What's New in OMNeT++

András Varga
Opensim Ltd
andras@omnetpp.org

2nd OMNeT++ Community Summit
September 4-5, 2015
IBM Research Labs, Zurich

PART 1

API CHANGES

New Logging Mechanism

More featureful API:

- log levels: fatal, error, warning, info, detail, debug, trace
  - *fatal, error, warning:* should refer to events in the *simulated system* (not to technical errors in the simulation program)
  - *info, detail*: should refer to the domain (i.e. understood by anyone who knows the protocol the model implements)
  - *debug, trace*: specific to the implementation (i.e. understanding requires familiarity with the source code)
- category support
  - category identifies a topic; example categories: “retx”, “frag”, “mgmt”

```
EV_INFO << “Received BA acks all outstanding frames\n”;  // same as EV << ...
EV_TRACE << “Returning from processBlockAck()\n”;

EV_WARNING << “Timeout, retransmitting frame\n”;
EV_ERROR << “Protocol violation: received ACK when CTS was expected\n”;

EV_DETAIL_C(“frag”) << “Fragment completes frame, sending up\n”;
```

New Logging Mechanism

Configurability:

- runtime loglevel threshold, compile-time loglevel threshold
- filtering by module and category

Implementation:

- tons of information is passed to the logging code with each line:
  - file/line, loglevel/category, event number, simulation time, msg name and type, module name and type, object name and type, etc.
- configurable log prefix (40+ "*%x*" format directives!)

Random Variate Generation

Introduced cRandom – random variable stream as object

- encapsulates an RNG (cRNG*) and parameters of the distribution
- numbers can be extracted with the draw() method
- subclasses: cUniform, cExponential, cNormal, etc.; also cStatistic!

Related change on random variate generation functions (normal(), etc.):

```
double normal(double mean, double stddev, int rngIndex=0);
                          ↓ ↓ ↓
double normal(cRNG *rng, double mean, double stddev);
```

- Motivation: break the functions' dependence on the context module
- Functions with the old signatures added to cComponent as methods, so most models won't notice the change

Event Class

Introduced `cEvent` as a base class of `cMessage`:

- `cEvent` allows scheduling of arbitrary code for a simulation time that runs independent of modules.

```
class cEvent {
  public:
    virtual void execute() = 0;
    ...
};
```

Motivation:

- allow implementing simulation time limit with an "end-simulation" event
- encapsulate foreign events (e.g. SystemC events) for seemless integration with the simulation event loop
- NOT intended for use in simulation models

Scheduler Changes

`cScheduler` API has changed:

- Partly in relation to the `cEvent` change
- Removed:
  - `virtual cMessage *getNextEvent() = 0;  // leaves event in FES`
- Added:
  - `virtual cEvent *takeNextEvent() = 0;  // removes event from FES`
  - `virtual cEvent *guessNextEvent() = 0;  // only for UI purposes`

Replaceable FES

FES (Future Event Set):

Stores the events that are scheduled to occur but have not been processed yet. send() variants and scheduleAt() add events to the FES.

Abstract base class: `cFutureEventSet`

Default implementation: `cEventHeap` (`ex-cMessageHeap`)

- implements binary heap

Configuration option:

- `futureeventset-class=<classname>`

Motivation:

- Alternative data structures may be more efficient than heap for specific workloads
- Examples: skiplist, various balanced trees, calendar queue

Simulation Lifecycle Listeners

Added simulation lifecycle listeners:

- Called back before and after network setup, on network initialization, before and after network finalization, etc.
- Motivation: allow more flexibility when writing initialization and shutdown code for schedulers, result file managers and other extensions

```
class cISimulationLifecycleListener {
    virtual void lifecycleEvent(SimulationLifecycleEventType event, cObject *details) = 0;
};
```

Events:

(LF_)ON_STARTUP, PRE_NETWORK_SETUP, POST_NETWORK_SETUP, PRE_NETWORK_INITIALIZE, POST_NETWORK_INITIALIZE, ON_SIMULATION_START, ON_SIMULATION_PAUSE, ON_SIMULATION_RESUME, ON_SIMULATION_SUCCESS, ON_SIMULATION_ERROR, PRE_NETWORK_FINISH, POST_NETWORK_FINISH, ON_RUN_END, PRE_NETWORK_DELETE, POST_NETWORK_DELETE, ON_SHUTDOWN

Fingerprint Changes

Simulation fingerprint:

A hash computed from events of a simulation run. For regression testing, one compares the fingerprint computed from a simulation run to a pre-recorded fingerprint.

Introduced in OMNeT++ 4.0, fingerprints have since proven to be extremely useful. We use them *daily* during INET development!

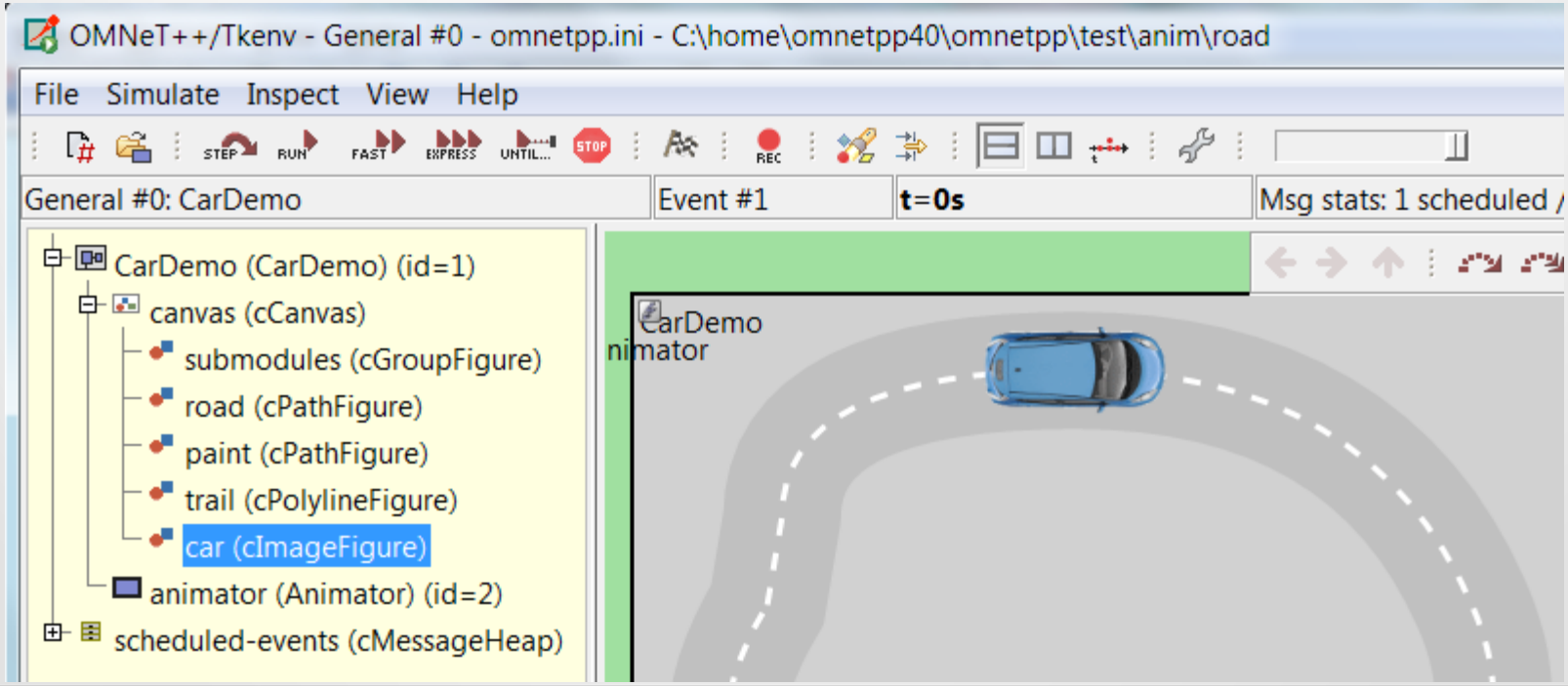
Changes to make fingerprints even more useful:

- use module path strings instead of module IDs (increases robustness)
- make it configurable what is taken as input for hash (message class names, packet lengths, control info class names, etc.)
- backward compatibility (4.x) mode also exists

Canvas API

Allows adding extra graphical elements to the usual display string based module graphics.

See last year's presentation



API Cleanup

API cleanup

- `omnetpp` namespace (recommended: `using namespace omnetpp;`)
- iterator API change: use `operator*` instead of `operator()` for dereference
- removed deprecated functions, classes and other API elements
- removed 3.x compatibility features (`WITH_DOUBLE_SIMTIME`, `WITHOUT_CPACKET`)
- images under `images/old/` are no longer accessible without the `old/` prefix
- `ev` and `simulation` macros have been removed

Codebase Cleanup

Cleanup and modernization of the codebase

- Modernizing: use `nullptr` and `override`; use `<cstdio>` instead of `<stdio.h>`, etc.
- Per-library namespaces (i.e. `omnetpp::nedxml`); fully qualified header guards; qualified includes (i.e. `#include "common/stringutil.h“` instead of `#include "stringutil.h”`)
- Include folder: contains `<omnetpp.h>`, all other files are in the `omnetpp/` subfolder
- Code style: renamed many identifiers (local variables, arguments, private data members, etc.) to have consistent, camelCase names

C++11:

OMNeT++ 5.0 will compile with, but at least the base parts will ***not*** require C++11, C++14 or newer C++ specs.

PART 2

AND MORE...

Demo

An example simulation

Tkenv is about to retire...

Qtenv:

A shiny new OMNeT++ graphical runtime based on the Qt toolkit

Release plan:

- OMNeT++ 5.0:
  - Qtenv will be included for testing and feedback, Tkenv still being the default UI
  - It will be roughly equivalent to Tkenv, in terms of UI and features
- OMNeT++ 5.1 and later:
  - Qtenv will become the default, but Tkenv will continue to be included as long as feasible or necessary
  - Features will be added gradually, e.g. improved packet flow animation

Demo

Glider demo...

3D Visualization Has Arrived

3D visualization is based on OpenSceneGraph (OSG), openscenegraph.org

"OpenSceneGraph is an open source high performance 3D graphics toolkit, used by application developers in fields such as visual simulation, games, virtual reality, scientific visualization and modeling. Written entirely in Standard C++ and OpenGL, it runs on all Windows platforms, OS X, GNU/Linux, IRIX, Solaris, HP-UX, AIX and FreeBSD operating systems. OpenSceneGraph is now well established as the world leading scene graph technology, used widely in the vis-sim, space, scientific, oil-gas, games and virtual reality industries."

Demo

Office demo...

Earth, Terrain, City Visualization

Provided by **osgEarth**, a geospatial SDK and terrain engine built on top of OpenSceneGraph

"Just create a simple XML file, point it at your map data, and go!"

- Able to use various street map providers, satellite imaging providers, altitude data sources, both online and offline
- Data from online sources may be exported into a file suitable for offline use
- Scene may be annotated with various types of graphical objects
- Includes conversion between various geographical coordinate systems

Scenarios

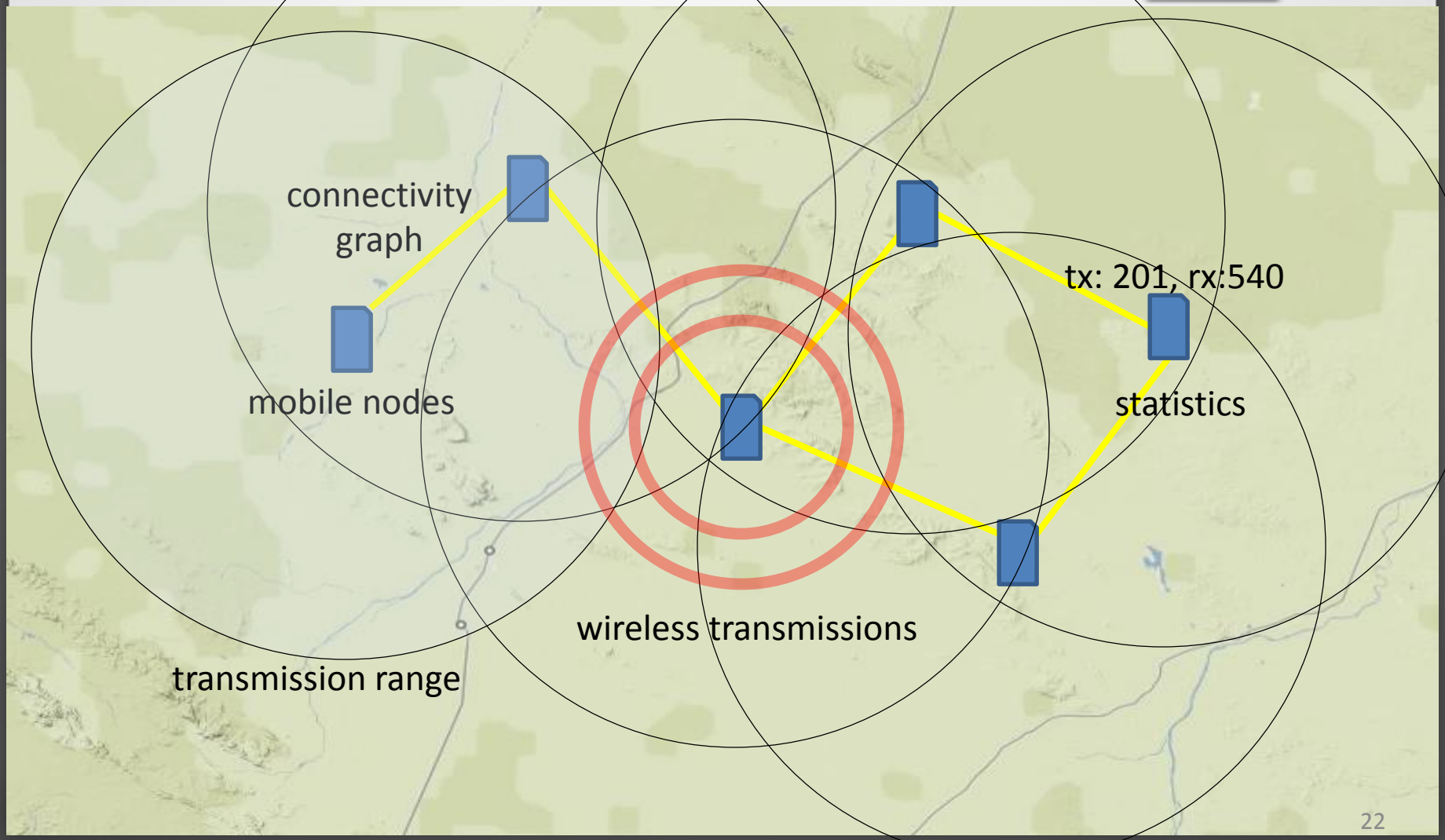
Terrains

Urban environments

Indoor environments

Satellites

Visualization



Demo

More 3D demos...

Why Cows?

The Boston Cow Parade

and the OSG Cow:

The OMNeT++ API

OK! How do I...?

Basically, use the OpenSceneGraph API.

- You assemble an OSG scene graph in the model, and give it to OMNeT++ for display. The scene graph can be updated at runtime, and changes will be reflected in the display.
- OMNeT++ class: `cOsgCanvas`
  - wraps a scene graph, plus hints such as default camera position
  - every module has a built-in `cOsgCanvas`, created on demand
  - additional `cOsgCanvas` instances may be created
- The OSG viewer is part of the OMNeT++ user interface, Qtenv – it is not directly accessible from models
- Model code should surround OSG-specific code with `#ifdef HAVE_OSG`

Example Code

"Glider" demo:

```
#include <osgDB/ReadFile>
#include <omnetpp.h>
...
void DemoModule::initialize() {
    osg::Node *scene = osgDB::readNodeFile("glider.osgb");
    cOsgCanvas *osgCanvas = getParentModule()->getOsgCanvas();
    osgCanvas->setScene(scene); // the scene graph
    osgCanvas->setClearColor(cOsgCanvas::Color(0,0,64)); // hint
}
```

"Boston" demo:

- substitute "boston.earth" for "glider.osgb"
- add the following line:

```
osgCanvas->setViewerStyle(cOsgCanvas::STYLE_EARTH);
```

Everything else is achieved by manipulating the scene graph via the OSG API!

Thank you for your attention!

Questions?