Simulations for Eternity

Creating and Sharing Reproducible Simulations Using the NIX Package Manager

Motivation

- **Installing models or even the base OMNeT++ simulator requires following manual instructions. Ideally it should be fully automated.**
- Simulations are primarily employed in research where the reproducibility of the results is extremely important. Sadly, current papers rarely provide a mechanism for this.
- Building from source is often confusing for beginners (especially if they don't intend to create their own models).
- Dependencies are not always readily available (Qt, OpenSceneGraph etc.)
- Just running a simulation requires a lot of wasted resources (Downloading IDE, compilers, sources, documentation etc.) while ideally it would need only a few megabytes of binary download.

Use Cases

- **Installing simulation models and OMNeT++ without manual installation of (distro specific) dependencies**
- **The ability to easily reproduce a paper's results for 3rd party verification**
- Trying out simulation models with a very low effort (demo, teaching)
- Deploying a model on a cluster or in the cloud
- Switching between various versions and variants during development
- Self-contained binary distribution of simulations (no compiler needed)

How to Install NIX? - Try it Now

Create a user owned NIX root directory
(this is optional: if you don't want to give root password to the installer):

```
$ sudo mkdir /nix && sudo chown $USER /nix
```

Install NIX in single user mode:

```
$ sh <(curl -L https://nixos.org/nix/install) --no-daemon
```

In your editor open `~/.config/nix/nix.conf` and add the following:

```
experimental-features = nix-command flakes
```

Or simply execute:

```
$ echo "experimental-features = nix-command flakes" >>~/.config/nix/nix.cfg
```

And then restart your shell and test:

```
$ nix –version
```

Currently: Source Distribution

Pro:

- Multiple versions of OMNeT++ and/or models can co-exist on the same machine
- Relatively independent from the base OS and toolchain versions (except when it is not)
- The running code is easy to modify (once you got it compiled first)
- Familiar for the current users

Cons:

- **Manual dependency handling for the models** (install guides explaining how to get, and set up the dependencies)
- Relatively complex for both the developers and the end users
- On Linux we use external dependencies (from native package managers), but those dependencies are unpredictable
- On Windows and macOS we bundle the dependencies, but because of this, they become very inflexible. It's hard to update or extends them if needed.

Solution 1: Use Native Package Managers

Pro:

- Easy for the beginner user
- Low resource usage (it is a pre-compiled binary distribution)

Cons:

- **Having different versions of OMNeT++ on the same computer is complicated / impossible**
- **Code is almost unmodifiable by the end user once it is packaged**
- Multiple package managers exist (Debian vs. Rpm. vs. HomeBrew vs. MacPorts. How about different distros? What about macOS or Windows?)
- Who will maintain the different packages / repos? Will model developers package their models, too?
- Very dependent on the base OS (you need new builds for each distro release).
- Low reproducibility (how do you plan to run a simulation in 10 years when the base OS is EOLed?)

Solution 2: Use Virtual Machine Images

Pro:

- Great reproducibility (as long as you have the original image)
- Cross platform (works exactly the same way on each host operating system)

Cons:

- **A VM comes with its own desktop environment and tools. Hard to integrate into your own development workflow. It's ok for demos but not for development.**
- **Hard to update base OS and tools. You have to rebuild the whole VM image from scratch usually by hand.**
- Extremely heavy distribution costs (even the smallest dependency change requires a full OS image which can be several gigabytes)
- Moderately hard to install for the end user
- No standard way to distribute OS images (i.e. no easily accessible repo)

Solution 3: Use Docker Images

Pro:

- Shares base layer images (less resource intensive than independent VM images), but an image can have only a single base, so it's hard to deal with project with multiple dependencies (e.g. omnet, inet, simu5G)
- Very good reproducibility
- Docker Hub is the defacto distribution repository (it is easy to publish and download docker images)
- Building of docker images can be automated on cloud build platforms
- Cross platform

Cons:

- **Base layer is hard to update (full image rebuild is required)**
- **Hard to use your own tools and models because a docker image it totally sealed.**
- **Primarily a binary format. If binary image is not available, the user must build it which requires special knowledge.**
- Hard to work with GUI tools (docker was not meant for GUI programs)
- Hard to update dependencies once you have more than one. What happens to a model that depends on OMNeT++, INET, Simu5G and Sumo? Publish a separate image for all combinations of these?

Solution 4: Use NIX Package Manager

Pro:

- Perfect reproducibility
- Source distribution, but with a transparent binary cache (so common dependencies are directly downloaded in binary form instead of a full rebuild)
- Any dependency is easily updatable and dependent components are only built on demand.
- Multiple OMNeT++ and model versions can co-exist peacefully
- Fine grained dependency management for saving resources (e.g. do not download the IDE if it is not needed for running the simulation)
- Package definitions can be provided in the model's source tree (e.g. a github repository), so no extra steps are needed by the model developers to distribute their packaged models
- Works with GUI apps (OpenScreenGraph is a bit complicated, but works too)
- Isolation can be fine tuned so you can use your own tools and programs that are already installed.
- The models (source or binary) are easily accessible on the native filesystem

Con:

- Works natively only on Linux and macOS (but you can use WSLg on Windows 11)

What is the NIX Package Manager?

- Allows reproducible builds and deployments
- Runs on macOS and Linux (and on WSL, too).
- With a simple package ‘flake’ it can build and/or install a package and all of its dependencies
- A package depends only on the formally defined dependencies and the source tarball (even compilers and tools are considered as a build time dependency)
- Primarily a source based distribution mechanism with transparent binary caching

When a package is requested by the user:

1. Checks if it’s already present in the local /nix/store. If yes, it maps that into the user environment.
2. If not, it checks whether a binary version exists on a global cache server. If yes, it downloads that into /nix/store
3. If there is no binary cache, it downloads the sources then builds and installs the package into /nix/store
4. It does this recursively to all of the package dependencies

How to Add NIX Support to your Model?

- A `flake.nix` file must be added to the root of the source repository.
- It specifies a function which receives inputs (other packages + build parameters) and provides an output (your package).
- It does NOT depend on anything else so as long as the same inputs are provided to the function, the same output will be generated.
- This behavior allows the caching of function outputs (i.e. packages)

NIX Flake Example

```
{
  description = "A flake for building Hello World";
  inputs.nixpkgs.url = github:NixOS/nixpkgs/nixos-22.05;
  outputs = { self, nixpkgs }: {
    defaultPackage.x86_64-linux
      # Notice the reference to nixpkgs here
      with import nixpkgs { system = "x86_64-linux"; }
      stdenv.mkDerivation
        name = "hello";
        src = self;
        buildPhase = "gcc -o hello ./hello.c";
        installPhase = "mkdir -p $out/bin; install -t $out/bin hello";
      };
  };
}
```

Installing OMNeT++ using NIX

The following command will open a shell with the specified version of OMNeT++ installed.

```
$ nix develop -i github:omnetpp/omnetpp/flake/omnetpp-6.0.x
```

All command line tools are available in this shell. We can generate a makefile with `opp_makemake` and then build and run the model. Even Qtenv works…

NOTE: The IDE is not yet available in this prompt.

Developing Models

NIX packaging support can be added easily to the OMNeT++ samples and they can be used as a starter template (currently the TicToc sample can be used as a template):

```
$ nix flake new mytictoc -t github:omnetpp/omnetpp/flake/omnetpp-6.0.x#templates.tictoc
```

The above command will create a directory named ‘mytictoc’ and initialize it with the content of the TicToc sample. Additionally a **`flake.nix`** file is provided so your project is ready to be used in the NIX ecosystem as soon as you push it to a git repository.

Running Models with a Single Command

- Once you have a flake file in a publicly accessible GIT repository, other projects can refer to it with a simple URL.
- You can also pre-define runnable scenarios that can be used for demo purposes or you can provide the URL in your paper to allow other researchers to run your model exactly how you ran it.

```
$ nix run github:omnetpp/omnetpp/flake/omnetpp-6.0.x?dir=samples/tictoc
```

The above command will launch the TicToc example in Qtenv.

Model Dependencies

- Models can set their own dependencies in their own flake file (they can either specify a fixed version number, or a running release number, like 6.0.x
- The flake files ensure that all dependencies are built with the same tools and OMNeT++ version
- Dependent models can redefine any of their dependencies as they see to fit (obviously, this is not guaranteed to work, but it is easily testable)
- **`nix flake metadata`** can display all the dependencies with exact version information:

```
Resolved URL:  git+file:///home/rhornig/omnetpp-dev?dir=samples%2ftictoc
Locked URL:    git+file:///home/rhornig/omnetpp-dev?dir=samples%2ftictoc&ref=refs%2fheads%2fflake%2fomnetpp-6.0.x&rev=254327c305091452fc911e1f61e8292639474203
Description:   TicToc Tutorial for OMNeT++
Path:          /nix/store/pk930y23jn2fnbd7gwdgbhkbr7jffms3-source
Revision:      254327c305091452fc911e1f61e8292639474203
Revisions:     26275
Last modified: 2022-11-01 17:40:58
Inputs:
├───flake-utils follows input 'omnetpp/flake-utils'
└───omnetpp: git+file:///home/rhornig/omnetpp-dev?ref=refs%2fheads%2fflake%2fomnetpp-6.0.x&rev=56ae6c473ed40b8dc5292d0f3528b2c2646b6b84
    ├───flake-utils: github:numtide/flake-utils/c0e246b9b83f637f4681389ecabcb2681b4f3af0
    └───nixpkgs: github:NixOS/nixpkgs/1c6eb4876f71e8903ae9f73e6adf45fdbebc0292
```

Future Enhancements: Optimizing the NIX Packages

- Packages can be split into multiple “outputs”
- Files in “bin” output are used only to run the simulation (i.e. files in `bin`, `lib` and `python` folder)
- Files in “dev” output allow development (header files, build tools + qtenv support)
- Debug binaries can be separated into their own package
- The IDE can be placed in a separate package

Splitting the package into multiple outputs allows omnet models to have only a minimal dependency, i.e. running an INET model on command line would not even need Qt5 or the OMNeT++ headers.

If the development environment (IDE) is needed for a model, that can be installed on an as-needed basis.