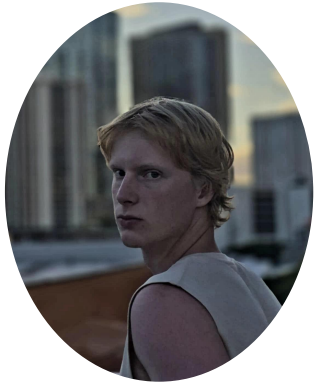
Enabling Automatic and Variation-Capable Reproducibility in Discrete-Event Simulators



Domas Davidavicius

@Large Research
@VU Amsterdam

Domas Davidavicius

Reproducibility: importance and issues

Reproducibility:

1. Allows to verify claims by reliably rerunning

the same experiments and obtaining consistent results.

2. Allows others to build on top of previous work with variation.

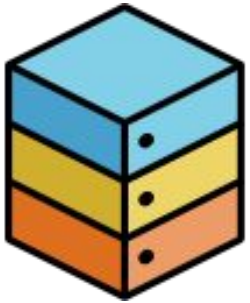
However:

1. Reproducibility requires a lot of manual work.
2. Often gets overlooked.

Simulation

Simulators, like OpenDC, allows for cost effective research on data centers.

However, as data centers grow, simulators increase in complexity, further complicating the reproducibility.



Research Question and Contributions

Main Research Question:
How to enable automatic, customizable, variation-capable reproducibility for DES?

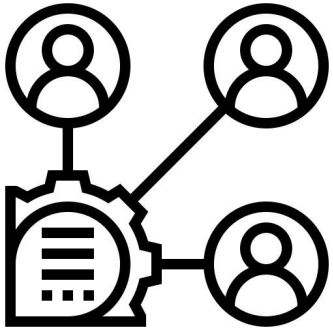
Contributions:

- CC1: structured requirement list for reproducibility in DES
- CC2: design of Reproducibility Capsule System
- TC1: prototype of Reproducibility Capsule System
- TC2: evaluation of Reproducibility Capsule System using real-world experiments

For whom it is important?

Creators:
- Novice
- Expert

Consumers:
- Verifiers
- Extenders



RQ1: User requirements

We conduct an online survey of novice and expert users in Discrete-Event simulators.

Main requirements from the survey:

- Automated and customizable experiment generation
- Exportable, self-contained capsules
- Integrated instructions and documentation generation

RQ2: Design (1)

We propose making a tool that would assist users with creating, verifying and extending reproducibility capsules.

Old Approach

Manual and Error-Prone

Unstructured and Time-Consuming

Hard to Reuse or Extend

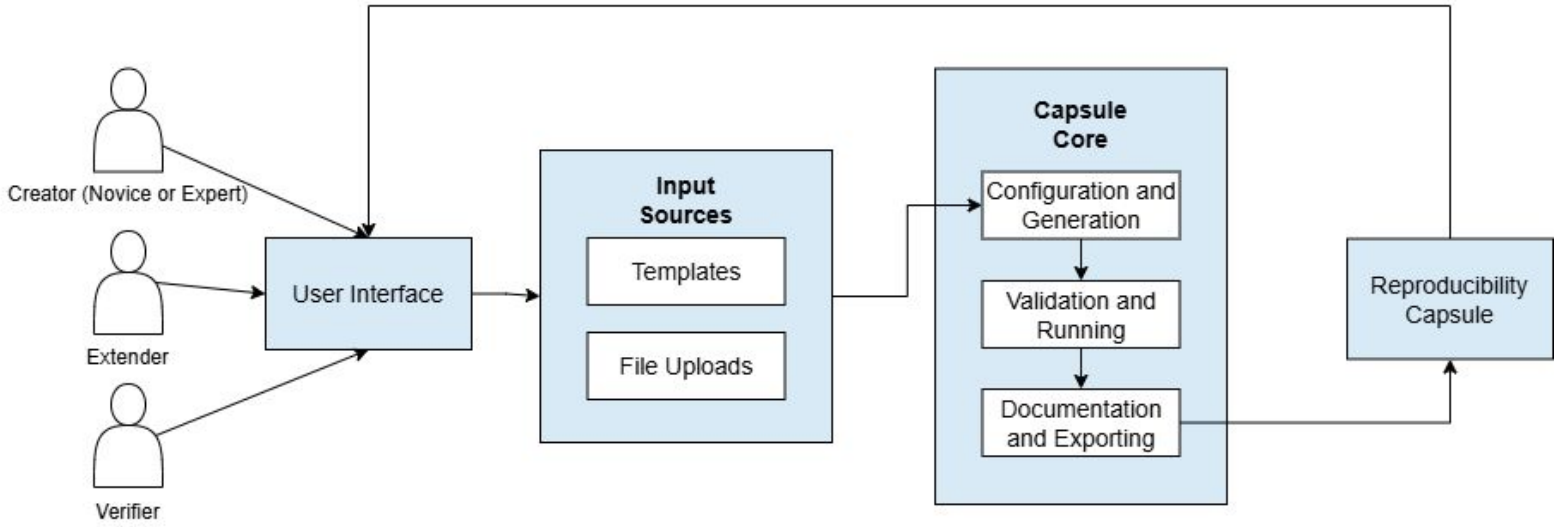
New Approach

Guided and Template-Based

Automated Workflow

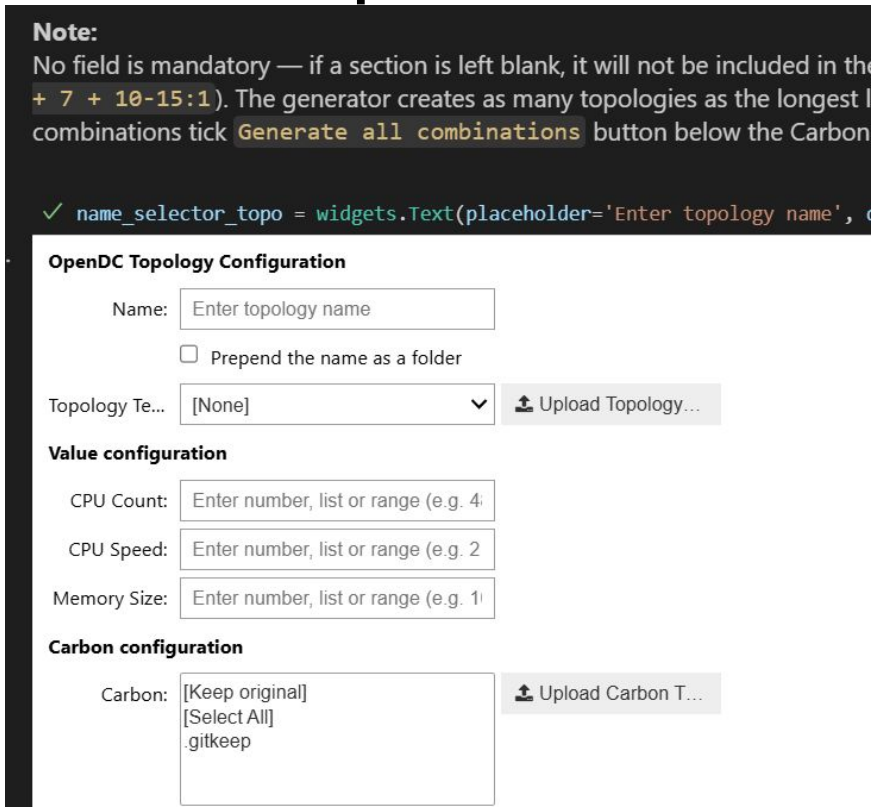
Reproducible and Extensible

RQ2: Design (2)



Overview of high-level design

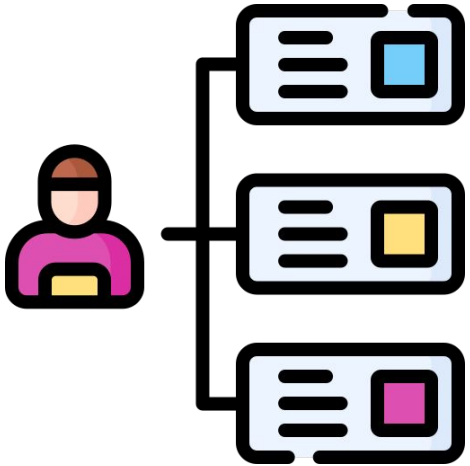
RQ3: Implementation



RQ4: Evaluation (1)

Use cases:

1. Running experiments and exporting a reproducibility capsule.
2. Reproducing a published experiment.
3. Extending an existing experiment.



RQ4: Evaluation (2)

Running experiments and exporting a reproducibility capsule:

Typical Workflow:	With Capsule:
1. Manually create the folder and file structure	1. Select from templates, generate or upload input files via the UI
2. Define and prepare all input files from scratch	2. Automatically generate multiple experiment variants
3. Manually validate input correctness	3. Run all variants with a single command with automatic validation
4. Manually run the experiments	4. Generate documentation automatically
5. Manually write documentation	5. Export the entire capsule with a couple of clicks
6. Manually organize and export files	

Conclusion and Limitations

We successfully designed and implemented a Reproducibility Capsule System for OpenDC, meeting key requirements by enabling the automated and variation-capable generation, execution, and export of experiment variants.

Limitations:

1. Impossible to automate everything without limiting flexibility.
2. Not all designed features were implemented due to time constraints.

Questions?

Table 5.3: Design-implementation coverage

Component	Implementation Status
UI	Implemented
Running Script	Implemented
Input Sources	Implemented
Generators	Implemented
Validator	Partially Implemented (syntathical checks missing)
Experiment Runner	Implemented
Simulator	Implemented
Capsule Exporter	Implemented
Version Controller	Not Implemented (manual version tracking)
Documentation Generator	Partially Implemented (AD generation missing)
Visualizer	Not Implemented

```
main.ipynb
|-- carbon_traces/
|-- experiments/
|-- failure_traces/
|-- OpenDCExperimentRunner/
|-- output/
|-- src/
|-- templates/
|   |-- carbon_traces/
|   |-- experiments/
|   |-- failure_traces/
|   |-- topologies/
|   |-- workload_traces/
|-- topologies/
|-- workload_traces/
```

Figure 5.3: Folder structure of the reproducibility capsule tool implementation.

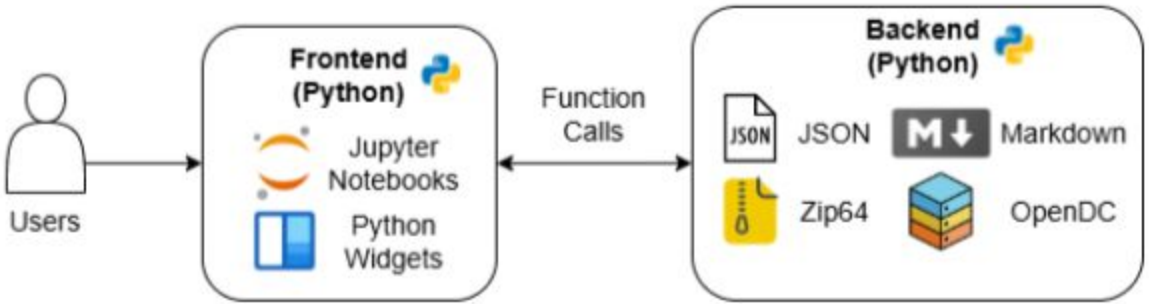
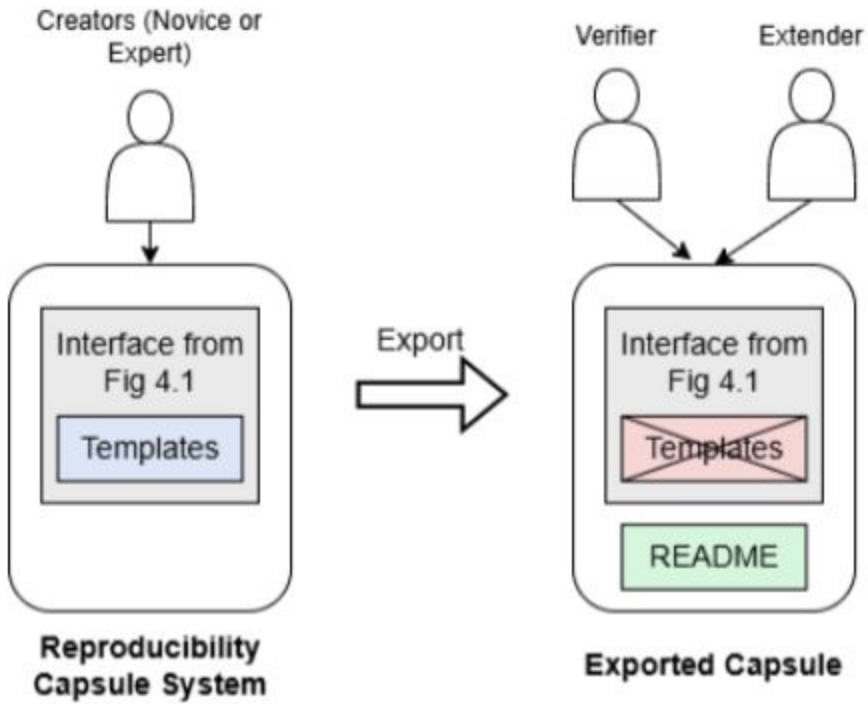


Figure 5.1: Overview of system architecture of the prototype implementation.

Table 4.5: Overview of reproducibility challenges and capsule-based solutions

Challenge	Capsule-Based Solution
Manual and error-prone experiment setup	Structured UI with templates and automatic file generation
Missing or inconsistent documentation	Auto-generated README with metadata and instructions
Unclear or inconsistent experiment structure	Predefined folder layout and consistent naming conventions
Incompatible or mismatched environments	Version tracking of simulator and library dependencies
Manual and error-prone reproducibility verification	One-click rerun with auto-validation and comparison
Need for manually writing scripts for batch generation	Built-in experiment generator with list/range support
High entry barrier for novices	Templates, guided input, and embedded interface instructions
Time-consuming reuse or extension of existing work	Structured and automatic customization of original setup with auto-export
Lack of standardized result inspection	Built-in visualizer for analysis and figure generation


Creators (Novice or Expert)
Interface from Fig 4.1
Templates
Reproducibility Capsule System
Export
Verifier
Extender
Interface from Fig 4.1
Templates
README
Exported Capsule

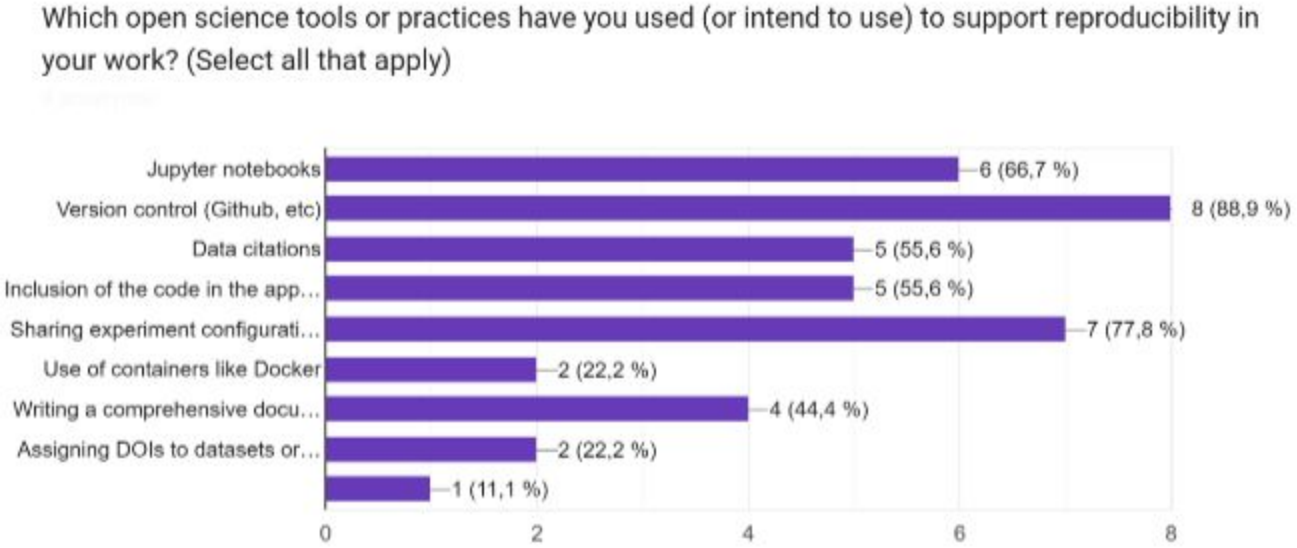


Figure 3.6: Open science tools used for reproducibility.

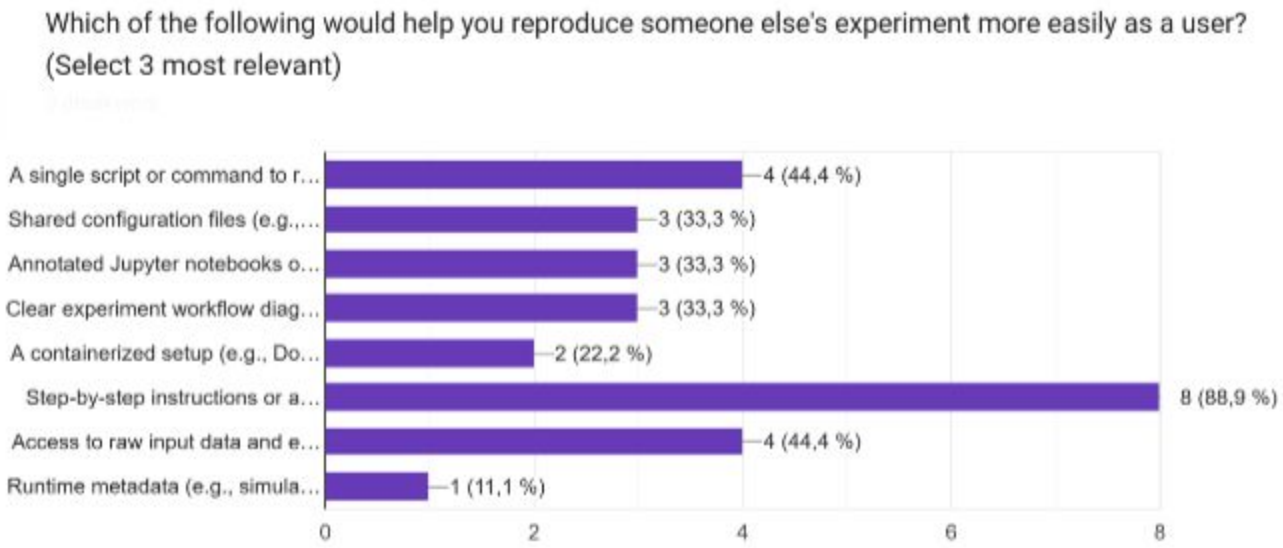


Figure 3.7: Preferred interfaces for reproducibility.

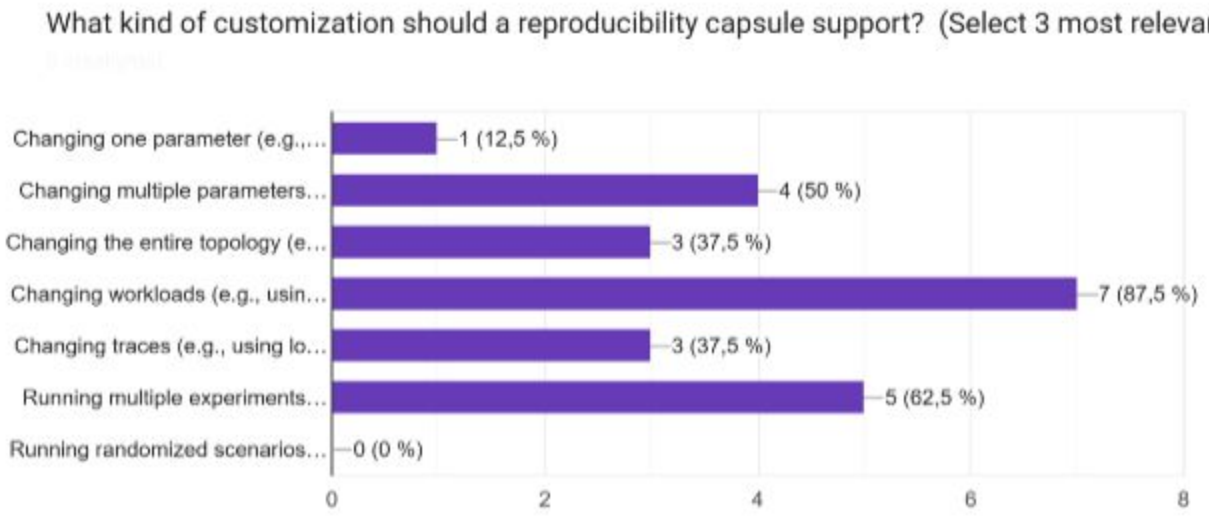


Figure 3.8: Customization options needed in capsules.

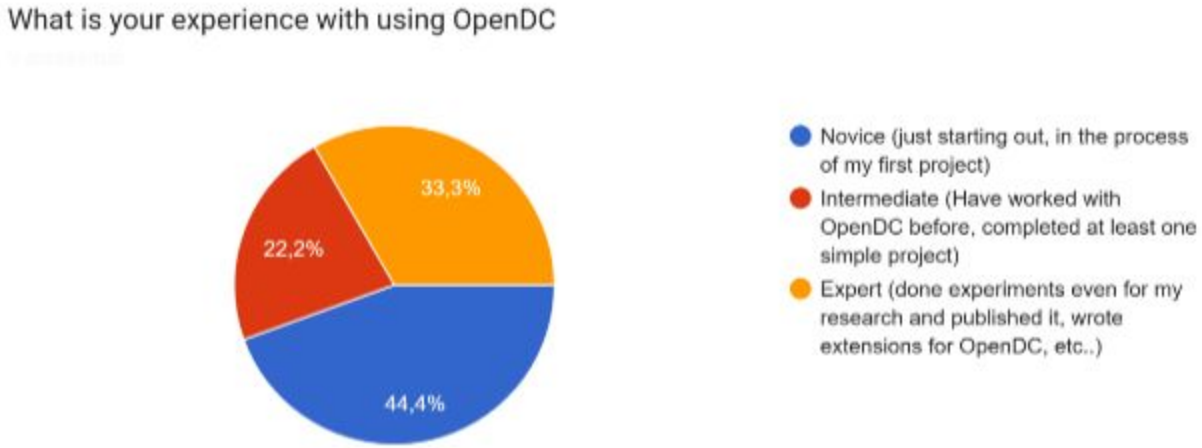


Figure 3.1: Participant experience level.

Table 5.1: Overview of supported values for topology generation of the prototype

Values	**Supported types**
Topology Template	Optional, file upload
CPU Count, CPU Speed, Memory Size, Number of Hosts	Single value, list, range
Carbon Trace	Optional, multi-select, file upload, select all
Battery Capacity, Starting CI, Charging Speed, Expected Lifetime	Single value, list, range
Power Model Type	Dropdown
Power, Max Power, Idle Power	Single value

Table 5.2: Overview of supported values for experiment generation of the prototype

Values	Supported types
Experiment Template	Optional, file upload
Topologies, Workload Traces, Failure Traces	Optional, multi-select, file upload, select all
Prefabs	Dropdown
Checkpoint Interval, Checkpoint Duration, Checkpoint IntervalScaling	Single value, list, range
Export Interval, Print Frequency, Seed, Runs	Single value, list, range
Max Num Failures	Single value, list, range
Files to Export	Multi-select

Reproducibility Capsule

This capsule contains all artifacts needed to reproduce the experiments listed below using OpenDC.

Experiments Overview

Capsule Metadata

- **Created on**: 2025-06-29
- **OpenDC Version**: 2.4e
- **Capsule Tool Version**: 1.0.0

Experiment 1: `carbon_experiment.json`

- **Topologies**: 4 files

▶ Show Topology List

- **Workloads**: 1 files

▶ Show Workload List

Execution Time per Experiment

Experiment	Duration (seconds)
carbon_experiment.json	76.61

Experiments were executed on the following system, we recommend to have at least these specifications when rerunning the experiments

System Information

- **Machine**: AMD64
- **Processor**: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
- **Cores**: 4
- **Threads**: 8
- **Memory**: 31.84 GB
- **Platform**: Windows-10-10.0.26100-SP0

How to Run

1. Make sure to have Java 21 and Jupyter Notebooks installed
2. Open `main.ipynb` in a Jupyter environment.
3. Click **'Run All Experiments'** to execute everything in the queue.
4. Outputs will appear in the `output/` directory.