GEC21 - OEDL Tutorial 2

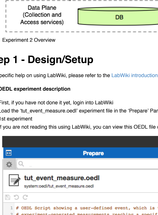
Overview

This second part shows how to design, execute, and view the results of an experiment, which dynamically reacts to measurements that are collected during its execution.

This experiment demonstrate OEDL's capability to allow user to define events which will trigger based on measurements produced by resources involved in the experiment. In other words, if one or more collected measurements reach a specific condition, then the event will trigger and some user-defined tasks will be executed.

In this experiment:

- we use only 1 resources, which will run a sine signal generator
- this signal generators is configured to collect and report the sine values
- we define a custom event which will trigger when the absolute value of the last measurement is above an arbitrary threshold of 0.99. The Experiment Controller will periodically monitor the collected measurements to check for this condition
- we also define some tasks to execute when this event is triggered. In this case, the task is to send one ICMP ping packet to a target host (using the same instrumented ping application as in the first part of this tutorial).
- we finally display both measurements from the sine generator and the ICMP ping packets, as a graph showing the sine values and a table showing ping's timestamp and RTT, respectively. Thus the table's number of rows should be equal to the graph's number of positive and negative peaks.



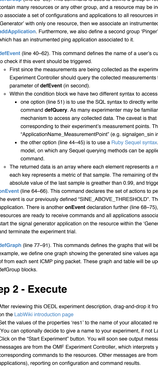
Experiment 2 Overview

Step 1 - Design/Setup

For specific help on using LabWiki, please refer to the LabWiki introduction page

The OEDL experiment description

- First, if you have not done it yet, login into LabWiki
- Load the *tut_event_measure.oedl* experiment file in the 'Prepare' Panel of LabWiki. This file contains the OEDL script for this 1st experiment
- If you are not reading this using LabWiki, you can view this OEDL file online at: http://git.io/31AM-w



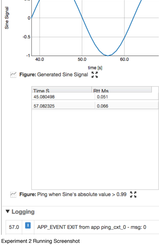
Experiment 2 OEDL Extract

Walk-through the OEDL experiment description

1. First, a reminder that all details on OEDL are available in the OEDL reference page
2. loadOEDL (line 16-17). This command includes in your OEDL experiment other external OEDL scripts. In this example, we are loading the definition of a signal generator and ping application (both instrumented with OML)
3. defProperty (line 19-20). This command defines experiment properties (aka variables), you can set the values of these properties as parameters for each experiment trials, and access them throughout the entire experiment run. In this example, we are defining 2 properties, to hold the name of the resource to use and the target for the ping application.
4. **Some internal variables** (line 21). This is a simple Ruby command that defines an array to hold the list of signal peaks throughout the experiment's execution. As opposed to the above defProperty variables, internal variables cannot be set at the start of each experiment trial (without having to change the content of the OEDL script itself)
5. defGroup (line 23–36). This command is used to define a group of resources which we will use in this experiment. A group may contain many resources or any other group, and a resource may be included in many groups. This commands may also be used to associate a set of configurations and applications to all resources in a group. In this example, we define a first group 'Generator' with only one resource, then we associate an instrumented signal generator to it. This association is done using the addApplication. Furthermore, we also define a second group 'Pinger', which contains the same resource as the first group, but which has an instrumented ping application associated to it.
6. defEvent (line 40-42). This command defines the name of a user's custom event and the block of conditions which will be used to check if this event should be triggered.
   - First since the measurements are being collected as the experiment is running, we need to specify how frequently the Experiment Controller should query the collected measurements to check for the conditions. This is set using the 'every:' parameter of **defEvent** (in seconds).
   - Within the condition block we have two different syntax to access some of the collected measurements:
     - one option (line 51) is to use the SQL syntax to directly write a query for the measurement database using the command **defQuery**. As many experimenter may be familiar with SQL, this option gives them a very flexible mechanism to access any collected data. The command is that they need to know the names of the database tables corresponding to their experiment's measurement points. This is usually of the form "ApplicationName_MeasurementPoint" (e.g. signalgen_sin in line 51.
     - the other option (line 44–45) is to use a Ruby SeqLet syntax. The method **max(ep)** returns a Measurement Point model, on which any Sequel querying methods can be applied. The resulting query is then passed to the **defQuery** command.
   - The returned data is an array where each element represents a measurement sample. Each sample is in turn a hash where each key represents a metric of that sample. The remaining of the condition block in this example (line 53–59) checks if the absolute value of the last sample is greater than 0.99, and triggers the event if so.
7. onEvent (line 64–66). This command declares the set of actions to perform when a specific event is triggered. In this example, the event is our previously defined "SINE_ABOVE_THRESHOLD". The actions to perform in this case is to select start a ping application. There is another **onEvent** declaration further (line 68–75), for the event "APP_UP_AND_INSTALLED", i.e. when all resources are ready to receive commands and all applications associated to them are installed. When this event triggers, we start the signal generator application on the resource within the 'Generator' group, then after 60 seconds we stop all applications and terminate the experiment trial.
8. defGraph (line 77–91). This commands defines the graphs that will be displayed while the experiment trial is running. In this example, we define one graph showing the generated sine values against time, and one table showing the timestamp and RTT of from each sent ICMP ping packet. These graph and table will be updated using measurements enabled in the previous defGroup blocks.

Step 2 - Execute

- After reviewing this OEDL experiment description, drag-and-drop it from the "Prepare" panel to the "Execute" panel, as described on the LabWiki introduction page
- Set the values of the properties 'rest:' to the name of your allocated resource. Similarly set the 'Slice' property to your own slice. (You can optionally decide to give a name to your experiment, if not LabWiki will assign a default unique name to it.)
- Click on the "Start Experiment" button. You will soon see output messages under the "Logging" section. Some of these messages are from the OMF Experiment Controller, which interprets your OEDL experiment description and sends corresponding commands to the resources. Other messages are from the resources themselves (either the VM nodes or the applications), reporting on configuration and command results.



Experiment 2 Execute Screenshot

- Above that "Logging" section, you should soon see the graph, which we defined in the OEDL experiment description. It is drawn dynamically as measurements are collected from the resources.

Experiment 2 Running Screenshot

Step 3 - Finish

- A message in the "Execute" panel will appear to inform you that the experiment execution has finished. At this stage, you should have the complete graph and table for this experiment in that panel, which should look as follows.

Experiment 2 Result Screenshot

- You may interact to with the graph, e.g. hover the pointer above a graph point to display the underlying data point, drag-and-drop the graph via its icon to the 'Plan' panel as described in the LabWiki introduction page
- The complete data set holding the measurements collected from this experiment is stored in an SQL database. You can retrieve a copy of that database by clicking on the 'Database Dump' button in the 'Execute' panel. The format of that copy is depends on your LabWiki's deployment configuration. It could be an *RCIOS* dump, a Zipped archive of CSV files, a SQLite3 dump or a PostgreSQL dump. By default, it is a PostgreSQL dump.

Database Dump

Help & Additional Resources

- LabWiki quick guide
- OEDL Reference Document
- GEC21 OMF and LabWiki Tutorial
- OMF6 Documentation
- OML Documentation