Vahana.jl Workshop

Part II - Handling Simulation Data

Steffen Fürst

Scaling Complexity Workshop
Berlin, 03.12.2024

Session Goals

1. Writing/Restoring Simulations
   - Snapshots & metadata
   - Postprocessing example: Plot showing the development of opinions over time
2. Global Values
   - Mean opinion tracking
   - Visualization
3. Simulation Visualization
   - Graph plotting
   - Edge/node styling

Vahana.jl uses HDF5 for simulation storage

HDF5 is a hierarchical file format optimized for storing large scientific datasets.

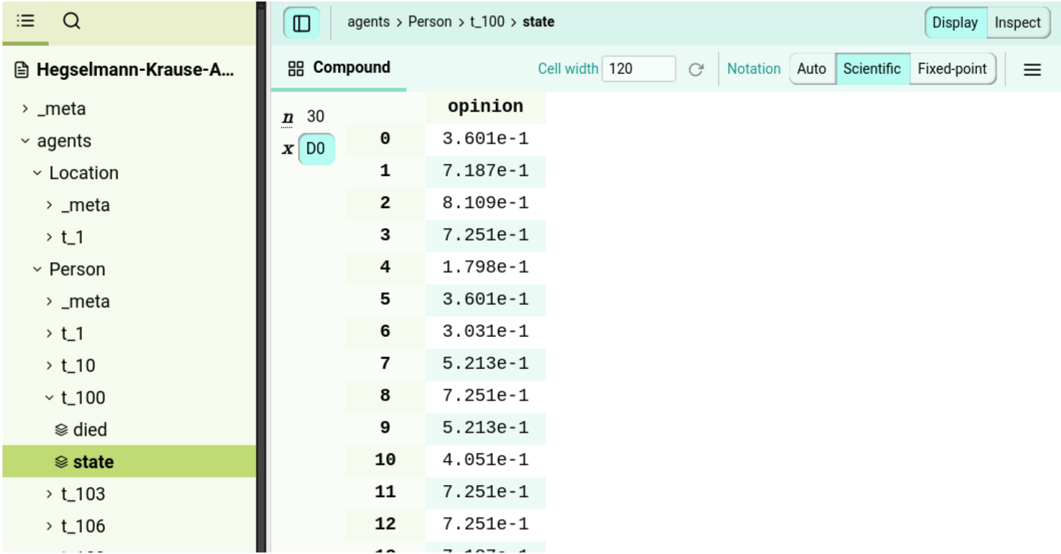
Benefits:

- Self-describing through metadata
- Platform independent
- Efficient storage and access of large datasets
- Single file for the whole simulation dataset
- Partial data access (without loading entire file)
- Supports parallel read/write operations

Trade-offs:

- **HDF5.jl does not support all DataTypes**
- Complex file structure requires specific tools
- File corruption can affect entire dataset

A peek into an HDF5 file



Hegselmann-Krause-A...

- _meta
- agents
  - Location
    - _meta
    - t_1
  - Person
    - _meta
    - t_1
    - t_10
    - t_100
      - died
      - state
    - t_103
    - t_106

agents › Person › t_100 › **state**

Display [✓] | Inspect

Compound

Cell width 120

Notation: Auto | Scientific [✓] | Fixed-point

n 30

x D0

	opinion
0	3.601e-1
1	7.187e-1
2	8.109e-1
3	7.251e-1
4	1.798e-1
5	3.601e-1
6	3.031e-1
7	5.213e-1
8	7.251e-1
9	5.213e-1
10	4.051e-1
11	7.251e-1
12	7.251e-1

Using HDF5 in Vahana.jl

Writing Simulations

```
# Save complete simulation state
write_snapshot(sim, "snapshot_1")

# Add custom metadata
write_sim_metadata(sim, "steps", 100)
```

Reading Simulations

```
# Load specific snapshot
read_snapshot!(sim; comment = "snapshot_1")

# Access metadata
steps = read_sim_metadata(sim, "steps")
```

Using HDF5 in Vahana.jl

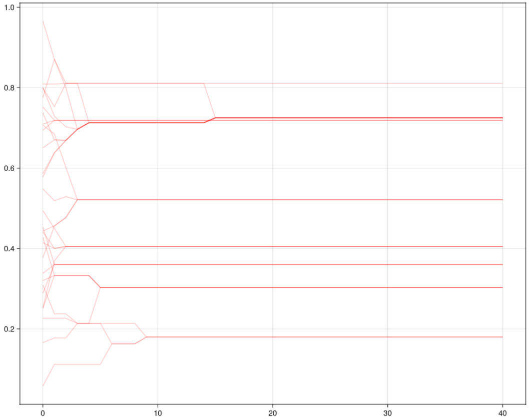
`read_snapshot!` requires an existing and matching simulation instance. If you don't need to restore the complete simulation, alternatives exist:

```
read_agents("Hegselmann-Krause-Analysis", Person)
```

Per default, files are stored in and read from a `h5` subfolder of the current working directory. You can customize this location using `set_hdf5_path(path::String)`

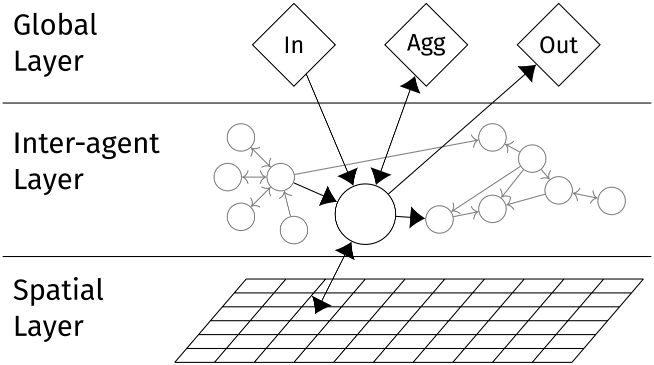
Task 1

The first task involves writing snapshots for each step, then reading and combining all opinions over time to generate this visualization:



The Global Layer

The global layer conceptually contains **vertices linked to all others**, allowing distribution of parameters and aggregation of agent data, and has also a set of **helper functions**.



Working with Global Values in Vahana.jl

Globals can either be defined as a struct and passed to `create_simulation`, or registered individually using `register_global!` on the model, where the last argument is the initial value.

```
register_global!(model, :mean_opinion, Float64[])
```

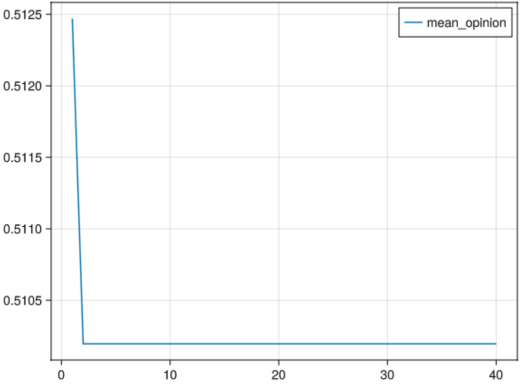
You can set a global value using `set_global!`, or in the case of a vector-type global value, append additional values using `push_global!`.

To access a global value, use `get_global`

Visualization of Time Series Data from Globals:

Global value vectors can be visualized using the provided `plot_globals` function:

```
plot_globals(sim, :mean_opinion) |> first
```



Aggregate values / mapreduce

To aggregate values in distributed simulations, we need to consider that agents are scattered across different processes. Vahana.jl handles this with `mapreduce`, which automatically collects and combines values from all processes, e.g. . . . ,

```
mapreduce(sim, a -> a.opinion, +, Person)
```

. . . to sum opinions across the entire simulation.
Task 2 is to add a global value tracking the mean opinion across all agents (although this metric is not be particularly meaningful in this context).

Interactive Graph Plots

Task 3 is to create an interactive visualization of the simulation using GraphMakie.jl, showing agents, locations, and their connections.