Automatic Code and Test Generation of Smart Contracts from Coordination Models (Artifact)

Elvis Konjoh Selabi
Università di Camerino, Italy
Gran Sasso Science Institute, L'Aquila, Italy

Maurizio Murgia
Gran Sasso Science Institute, L'Aquila, Italy

António Ravara
NOVA School of Science and Technology, Almada, Portugal

Emilio Tuosto
Gran Sasso Science Institute, L'Aquila, Italy

Abstract

The companion paper proposes a formal approach for specifying and implementing decentralised coordination in distributed systems, with a focus on smart contracts. The model captures dynamic roles, data-driven transitions, and external coordination interfaces, enabling high-level reasoning about decentralised workflows. A toolchain supports formal model validation, Solidity code generation (extensible to other smart contract languages), and automated test synthesis. Although targeting blockchain platforms, the methodology is platform-agnostic and may generalise to other service-oriented and distributed architectures. The expressiveness and practicality of the approach are demonstrated through modelling and realising coordination patterns in smart contracts.

This artifact accompanies our paper [1]. It provides a toolchain for generating smart contract code from EDAM (Extended Data-Aware Machines) specifications. The artifact includes the complete source code, a Docker image for easy deployment, pre-generated experiment data (generated code, automated tests, and mutation testing results), and reproduction scripts.

2012 ACM Subject Classification Software and its engineering → Formal methods; Theory of computation → Program semantics; Software and its engineering → Domain specific languages

Keywords and phrases Smart Contracts, Coordination Models, Formal Semantics, Role-Based Access, Decentralised Systems, Code Generation, Solidity, Verification

Digital Object Identifier 10.4230/DARTS.12.1.22

Funding *Maurizio Murgia*: The MUR "dipartimento di eccellenza"
António Ravara: Supported by FCT (Fundação para a Ciência e a Tecnologia) through the project NOVA LINCS (UID/04516/2025, `https://doi.org/10.54499/UID/04516/2025`).
Emilio Tuosto: The MUR "dipartimento di eccellenza"

Related Article Elvis Konjoh Selabi, Maurizio Murgia, António Ravara, and Emilio Tuosto, "Automatic Code and Test Generation of Smart Contracts from Coordination Models", in 40th European Conference on Object-Oriented Programming (ECOOP 2026), LIPIcs, Vol. 372, pp. 15:1–15:31, 2026.
`https://doi.org/10.4230/LIPIcs.ECOOP.2026.15`

Related Conference 40th European Conference on Object-Oriented Programming (ECOOP 2026), June 29–July 3, 2026, Brussels, Belgium

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2026 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

This artifact is designed to reproduce the experimental results from the paper. In particular, it can be used to reproduce the code generation, automated test generation, and mutation testing evaluations from the code generation and evaluation section of the paper. The artifact also enables programmers to write and test custom EDAM models and generate smart contracts. Specifically, it enables reproduction of:

- **Code generation from EDAM models**: The tool generates Solidity smart contracts from EDAM specifications for various models (e.g., asset transfer, ERC-20 tokens, AMM, basic provenance tracking) Table 2 in the paper.
- **Automated test generation**: We run the semantic to generate symbolic traces from the EDAM models which are used to generate test cases using random exploration of the finite state space of the network of EDAMs.
- **Mutation testing evaluation**: Mutation scores and results obtained using ReSuMo on the generated code, including comparison across different models and configurations.
- **Performance metrics**: Code generation time, test execution time, and mutation testing duration.
- **Case study**: The artifact can be used to reproduce the case study from the paper.

The artifact provides both pre-computed results (in `EXPERIMENT_DATA/`) and the means to regenerate them. Note that regenerated results may not match the paper exactly due to machine-dependent factors and the non-deterministic nature of symbolic trace generation.

2 Content

The artifact package includes:

- **EDAM Studio source code** (type: software; format: Python, TypeScript, OCaml, JavaScript): The complete implementation in `Studio/`, comprising:
  - `API/`: Django backend for code generation.
  - `GUI/`: React/Vite frontend for visual model editing.
  - `CLI/`: Command-line interface for batch code generation, testing and mutation testing of the generated code.
  - `edams-models/`: Predefined EDAM models in the JSON format (TypeScript/TSX) (assettransfer, c20, amm, basicprovenance, etc.), used for code generation and mutation testing.
  - `ReSuMo/`: Mutation testing tool for Solidity smart contracts, used to evaluate the robustness of the generated code.
- **Experiment data** (type: data; format: ZIP, XLSX): Located in `EXPERIMENT_DATA/`:
  - `Generated Code.zip`: Pre-generated Solidity contracts and tests for the paper's models
  - `Mutations.zip`: Generated mutations from ReSuMo
  - `Excels Mutations Results.zip`: Detailed mutation testing results per operator
  - `Mutation result - Recaps.xlsx`: Summary mutation scores and timing data
- **Docker configuration** (type: software; format: Dockerfile, docker-compose): In `Docker/` for containerized deployment.
- **Documentation**: `README.md` and `SETUP_README.md` with installation and usage instructions.

Reusability. The implementation is open-source (GPL-3.0). To compile execute `Studio/install.sh` (Linux/macOS) or `Studio/install.bat` (Windows). The EDAM models and benchmarks are public and included in the repository. The artifact can (1) generate smart contracts from custom EDAM models via the GUI or CLI; (2) integrate the code generation pipeline into other tools via the API; (3) extend the mutation operators in ReSuMo; (4) add new EDAM models by implementing the model interface in edams-models/edam/types.ts.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). The artifact is available from two sources:

Get image from Zenodo (Download the zip file and load it with the docker load command) `https://doi.org/10.5281/zenodo.18475933`

```
gunzip -c edam-studio-docker-image.tar.gz | docker load
docker run -d -p 3000:3000 -p 5000:5000 \
--name edam-studio edam-studio:latest
```

Docker Hub

```
docker pull loctet/edam-studio:latest
docker run -d -p 3000:3000 -p 5000:5000 \
--name edam-studio loctet/edam-studio:latest
```

Access: GUI at `http://127.0.0.1:3000`, API at `http://127.0.0.1:5000`.

GitHub (source code and manual setup): `https://github.com/loctet/Edam-Studio`

For manual installation, see `README.md` and `SETUP_README.md` in the repository. Or generate the Docker image from the source code:

```
cd Docker
./build-docker.sh
```

Quick-start (kick-the-tires):

1. Pull and run the Docker image (commands above) or start the frontend and backend services manually.
2. Open `http://127.0.0.1:3000` and verify the GUI loads.
3. Generate code via CLI:

   ```
   docker exec -it edam-studio edam-cli cop --mode 4
   ```

   This will generate the code of figure 3 in the paper.
4. Check generated output (a ZIP file) in the container: `/app/Studio/Generated-code/` Using the docker container:

   ```
   docker exec -it edam-studio bash -c \
     "cd /app/Studio/Generated-code && ls"
   ```

4 Tested platforms

OS and resources used by the authors. Experiments performed on a Dell XPS 8960, 13th Gen Intel Core (9 -13900K) with 32 cores and 32GB RAM running Linux 6.5.0-44-generic (Ubuntu 24.04.2, 64bit).

Scaled-down evaluation. For time-constrained review, use `-number_symbolic_traces 50 -number_transition_per_trace 2` to reduce generation time. Run mutation testing on a single model (e.g., `assettransfer`) instead of all models.

Known compatibility. The Docker image is built for Linux/amd64. Windows and macOS users can run via Docker Desktop. opam/OCaml may require WSL on Windows for manual installation.

5 License

The artifact is available under the GPL-3.0 License. See the `LICENSE` file in the repository. The EDAM models and benchmarks are public and included in the repository.

6 MD5 sum of the artifact

d3507c5c9122294651d1b088de6559c8

7 Size of the artifact

998 MB

References

1 Elvis Konjoh Selabi, Maurizio Murgia, Antonio Ravara, and Emilio Tuosto. Automatic Code and Test Generation of Smart Contracts from Coordination Models. In *40th European Conference on Object-Oriented Programming (ECOOP 2026)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPIcs.ECOOP.2026.13.

A CLI Usage

The EDAM Studio CLI provides two main interfaces for command-line workflows: `edam-cli` for code generation, and `CLI/cli.sh` for running tests, mutation analysis, and experiment data processing.

Code generation (`edam-cli`). When running via Docker, use `edam-cli` to generate Solidity contracts from EDAM models:

```
docker exec -it edam-studio edam-cli \
<model1> [model2] ... --mode <3|4> [options]
```

Models can be predefined names (e.g., `assettransfer`, `c20`, `amm`). The `-mode` parameter is required: 1 (single trace), 2 (few traces), 3 (many traces), or 4 (all models). Generated ZIP files appear in `/app/Studio/Generated-code/` inside the container. See Table 2 for all optional parameters.

Auxiliary commands (`cli.sh`). From within the container (or locally with `cd Studio`), use:

- `.generate edams <models> -mode <3|4> [options]`: Generate code (equivalent to `edam-cli`).
- `.run test <zip_file> [test|coverage]`: Extract a generated ZIP, run `npm install`, then execute Hardhat tests or coverage.
- `.run resumo <zip_file>`: Run ReSuMo mutation testing on a generated ZIP file.

- `.run experiment_data [-mutation <zip_file>]`: Process all ZIPs in `EXPERIMENT_DATA/Generated Code.zip` (tests and coverage; optionally run mutation on a specific ZIP).

Example (using the docker container) for testing the model `assettransfer`:

```
docker exec -it edam-studio bash -c \
"cd /app/Studio && CLI/cli.sh .run test 'assettransfer_5_0 .01.zip' test"
```

For mutation testing of the model `assettransfer`:

```
docker exec -it edam-studio bash -c \
"cd /app/Studio && CLI/cli.sh .run resumo 'assettransfer_5_0 .01.zip'"
```

List of all models available in the `Studio/edams-models` directory:

Table 1 List of all models available in the `Studio/edams-models` directory.

Model Name	Description
`assettransfer`	Asset transfer Model
`basicprovenance`	Basic provenance tracking Model
`defectivecounter`	Defective counter Model
`digital_locker`	Digital locker Model
`frequentflyer`	Frequent flyer program Model
`helloblockchain`	Hello blockchain Model
`refrigeratedtransport`	Refrigerated transport tracking Model
`simplemarketplace`	Simple marketplace Model
`thermostatoperation`	Thermostat operation Model
`amm`	Automated Market Maker Model
`c20`	ERC-20 token Model
`c20_2`	ERC-20 token variant Model
`simplewallet`	Simple wallet Model
`c1, c2, c3`	C1, C2, C3 Models in figure 1 of the paper
`cop`	COP Model for figure 3 of the paper
`cpay`	Payment Model with C20 for figure 4 of the paper
`cm`	Case study Model of the paper

To generate code for any of the models (except AMM, COP, CPAY, CM) in the table, use the following command:

```
docker exec -it edam-studio edam-cli <model_name> --mode 4 [options]
```

Replace `<model_name>` with the name of the model you want to generate code for. The options are the same as the ones listed in Table 2.

To generate code for models AMM, COP, CPAY, CM, use the following command:

```
docker exec -it edam-studio edam-cli c20 c20_2 amm --mode 3 [options]
```

Here we have to pass the network of all models. In the case of AMM, we have to pass the network of C20, C20_2 and AMM. Likewise, for COP, we have to pass the network of C20 and COP, for CPAY, we have to pass the network of C20 and CPAY, for CM, we have to pass the network of C20 and CM.

B Reproduction of Experimental Claims

For each experimental claim in the paper, the following commands and procedures enable reproduction:

B.1 Running the Case Study

The case study is run on the model `cm` with the following command:

```
docker exec -it edam-studio edam-cli c20 cm --mode 3 \
  --number_symbolic_traces 1 --number_transition_per_trace 1 \
  --max_fail_try 1 --probability_new_participant 0.1
```

That will generate the code and the tests for the model `cm`. To run the semantic on the model `cm`, we can use the following command:

```
docker exec -it edam-studio bash -c \
  "cd /app/Studio/API/base_code && bash cmd_test_run_cm_c20.sh"
```

This will run the semantic on the model `cm` and the result will be the screenshot seen the case study section of the paper.

B.2 Code Generation

Claim. EDAM Studio generates Solidity smart contracts from EDAM models.
Reproduction. Using the Docker container:

```
docker exec -it edam-studio edam-cli \
  assettransfer --mode 4
```

Generated code (a ZIP file named `<ModelName>_<number_of_total_traces>_<new_participant_probability>_<id>.zip` containing the generated code, tests, and configuration files) appears in `/app/Studio/Generated-code/` in the Docker container.

The Zip file can be generated and downloaded from the GUI by adding the model to the `Generation List` and clicking on the `Download Generated ZIP` button. For custom trace counts (as in the paper):

```
docker exec -it edam-studio edam-cli <model_name> --mode 4 \
  --number_symbolic_traces 2000 --number_transition_per_trace 50 \
  --max_fail_try 5 --probability_new_participant 0.1
```

Command template.

```
docker exec -it edam-studio edam-cli <model1> [model2] ... \
  --mode <3|4> [options]
```

Table 2 lists all optional parameters for code generation.

B.3 Pre-generated Code and Data

Claim. The data in `EXPERIMENT_DATA/` corresponds to the paper's experiments.
Reproduction. The archives `Generated Code.zip`, `Mutations.zip`, and `Excels Mutations Results.zip` contain the exact outputs used in the paper. No regeneration needed for verification.

Table 2 Full parameters for `edam-cli` code generation.

Parameter	Default	Description
Required		
`-mode`	–	Generation mode: 3 (all models form one network), 4 (Each model form one network)
Symbolic trace generation		
`-number_symbolic_traces`	200	Number of symbolic traces to generate per model
`-number_transition_per_trace`	10	Maximum transitions per symbolic trace
`-number_real_traces`	5	Number of real traces for validation
`-max_fail_try`	2	Maximum retries on trace generation failure
Probability parameters		
`-probability_new_participant`	0.35	Probability of adding a new participant in a trace
`-probability_right_participant`	0.7	Probability of selecting the correct participant
`-probability_true_for_bool`	0.5	Probability of `true` for boolean values
Value bounds for generated data		
`-min_int_value`	0	Minimum integer value in generated tests
`-max_int_value`	100	Maximum integer value in generated tests
`-max_gen_array_size`	10	Maximum size of generated arrays
`-min_gen_string_length`	5	Minimum length of generated strings
`-max_gen_string_length`	10	Maximum length of generated strings
Test generation options		
`-add_pi_to_test`	False	Add process invariants to generated tests
`-add_test_of_state`	True	Include state assertions in tests
`-add_test_of_variables`	True	Include variable assertions in tests
`-z3_check_enabled`	True	Enable Z3 SMT solver checks during trace generation

The following command will generate the code for the models `assettransfer`, `basicprovenance`, `defectivecounter`, `digital_locker`, `frequentflyer`, `helloblockchain`, `refrigeratedtransport`, `simplemarketplace`, `thermostatoperation`, `simplewallet`, and `c20`. The number of total traces is 10000 for each model and thus may take a long time to generate.

```
docker exec -it edam-studio edam-cli \
  assettransfer basicprovenance defectivecounter \
  digital_locker frequentflyer helloblockchain \
  refrigeratedtransport simplemarketplace \
  thermostatoperation simplewallet  c20 \
  --probability_new_participant 0.1 --number_symbolic_traces 2000 \
  --number_transition_per_trace 20 \
  --number_real_traces 5 --max_fail_try 5 --mode 4
```

For the models `amm`, the following command will generate the code for the models `c20`, `c20_2`, and `amm`:

```
docker exec -it edam-studio edam-cli \
   c20 c20_2 amm --probability_new_participant 0.1 \
  --number_symbolic_traces 2000 --number_transition_per_trace 50 \
  --max_fail_try 5 --mode 3
```

Running tests and mutation on pre-generated data. To run tests and coverage for all pre-generated zip files in `EXPERIMENT_DATA/Generated Code.zip`, use:

```
docker exec -it edam-studio bash -c \
  "cd /app/Studio && CLI/cli.sh .run experiment_data"
```

This command will:

1. Extract `EXPERIMENT_DATA/Generated Code.zip`
2. For each zip file found:
   - Extract the zip file
   - Run `npm install`
   - Run `npx hardhat test`
   - Run `npx hardhat coverage`
3. Display each processed file's results

This process may take a long time to complete as each file contains 10000 tests and thus may take several hours to complete.

Running mutation testing on a specific pre-generated zip. To run mutation testing (ReSuMo) on a specific zip file from the pre-generated data:

```
docker exec -it edam-studio bash -c \
 "cd /app/Studio && CLI/cli.sh .run experiment_data --mutation <zip_filename>"
```

Replace `<zip_filename>` with the name of the zip file (e.g., `assettransfer_0.35_1.zip`). The command will copy the zip to the Generated-code directory, run ReSuMo mutation testing, and clean up afterward.

For the experiments in the paper, the results are in the file `EXPERIMENT_DATA/Mutation result - Recaps.xlsx`. Detailed per-operator results are in `EXPERIMENT_DATA/Excels Mutations Results.zip`.

Note that mutation testing is time consuming and can take several hours to complete for some models (e.g., AMM, COP, CPAY, CM), also depends on size of the test suite and the number of mutants.

B.4 Mutation Testing Results

Claim: Mutation scores and timing reported in tables/figures.
Reproduction: Open `EXPERIMENT_DATA/Mutation result - Recaps.xlsx` for summary data. Detailed per-operator results are in `Excels Mutations Results.zip`. To reproduce mutation testing on a single model, using the Docker container:

```
docker exec -it edam-studio bash -c \
  "cd /app/Studio && CLI/cli.sh .run resumo <zip_filename>.zip"
```

Replace `<zip_filename>` with the name of the zip file (e.g., `assettransfer_0.35_1.zip`). The command will copy the zip to the Generated-code directory, run ReSuMo mutation testing, and clean up afterward.

B.5 Test Execution

Claim. Generated tests execute successfully and achieve coverage.
Reproduction.

```
docker exec -it edam-studio bash -c \
  "cd /app/Studio && CLI/cli.sh .run test <zip_file> test"
```

For coverage: `CLI/cli.sh .run test <zip_file> coverage`.

B.6 Timing Expectations

Results may differ from the paper due to:

- Machine speed (CPU, disk I/O)
- Non-deterministic symbolic trace generation (Z3 solver, random sampling)
- Node/npm version differences

Code generation: typically 5–20 min per model. Testing: up to 20 min for 10k tests. Mutation: less than 2 h for most models; AMM can take days.

C GUI Usage

The EDAM Studio GUI (available at `http://127.0.0.1:3000` when running via Docker) supports editing EDAM models, generating Solidity code, and verifying model behavior.

Editing the model. The model can be edited in three ways:

- **Force-Directed Graph** or **GraphViz**: The model is displayed as a graph. Edit by dragging nodes and edges, or use the sidebar to add transitions and states.
- **Manual Editor**: Enter the EDAM specification as text in the `Manual Editor` tab, then click `View EDAM` to load it into the editor. The text format uses a compact syntax (states, roles, variables, and transitions). If a model is saved in a .edam file, it can be loaded by clicking `Load EDAM File` and selecting the file, conversaly, the model can be saved by clicking `Save EDAM File` and selecting the file. The following is the format of a .edam file:

  ```
  ModelName
  Role1,Role2,Role3
  variable1:type1, variable2:type2
  [from_state] {userVar:role:mode}, guard, [external_calls] \
  callerVar:operation(params){assignments} {user:role:mode} [to_state]
  ```

  First line is the model name, second line is the roles, third line is the variables, and the following lines are the transitions. The transitions are formatted as follows:

  ```
  [from_state] {userVar:role:mode}, guard, [external_calls] \
  callerVar:operation(params){assignments} {user:role:mode} [to_state]
  ```

  The `from_state` is the source state, the `userVar:role:mode` is the user and role and mode, the `guard` is the guard expression, the `external_calls` are the external calls, the `callerVar:operation(params)` is the caller variable and operation and parameters, the `assignments` are the assignments, and the `to_state` is the target state.
- **JSON Editor**: The bottom panel provides a JSON editor for direct structural editing. Changes are reflected in the graph (Select some examples from the available models in the `edams-models` repository to see the JSON format).

Generating code. Add the desired model(s) to the `Generation List` via `Add to Generation List`, then click `Generate Code` in the header. After generation, the results appear with two tabs:

- **Generated Code**: View the generated Solidity contracts. Use `Download Generated ZIP` to save the code (contracts, tests, and configuration) locally.
- **Run Solidity Test**: Execute the generated tests on the server and view pass/fail results.

Running a trace test. To verify model behavior without generating full code, click `Run A Trace Test` (bottom-right). Enter trace commands in the text area (e.g., `p1>Token1.deploy [] [10]`) and click `Test Trace` to execute. The tool simulates the trace against the model and reports the outcome.