int32                                       lastUsecTime;
    /** Last water height (for the integral) */
    float                                       lastHeight;
    /** Last voltage value after saturation*/
    float                                       lastVoltage;
    /** The input flow rate constant*/
    float                                       bFlowRate;
    /** The output flow rate constant */
    float                                       aFlowRate;
    /** Tank area */
    float                                       tankArea;
    /** Maximum voltage that can be requested */
    float                                       maxVoltage;
...
};
```

# What input/output signals?

- Input
  - Time
  - Requested voltage from PID
- Output
  - Water height



# Water Tank read config.

```
bool WaterTank::Initialise(ConfigurationDataBase& cdbData) {
    if(!AddInputInterface(input,"InputInterface")){
            AssertErrorCondition(InitialisationError,
"WaterTank::Initialise: %s failed to add input interface", Name());
           return False;
        }
...
    if(!cdb->Move("InputSignals")){
                AssertErrorCondition(InitialisationError,
"WaterTank::Initialise: %s did not specify InputSignals entry", Name());
           return False;
    }
...
    if(!cdb.ReadFloat(aFlowRate, "aFlowRate", 20)){
            AssertErrorCondition(Information, "WaterTank %s::Initialise:
output flow rate not specified. Using default %f", Name(), aFlowRate);
        }
...
    if(!cdb.ReadFloat(tankArea, "TankArea", 20)){
            AssertErrorCondition(Information, "WaterTank %s::Initialise:
tank area not specified. Using default %f", Name(), aFlowRate);
        }
}
```

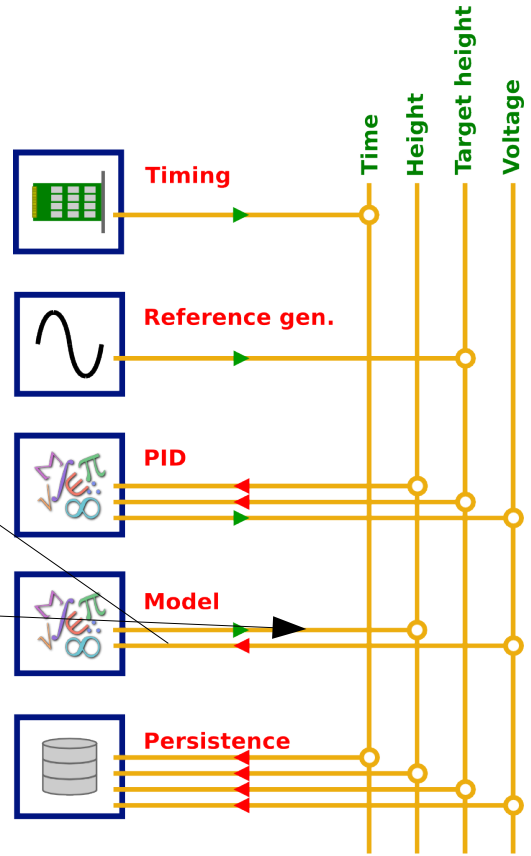
**Input signals from DDB**

# Water Tank execution

```
bool WaterTank::Execute(GAM_FunctionNumbers functionNumber) {
    // Get input and output data pointers
    input->Read();
    int32 usecTime  = *((int32*)input->Buffer());
    float voltage       = ((float *)input->Buffer())[1];
    float *outputBuff = (float*)  output->Buffer();
    float height         = 0;
...
    //Saturate voltage
    if(voltage > maxVoltage){
                voltage = maxVoltage;
    }
    if(voltage < minVoltage){
                voltage = minVoltage;
    }
    //simple Euler method
    height   = (voltage * bFlowRate - aFlowRate * sqrt(lastHeight)) / tankArea
* (usecTime - lastUsecTime) * 1e-6 + lastHeight;
    lastHeight   = height;
...
    *outputBuff  = height;
...
    // Update the data output buffer
    output->Write();
}
```

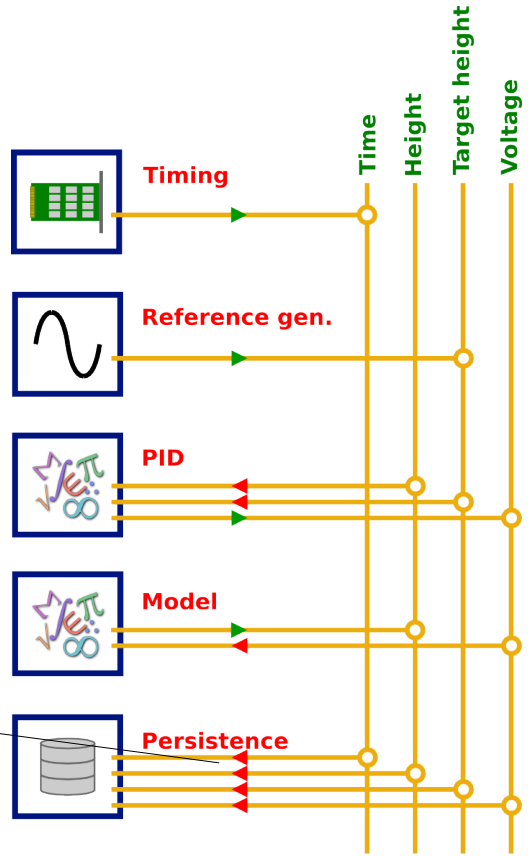
# Water tank configuration

```
+Thread_1 = {
    +WaterTank = {
        Class = WaterTank
        InputSignals = {
            usecTime = {
                SignalName = usecTime
                SignalType = int32
            }
            voltage = {
                SignalName = pumpVoltageRequest
                SignalType = float
            }
        }
        OutputSignals = {
            height = {
                SignalName = waterHeight
                SignalType = float
            }
            pumpVoltage = {
                SignalName = pumpVoltage
                SignalType = float
            }
        }
        aFlowRate = 20.0
        TankArea = 20.0
        ...
```



# Data collection

```
+Thread_1 = {
    ...
    +Collection = {
        Class = CollectionGAMs::DataCollectionGAM
        EventTrigger = {
            TimeWindow0 = {
                NOfSamples = 40000
                UsecPeriod = 250
            }
        }
        Signals = {
            CLOCK = {
                SignalName = usecTime
                JPFName = "TIME"
                SignalType = int32
            }
            WaterHeight = {
                SignalName = waterHeight
                JPFName = "WaterHeight"
                SignalType = float
            }
            ...
```



# Plotting and Webstatistics

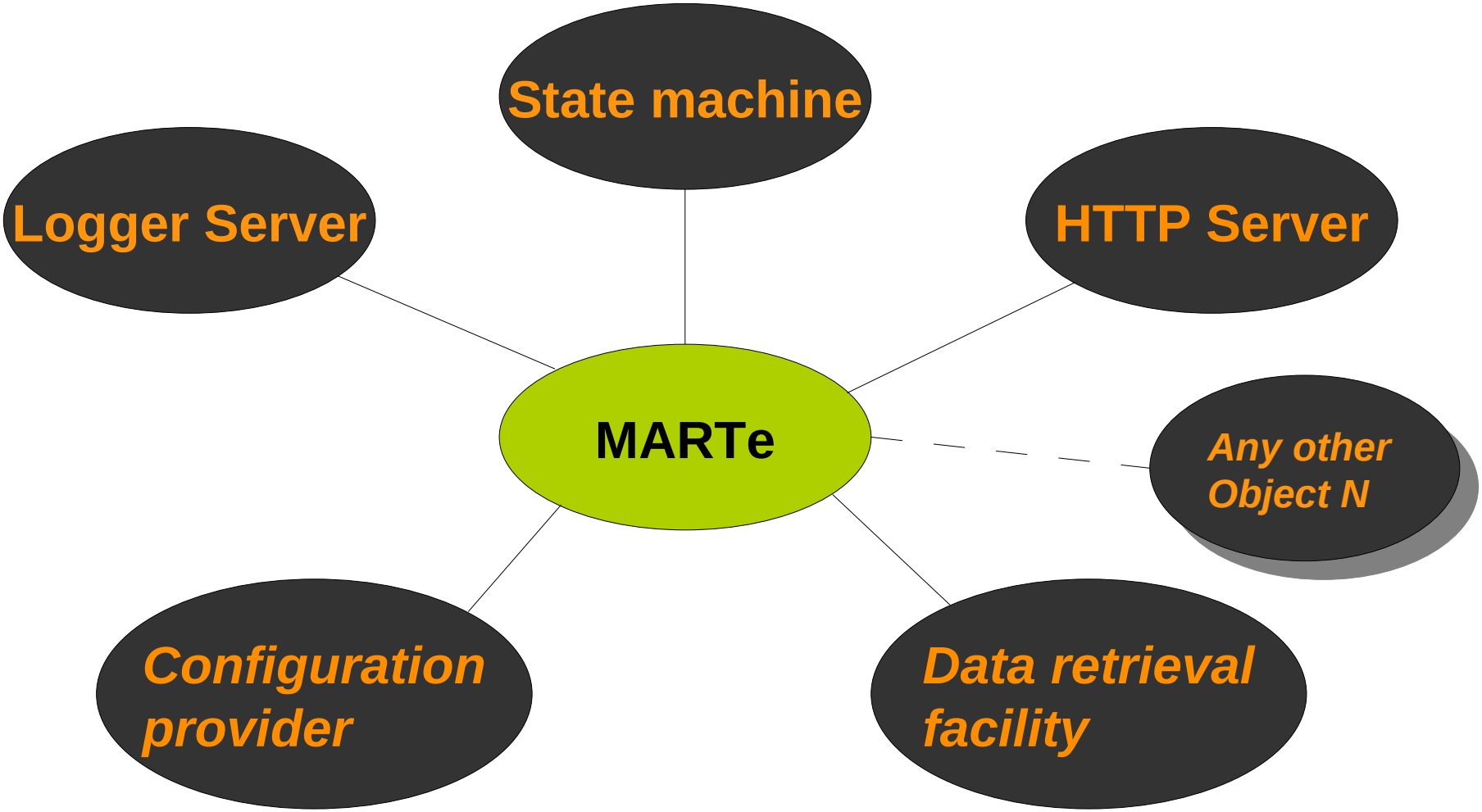
- GAMs readily available
- Display data using the HTTP interface
- Extremely useful to quickly debug and inspect signals in the DDB
- More information on the example slides

# Execution order

- MARTe has two runtime cycles
- Online is associated with the real-time cycle, whereas offline is the stand-by mode
- GAMs can be Online *forever*

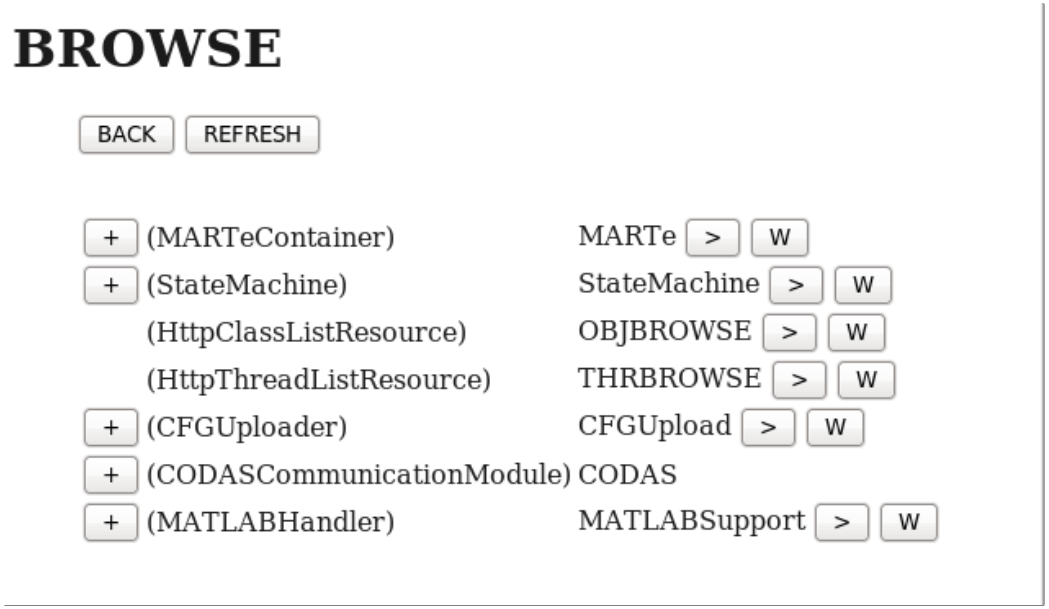

```
+Thread_1 = {
    ...
    Online = "Timer WaveformGen PIDGAM WaterTank
PlottingGAM Statistic Collection"
    Offline = "Timer PlottingGAM Statistic"
}
```

# MARTe Universe



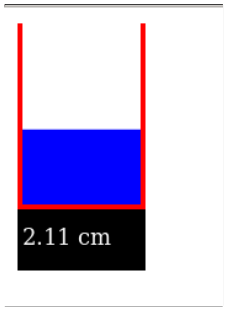
# MARTe Universe components

```
LoggerAddress = localhost
DefaultCPUs = 8
+HTTPSERVER= {
     Class = HttpService
     Port = 8084
     Root = WEB
}
+WEB= {
     Class = HttpGroupResource
     +BROWSE = {
          Class = HttpGCRCBrowser
          Title = "Http Object
browser"
          AddReference = "MARTe
StateMachine OBJBROWSE THRBROWSE
CFGUpload MATLABSupport"
     }
}
+MATLABSupport = {
    Class = MATLABHandler
}
+CFGUpload = {
    Class = CFGUploader
}
+StateMachine = {
...
```



# GAMs and HTTP interface

- GAMs may expose information about themselves using the HTTP interface
  - Write to a stream facility which is provided every time an HTTP request for their URL is performed




| Time window n. | NumOfSamples | UsecPeriod | Start | End |
|---|---|---|---|---|
| 0 | 40000 | 250 | 0 | 10000000 |

**Total Samples = 40000**

**Total already read Samples = 39102**

**Current statistics:**

Data was updated 0.000117 seconds ago

| | Last value | Mean | Variance | Abs Max | Abs Min | Rel Max | Rel Min | Type |
|---|---|---|---|---|---|---|---|---|
| usecTime | 5.565e+006 | 5.311e+006 | 6.426e+010 | 5.565e+006 | 9.918e+005 | 5.565e+006 | 9.918e+005 | int32 |
| CycleUsecTime | 2.505e-004 | 2.529e-004 | 2.734e-009 | 7.997e-003 | 1.631e-005 | 7.997e-003 | 1.631e-005 | float |
| waterHeightReference | 2.068e+000 | 2.219e+000 | 2.369e-002 | 3.500e+000 | 2.068e+000 | 3.500e+000 | 2.068e+000 | float |
| waterHeight | 2.121e+000 | 2.278e+000 | 2.501e-002 | 3.540e+000 | 2.121e+000 | 3.540e+000 | 2.121e+000 | float |
| pumpVoltageRequest | 3.453e-001 | 3.575e-001 | 5.719e-003 | 1.220e+001 | -5.590e+001 | 1.220e+001 | -5.590e+001 | float |
| pumpVoltage | 3.453e-001 | 3.585e-001 | 4.261e-003 | 5.000e+000 | 0.000e+000 | 5.000e+000 | 0.000e+000 | float |

**Integer 32 bits signals**

| | Decimal | Hex | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| usecTime | 5565500 | 0x54ec3c | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

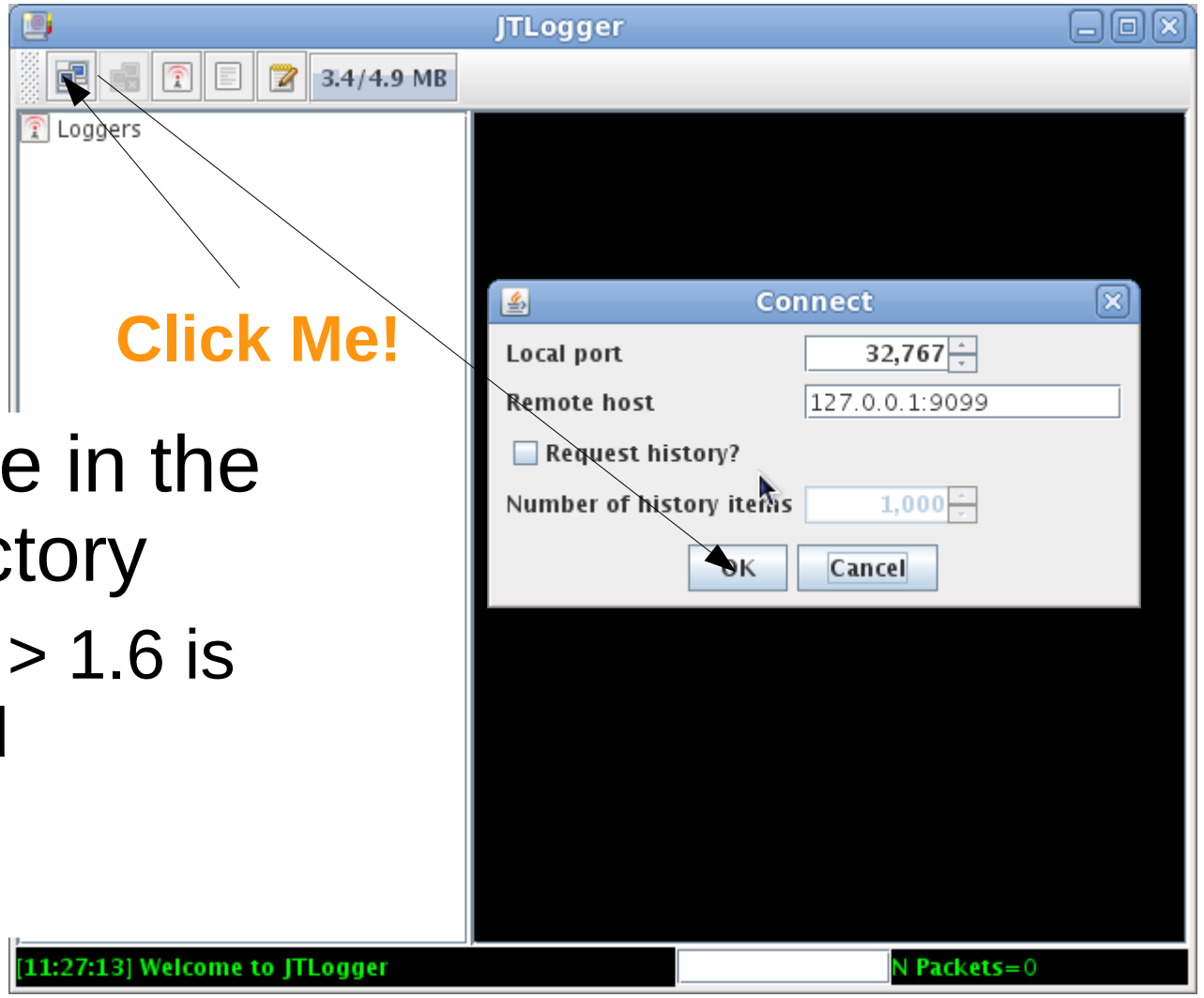
**Integer 64 bits signals**

| Decimal | Hex | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# Running the demo

- Install the executable on your computer
  - Linux users, remember to set the LD_LIBRARY_PATH to point to the location where you have installed MARTe
    - `e.g:export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.`
  - Ubuntu users might require libtinfo (ncurses)
    - `sudo ln -s /usr/lib/libncurses.so.5 /usr/lib/libtinfo.so.5`
    - `sudo ln -s /usr/lib/libtinfo.so.5 /usr/lib/libtinfo.so`
  - Antivirus software might interfere with MARTe
    - (Disable it at your own risk)

# Starting the logger




- Run the jar file in the *logger* directory
  - Sun Java > 1.6 is required

# Starting MARTe

- MARTe can be run either as a service or as a console program
  - The latter will be used in this demo
- Start MARTe by executing MARTe.ex
  - If no parameters are given MARTe will look for a MARTe.cfg file in the same directory
    - Use this option for the demo
  - Otherwise it will assume the first parameter is a path to a configuration file

# The logger is your friend...

```
[11:34:58]:localhost.localdomain:FatalError:tid=0xb78046d0 ():cid=0x0:obj=GLOBAL:InitGlobalContainer:: Failed opening file configurations/MARTe-WaterTank-OOOPSS.cfg
[11:34:58]:localhost.localdomain:Information:tid=0xb7820b70 ():cid=0x0:obj=GLOBAL:Switching log server to localhost:32767
[11:35:0]:localhost.localdomain:FatalError:tid=0xb78046d0 ():cid=0x0:obj=GLOBAL:MARTe:: MARTe Initialization has failed.
```

- Sometimes it can get too verbose
  - Use the menu on the left to filter accordingly to error (value < 1)
- Even if not demonstrated here the logger can be connected to a logger service
  - Stores logging history
  - Enables history retrieving
  - Logging broadcast across networks
  - Multi-client support

# The MARTe console

- Menu automatically created accordingly to the configuration file
  - Obsolete, but quick, interface

```
##############################################################################
##                              MARTe Menu                                  ##
##############################################################################
                                      #
A:->         State Machine Interface  #
                                      #
B:->                           MARTe  #
                                      #
                                      #
                                      #
                                      #
                                      #
                                      #
                                      #
                                      #
                                      #
                                      #
                                      #
                                      #
                                      #
##############################################################################
0: EXIT
##############################################################################
```

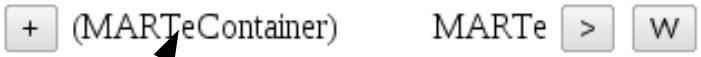
# The HTTP interface

- Point your browser to http://localhost:8084
- Click on BROWSE
  - Any objects that implement the HTTP interface can be inspected
- Open the MARTe.cfg and compare with the object tree in your browser

```
+BROWSE = {
        Title = "Http Object Browser"
        Class = HttpGCRCBrowser
        AddReference = {MARTe StateMachine OBJBROWSE
THRBROWSE CFGUpload HTTPSignalServer MatlabSignalServer}
    }
```

# Browsing objects

- Containers can be expanded to display their objects



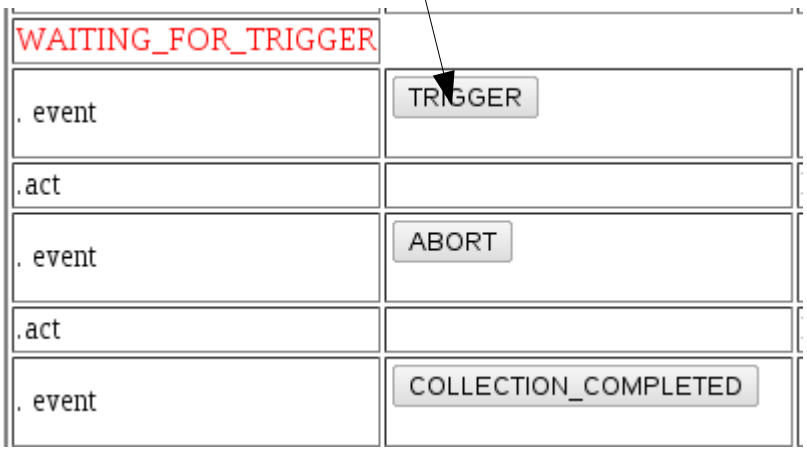

  - Click on the + symbol next to (MARTeContainer)
  - Click on the + symbol next to (RealtimeThread)
  - Two GAMs are displayed
  - Click on the > symbol next to Statistic
- Debug information can also be displayed using this interface
  - Click on THRBROWSE  and OBJBROWSE

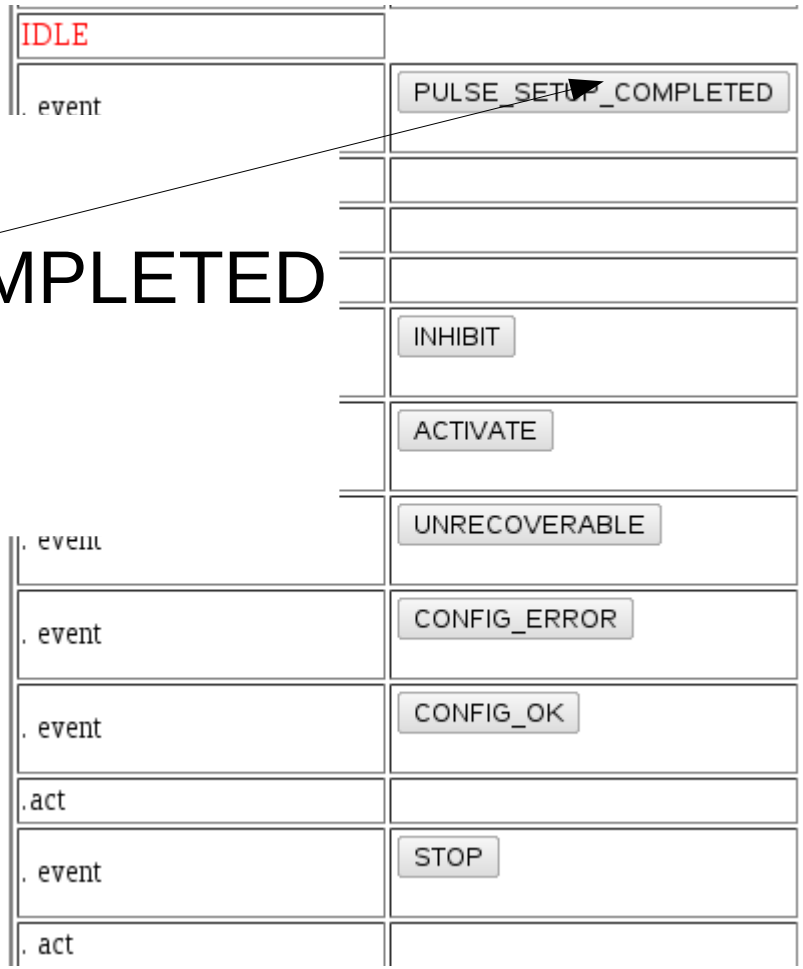
# Loading a new configuration file

- *In a fully deployed production system this should be performed using other protocols...*
- Click on the *CFGUpload*
  - Select Wait reply
  - Select the configuration file *MARTe-WaterTank.cfg*
  - Click on the REFRESH button (upper left part of the screen)
- If you list the objects inside the RealtimeThread, you should see new GAMs...

# Using the State machine

- Change to online
  - PULSE_SETUP_COMPLETED
  - TRIGGER



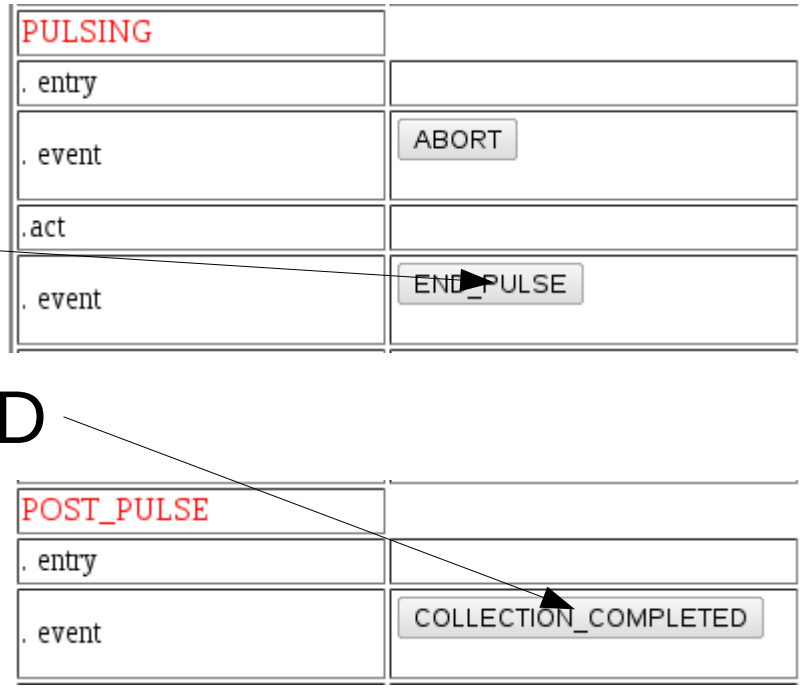



# Figures of merit

- Visit the WebStatistics page
  - http://localhost:8084/BROWSE/MARTe/Thread_1/Statistic/
  - **CycleUsecTime**: the control cycle time
  - GAM_N_**RelativeUsecTime**: execution time of GAM_N
  - GAM_N_**AbsoluteUsecTime**: elapsed time from start of cycle until end of GAM_N execution
- More information in PlottingGAM and WaterTank:
  - http://localhost:8084/BROWSE/MARTe/Thread_1/PlottingGAM/
  - http://localhost:8084/BROWSE/MARTe/Thread_1/WaterTank

# Returning to offline...

- Back to StateMachine page
- Select END_PULSE
- Select COLLECTION_COMPLETED
- New configurations are only accepted when offline
- Data collection only possible in offline



# Downloading acquired data

- HTTPSignalServer
  - Signals stored in ASCII format
- MatlabSignalServer
  - Signals store in Matlab format (can also be opened in Octave)
- Select the MatlabSignalServer
  - Select *Dump all signals to file* and *Send*

# Analysing data (with octave)

- Open Octave and move to the folder to where the signals were download
- `octave:1> load allsignals.mat`
- `octave:2> whos`
- `octave:3> plot(TIME/1e6, WATERHEIGHTREFERENCE, TIME/1e6, WATERHEIGHT)`
- `octave:4> xlim([0 5])`
- `octave:5> plot(TIME/1e6, CYCLETIME)`
- `octave:6> xlim([1 2]);ylim([1e-3 5e-3])`

# Bad configurations (1/2)

- Return to the CFGUpload
- Select the configuration file
  *configurations/MARTe-WaterTank-bug.cfg*
- And select Wait reply
- You should see
  - **MARTe replied: ERROR**
- Investigate the error in the logger

```
GLOBAL:Signal waterHeightReferences, used by PIDGAM with interface InputInterface reports error: missingSourceError.
(DDB *)0xb7600ae8:DDB::CheckAndAllocate(): One or more errors prevent DDB allocation.
0:obj=(RealTimeThread *)0x9fb0f08:RealTimeThread::CreatePerformanceMonitors4Gams: Thread_1: DDB CheckAndAllocate Failed
```

# Bad configurations (2/2)

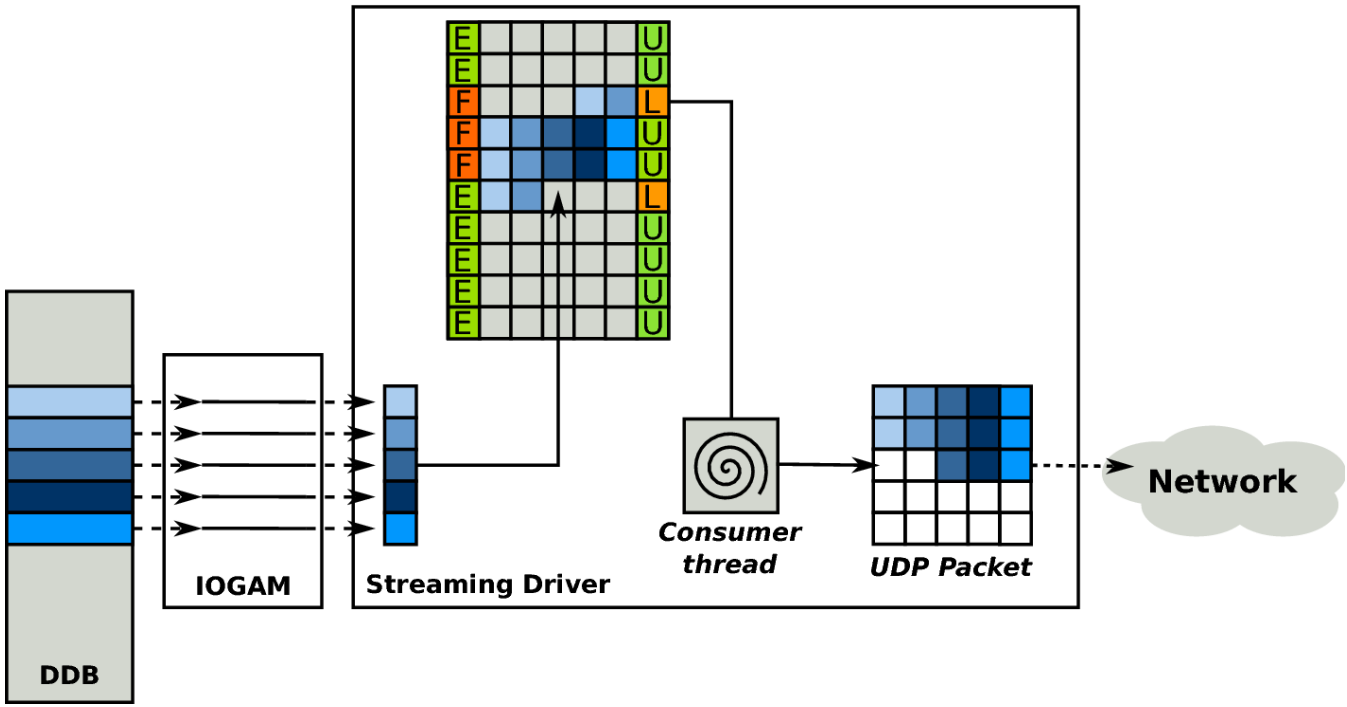
- State machine will go to ERROR
- Recover by loading the MARTe-WaterTank.cfg
- State machine will recover
- All states and error responses are configurable

| ERROR | |
|---|---|
| . event | ACTIVATE |
| . event | COLLECTION_COMPLETED |

| IDLE | |
|---|---|
| . event | PULSE_SETUP_COMPLETED |
| .act | |
| .act | |
| .act | |
| . event | INHIBIT |
| . event | ACTIVATE |
| . event | UNRECOVERABLE |

# Streaming

- Streaming UDP driver
  - Connect several RT MARTe
  - Send data to an HMI



# Main configuration parameters

```
+StreamingDrv = {
            Class = StreamingDriver
            NumberOfOutputs = 4
             NumberOfBuffers = 40
            NumberOfTransferBuffers = 1
            ReceiverUDPPort = 14500
            ReceiverUDPAddress = localhost
            CPUMask = 1
        }

+StreamingReceiver = {
            Class = StreamingDriverReceiver
            NumberOfInputs = 4
            NumberOfOutputs = 0
            NumberOfTransferBuffers = 1
            ReceiverUDPPort = 14500
            SynchronizationMethod = Synchronizing
            CPUMask = 1
        }
```

# Running the example (1/2)

- Start one MARTe with the cfg. file
  configurations/MARTe-WaterTank-Streaming-Sender.cfg
- Start a second MARTe with the cfg.
  configurations/MARTe-WaterTank-Streaming-Receiver.cfg
- Go to the sender MARTe state machine
  - http://localhost:8084/BROWSE/
  - Change the status to online
- Go to the receiver MARTe
  - http://localhost:8085/BROWSE/
  - Check the statistics page

# Running the example (2/2)

- Change the receiver state to online
- Visit the GAMs pages
  - Statistics
  - PlottingGAM
  - WaterTank

# Backup slides