OMNeT++ Community Summit 2016, Brno University of Technology (FIT-BUT), Sept 15-16.

Enhancing Visualization and Animation in Simulation Models

Attila Török, Levente Mészáros, András Varga

Contents

Parts of this presentation:

1. Adding gauges, indicators and plots to INET simulations
2. How simulation visualization is organized in INET
3. Creating smooth custom animations in OMNeT++ (planned for 5.2)

Part 1:
Adding gauges, indicators and plots to INET simulations

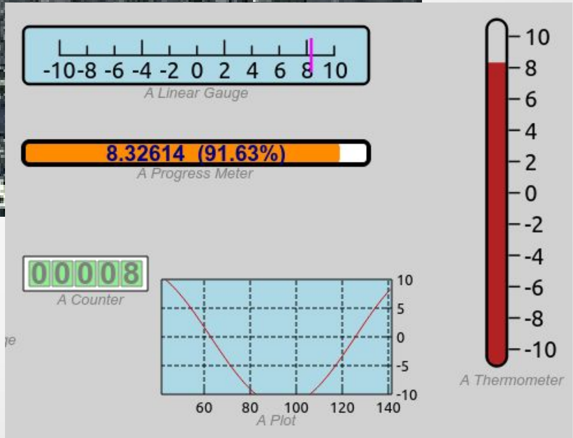
Motivation

Some use cases:

- Throughput over time
- Utilization
- Number of packet drops

Motivation:

- Quick feedback during simulation
- Demonstration purposes



Adding Instruments

- Instruments are **figures**, driven by **signals**

signal ↓

Module emits raw data as signals.

statistic ↓

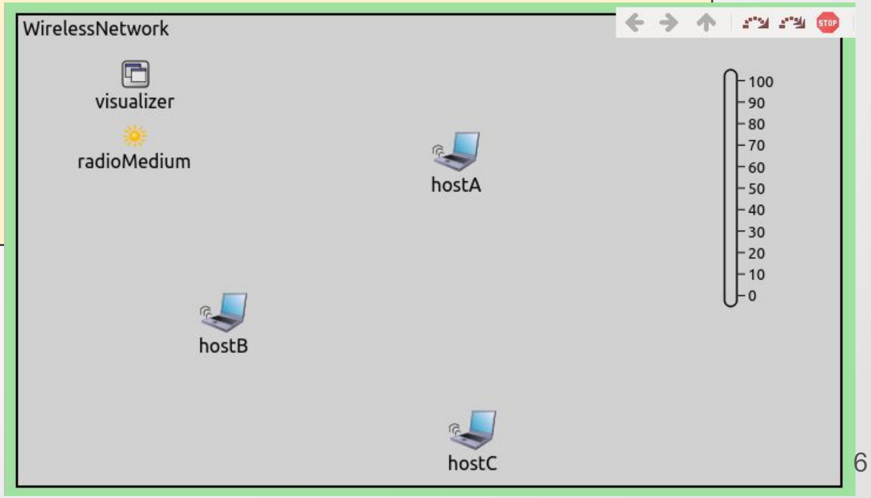
@statistic subscribes to signal, and “records” it to a figure.
- Trick: “record=figure” (uses special result recorder)
- Signals of sub, sub-sub- etc. modules may be used as source
- Result filters like sum, mean, average, arithmetic expressions, etc. are available

figure

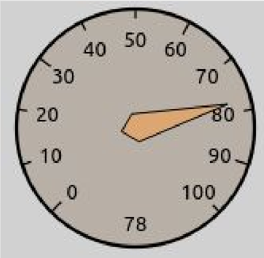
Instrument figure receives data from the “figure” result recorder, and updates on next refreshDisplay() call.
- Typically compound figures (subclass from cGroupFigure)
- Implement inet::IIndicatorFigure (contains setData())

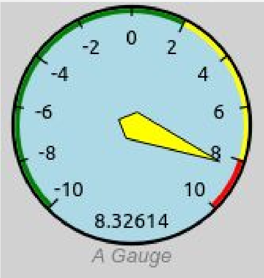
An Example

```
network WirelessNetwork {
    parameters:
        @figure[txPowerIndicator](type=thermometer; pos=700,50; size=50,300);
        @statistic[dummy](source=hostA.wlan[0].txPower; record=figure;
                                      targetFigure=txPowerIndicator);
    submodules:
        hostA: WirelessHost;
        ...
}
```

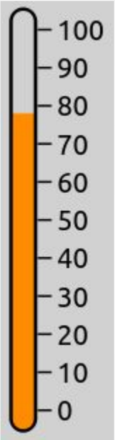


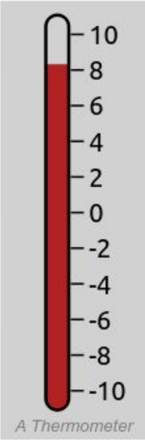
Available Figure Types 1



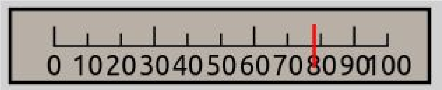


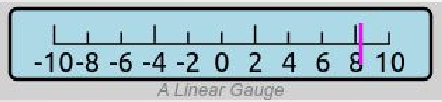
Gauge



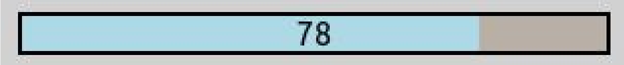


“Thermometer”



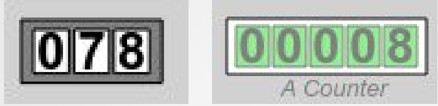


Linear Gauge

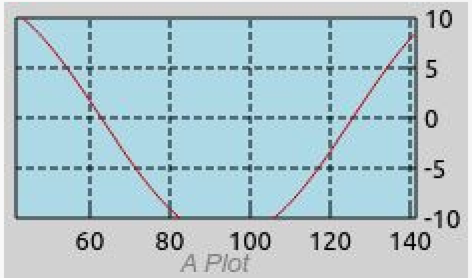


Progress Meter

Available Figure Types 2

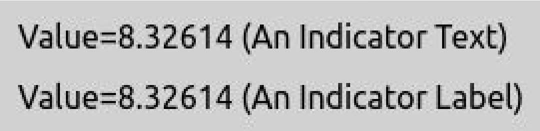


Counter



Plot

Indexed Image



Text/Label

Implementing New Instrument Figures

Some advice:

- Subclass from `cGroupFigure`
- Implement `inet::IIndicatorFigure` (mandatory, contains `setValue()`)
- Add parts as sub-figures e.g. in constructor
- Add setters/getters for properties, and `parse()` to allow `@figure`
- `setValue()` just stores value
- Update visual appearance in `refreshDisplay()`
- Copy an existing figure as template :-)

Part 2:
How simulation visualization is organized in INET

Part 3:
Creating smooth custom animations in OMNeT++

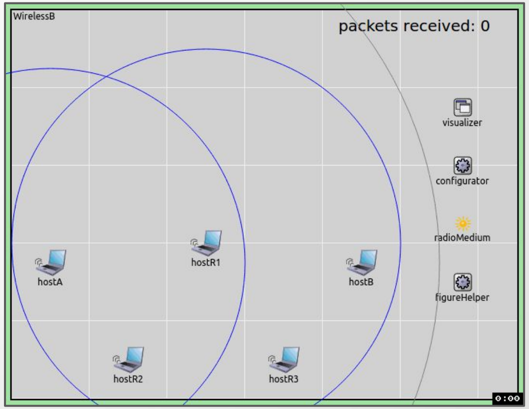
What do we want to animate?

- Node movement
- Radio transmissions
- Frames on a link
- Packet drops
- Exchange between protocol layers
- Other useful details to inform the user
  - Similar to `cEnvir::bubble()`

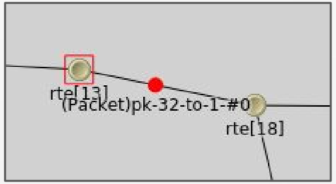
Current animation in INET

- Periodic timer ticks (artificial events) to update node positions, radio signals…

  Problems:
  - Not smooth! (tick interval = ?)
  - Different time scales
  - Overhead in Express mode
  - Noise in the logs



- Issues with built-in animations:
  - Not customizable enough
  - Cannot be reproduced from models



Key ideas

- Animation independent from simulation events
- Interpolate between events by inserting extra frames
- Call `refreshDisplay()` with intermediate SimTime values for rendering
- First approximation:
    - Fixed framerate (frames/real second)
    - Linear mapping of SimTime to real time (fixed number of frames/simsec)
    - A slider to adjust the speed

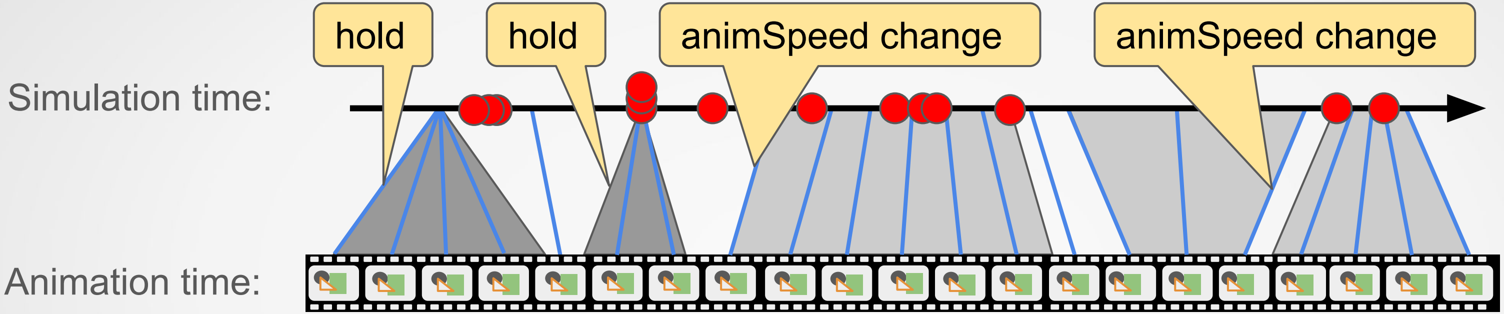
Refinements

- Problems with linear mapping:
  - Signal propagation and node movement are in different time scales
  - Animation is either boring, or skips over short duration details
- Solution:
  - Different parts of the simulation can request different animation speeds
  - Each `cCanvas` will take the minimum of all current requests as its own animation speed
- No animation speed requests:
  - Qtenv will run with a tweakable, non-linear mapping of SimTime to animation time
  - Short inter-event intervals will be inflated, and long waits shortened

Handling zero-time animations

- Some animations take zero simulation time, like
  - Sending a message over a zero-delay link
  - Methodcalls
  - Other important moments that the model wants to inform the user about
- Solution: “hold” time
  - Event processing (and the progression of SimTime) is paused
  - Animation time continues to pass
  - Using a per-cCanvas timer, so the holds in inner modules can be ignored
  - The maximum of the requested and the current (remaining) time is used

Simulation time, animation time



- Animation time can be thought of as the current play position in a movie
- What the movie looks like is directed by the mapping above
- How the movie is played back is defined by the current run mode
- Playback speed is controlled by a slider on the UI
- Adaptive rendering frame rate based on CPU utilization

Run modes

- **Step**: Animate until the next event, then stop
    - As if the movie automatically paused at the end of each cut
- **Run**: Strives to animate at a target frame rate, e.g. 10-60 FPS
    - Simply plays the movie, balancing CPU usage between animation and simulation
- **Fast**: No waiting between events, less CPU for animation, holds are ignored
    - Similar to fast-forwarding a video tape
- **Express**: Simulate as fast as possible, negligible CPU time for animation
    - Just quickly skipping through the movie

API

- cCanvas:
  - `void` **`setAnimationSpeed`**`(double` **`animationSpeed`**`, const cObject *`**`source`**`);`
  - `void` **`holdSimulationFor`**`(double` **`animationTimeDelta`**`);`
- cEnvir:
  - `double` **`getAnimationTime`**`();`
  - `double` **`getAnimationSpeed`**`();`
  - `double` **`getRemainingAnimationHoldTime`**`();`

Cooperation with schedulers

- Should still support custom event scheduling
  - Think of `cRealtimeScheduler` or hardware in the loop
- Waiting has to be delegated to `cScheduler` to make this possible
  - So it can resume execution if an event comes in that has to be processed immediately
- For the UI to be responsive, `cEnvir::idle()` has to be called periodically
- `cScheduler` can also have control over the current SimTime
- New `cScheduler` methods:
  - `bool` **`wait`**`(int msecs), bool` **`governsSimTime`**`(), SimTime` **`simTimeNow`**`()`
- Default implementations are in place for all of them

One way to implement animations

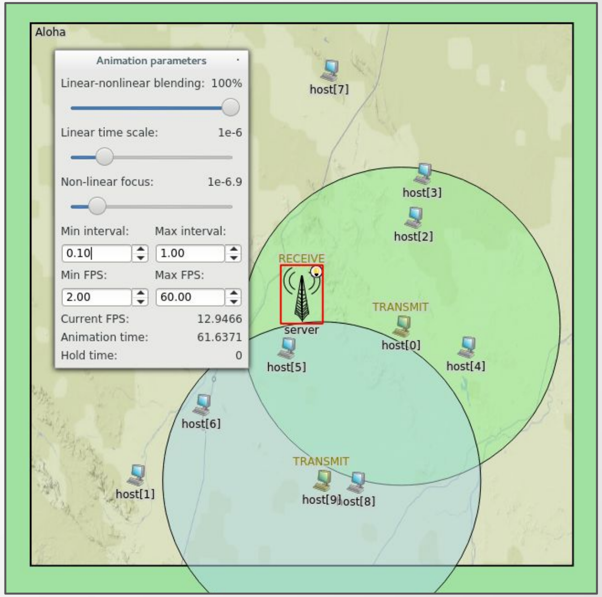
- Interesting events are recording their animations into a “screenplay”
- At an appropriate time the visualizer will call a hold
- Then the recorded sequence can be played back
    - Rendering is done in `refreshDisplay()`
    - Progression using `getAnimationTime()`
- This is similar to how the embedded animations work

Deterministic video recording

- Support for built-in video recording is planned
- Frames are rendered on well defined points in time
- Advantages compared to simple screen capturing:
  - Eliminates the occasional jerks caused by varying system load
  - No need for additional software and configuration
  - Simple “push button” usage
  - Output is easily reproducible and can be fine-tuned
  - The simulation/animation doesn’t have to run in real time with high framerate

Status

- Experimental implementation available in OMNeT++ 5.1 Tech Preview, release planned for version 5.2
- You can try it now on the Aloha example:
  - Hosts have fixed position, computed `radioDelay`
  - A parameter to enable/disable `setAnimationSpeed`
  - Can optionally hold time upon collision
  - Illustrates the protocol much better than before
  - Collisions and slotting are clearly visible
- Porting of INET visualizers will follow



Thank You