Table of Contents

Logging and Configuration in

Summary of updates:

- The code generation (the one that created model.h) was updated so that the Simulator::father member variables get sets BEFORE calling init.
  This is needed in other to traverse the model hierarchy in its initialization (for example to read configuration parameters using the full hierarcy name).
- A new ConfiguraitonLogger class was created. It read configuration for the logLevel and logger each variable uses. Verbose defaults are used in case nothing is explicitly set in configuration.
- A new SamplerLogger class was created. It samples a variable logging max,min, count, timeAvg, sum every samplingPeriod (read from config)
- Variables are now logger with "." separator instead of the previous "_". This allows for longer names, as Scilab treats the "." as structure separator.

Important breaking changes

When using the defaults, the ConfigurationLogger is used. Before it used the ScilabLogger. Some differences between these loggers:

1. If using the defaults (not setting configuration values), the ConfigurationLogger creates a ScilabLogger for each variable. This means that instead of having one t variable for each model, now there will be a t variable for each variable. EJ: before you had server.t, server.count, server.bits. Now you have server.count, server.count.t, server.bits, server.bits.t.

2. Because of the change in separator (from "_" to "."), in R when reading files with H5, the "." separator is treated as a structure, which in H5 parlance is a "/" separator. You can use H5ls() to list variables stored in a file.
3.

A new ConfiguraitonLogger class was created, implementing the IPowerDEVSLogger interface. It creates the configured logger for each variable.

logLevel configuration

At the InitSignals method, it removes variables which are configured not to be logged. For this it reads the logLevel for each variable using the readLogLevel method global method. This method looks in the configuration in different scopes:

1. First, it trys with full name: EJ: coupled.atomic.variable.logLevel=1
2. If it fails, it uses the scope search for without variable name. EJ: coupled.atomic.logLevel --> atomic.logLevel --> logLevel
3. If there is no configuration of the variable, it is by default logged

Logger type configuration

For all the variables which are configured to be logged, it searches which logger is to be used. By default all the values and times of the variable are logged. Other Logger types are the SamplerLogger and the HistogramLogger (not yet implemented).
All logger types use the main backend Logger set in configuration (either Scilab or OpenTSDB). This is set with the variable_logging_backend option.

Some samplers require extra parameters to be configured. For example, the if the SamplerLogger is configured it will attempt to read the sample_period from configuration.

Example configuration

logLevel= LOG_LEVEL_NONE; // set global logging level to NONE, so that no models are logging
FelixServer1.logLevel = LOG_LEVEL_ALL; // set the FelixServer1 model to log all its variables
FelixServer1.count.logLevel = LOG_LEVEL_NONE; // Except for the FelixServer1.count variable
FelixServer1.intergen.logger = LOGGER_SAMPLER; // for the FelixServer1.intergen variable, use a sampler logger
FelixServer1.intergen.sample_period = 10; // sampling period for the FelixServer1.intergen sampler
Coupled0.FelixServer2.sent_bits.logLevel = LOG_LEVEL_PRIORITY; // FelixServer2 atomic model, which is inside the Coupled0 coupled model will log only the priority and important variables
Coupled0.FelixServer2.sent_bits.logger = LOGGER_SAMPLER; // for its sent_bits variable it will use a samplerLogger
Coupled0.FelixServer2.sent_bits.sample_period = 1; // sampling every 1s

The sampler receives values every time a variable is logged (it changes), but logs only every a fixed time period (ej: 1s). When this period elapses the sampler logs the following metrics
- **max**: maximum value received during the period
- **min**: minimum value received during the period
- **sum**: sum of values received during the period
- **avg**: average of received values (sum/nValues)
- **timeAvg**: time weighted average of the received values
I.E: timeAvg=SUM_1_N(valueN * etN) / samplingPeriod, where value1.. valueN are the received values, and et1 ..et N are the elapsed time where the valueN was observed.

For all previous metrics, the Sampler only records one value per timestamp. I.E: 2 values for the same simulated time, it only records the last one

- **evCount**: count of calls to logSignal, even it they are in the same timestamp

definition

The first 4 metrics are obvious to understand. The 'timeAvg' metric is a time weighted average of the received values. I.E: SUM_1_N(valueN * etN) / samplingPeriod, where value1.. valueN are the received values, and et1 ..et N are the elapsed time where the valueN was observed.
EJ: if the sampling at 1s and receives the following values (t, value): (0.1, 1), (0.5, 5), (0.8, 9), then the metrics would be: max=9. , min=1, count=3, sum=15, timeAvg=(1*0.4 + 5*0.3 + 9*0.2)=3.7

Using the sampler, the "normal" avg can be calculated as sum(sampler_sum) / sum(sampler_count)

It is important to note that the *timeAvg = avg.*
For example the avg in the previous example would be (1+5+9)/3 = 5.

Averages in our model

It is important to note also that the sampler only records one value per timestamp. That means that if it receives 2 values for the same simulated time, it only records the last one.

This is important because the queue publishes queueSize values everytime the queueSize changes, but it can change more than once in the same instant (ej: if the server requested next packet and the queue was empty, when the queue receives next packet it will publish queueSize=1 and in the same instant queueSize=0).

Because of this difference, ***mean(queue.size) = samplerAvg.***
For example, if the server has serviceTime=0 it will instantly request new jobs to the queue. The queue will publish queue.size of 0 and 1: (0,0); (0.1,1); (0.1,0); (0.2, 1); (0.2,0).. etc. Then mean(queue.size)=0.5. The samplerAvg=sum(sampler.sum)/sum(sampler.count) = 0

The **samplerAvg seems more accurate of reality**, and in practice they don't differ much with reasonable loads.

But in practice there is a big difference between the samplerAvg and the sampler.timeAvg. Although the sampler.timeAvg seems more accurate, it does not match with the MVA averages.
That the reason **we use the samplerAvg as metric**.

- If more than one value arrive in the same T, only sample the last one for that T

-- MatiasAlejandroBonaventura - 2016-06-17

This topic: Main > PowerDEVSLoggingAndConfiguration
Topic revision: r4 - 2016-06-21 - MatiasAlejandroBonaventura