Vahana.jl Workshop

Part I - Vahana.jl Basics & First Model Implementation

Steffen Fürst

Scaling Complexity Workshop
Berlin, 03.12.2024

Introducing Vahana.jl

- Vahana is an open source HPC framework designed for **large-scale ABMs**, particularly those focusing on complex social systems.
- Vahana has a focus on **network-based models**, although it is allowed to include a spatial dimension.
- Vahana simplifies the development phase with its **interactive capabilities**, using Julia's REPL.
- Once developed, a Vahana simulation can be **run parallelized** without additional actions from the developer.
- Models must be formulated as a **Graph Dynamic System** (GDS), which imposes certain constraints.

(Synchronious) Graph Dynamical Systems (GDS)

- A GDS involve a **finite graph** with a state assigned to each vertex and a **vertex function** that update the state of a vertex based on the states of its neighboring vertices.
- In a synchronious GDS (SyGDS), all vertex state updates happen **simultaneously**.
- A SyGDS can be interpreted as a **generalized cellular automata**, where the neighborhood is determined by the network instead of the position of the cell.
- When an ABM is implemented as a GDS, the **vertices represent the individual agents** of the model.

Vahana's SyGDS Extension

We have ...

- A **directed graph** $G_t = (V_t, E_T)$.
- Vertices of **different types** $\delta_{\mathrm{v}} \in \Delta_{\mathrm{V}}$, and a mapping function $\pi_v : V_t \to \Delta_V$.
- Likewise edges of **different types** $\delta_{\mathrm{e}} \in \Delta_{\mathrm{E}}$, and a mapping function $\pi_e : E_t \to \Delta_E$.

Vahana's SyGDS Extension

We have ...

- A **directed graph** $G_t = (V_t, E_T)$.
- Vertices of **different types** $\delta_\mathrm{v} \in \Delta_\mathrm{V}$, and a mapping function $\pi_v : V_t \to \Delta_V$.
- Likewise edges of **different types** $\delta_\mathrm{e} \in \Delta_\mathrm{E}$, and a mapping function $\pi_e : E_t \to \Delta_E$.
- For each δ_v and δ_e a different **state space** $\Theta_\delta = X_{\delta,1} \times \ldots \times X_{\delta,n_\delta}$, where each $X_{\delta,i}$ is itself a set (like e.g. $\mathbb{N}$).
- At least one **transition (vertex) function** $f_{\delta_\mathrm{v},i} : G'_{v,t} \to (V'_{v,t+1}, E'_{v,t+1})$, where $G'_{v,t}$ is the 1-neighborhood of vertex $v \in G_t$.
- The source v_s and target v_t of an $e \in E'_{v,t+1}$ must be vertices of $G'_{v,t} \cup V'_{v,t+1}$.

Vahana's SyGDS Extension

We have ...

- A **directed graph** $G_t = (V_t, E_T)$.
- Vertices of **different types** $\delta_\mathrm{v} \in \Delta_\mathrm{V}$, and a mapping function $\pi_V : V_t \to \Delta_V$.
- Likewise edges of **different types** $\delta_\mathrm{e} \in \Delta_\mathrm{E}$, and a mapping function $\pi_e : E_t \to \Delta_E$.
- For each δ_v and δ_e a different **state space** $\Theta_\delta = X_{\delta,1} \times \ldots \times X_{\delta,n_\delta}$, where each $X_{\delta,i}$ is itself a set (like e.g. $\mathbb{N}$).
- At least one **transition (vertex) function** $f_{\delta_\mathrm{v},i} : G'_{v,t} \to (V'_{v,t+1}, E'_{v,t+1})$, where $G'_{v,t}$ is the 1-neighborhood of vertex $v \in G_t$.
- The source v_s and target v_t of an $e \in E'_{v,t+1}$ must be vertices of $G'_{v,t} \cup V'_{v,t+1}$ **or a raster vertex.**

Constructing G_{t+1}

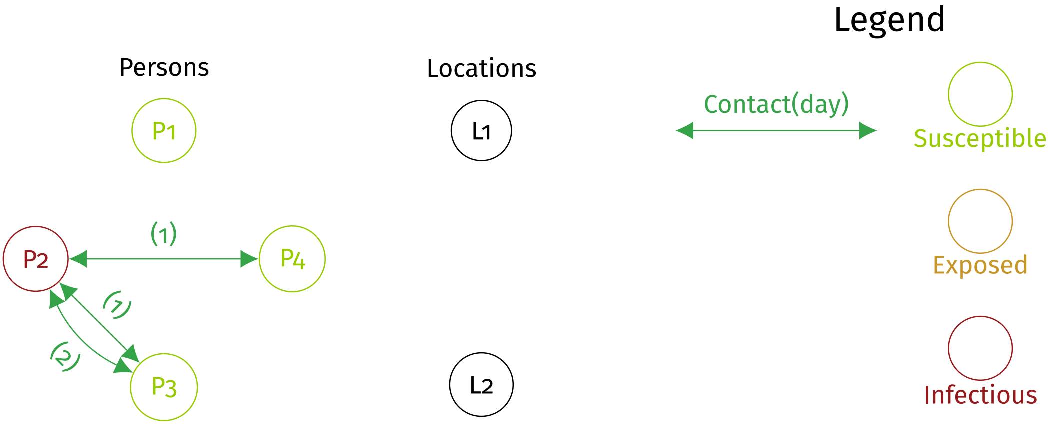
To construct G_{t+1}, we take the following actions:

- For each vertex $v \in V_t$ we **apply** $f_{\pi_v(v),i}$.
- **Combine** the results: $G_{t+1} = (\bigcup_{v \in V_t} V'_{v,t+1}, \bigcup_{v \in V_t} E'_{v,t+1})$.
- **Remove** all $e \in E_{t+1}$ with a source v_s or target v_t that is not in V_{t+1}.

It would be inefficient and cumbersome to require that even **invariant parts of the graph** must be reconstructed, so:

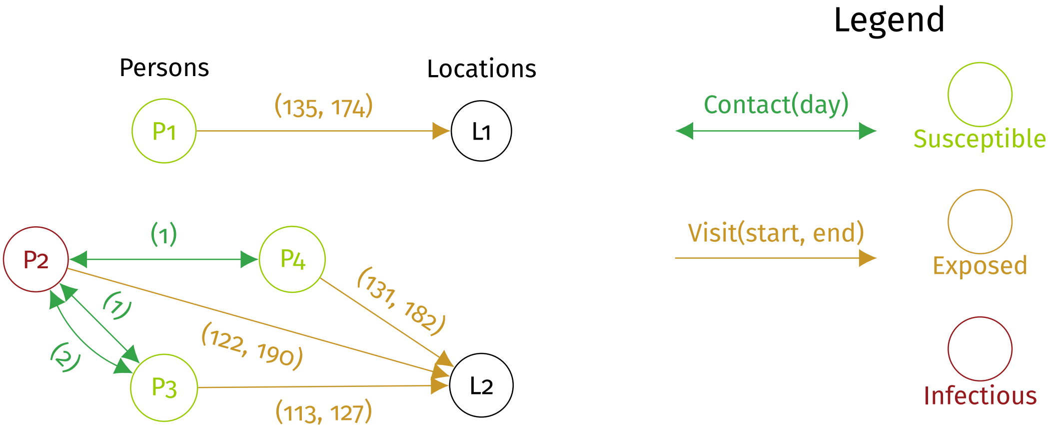
- Users specify which vertex/edge types are **mutable** during transitions.
- There exist an option to retain existing agents/edges of a type, **only adding new ones**.

Example: A epidemiological model



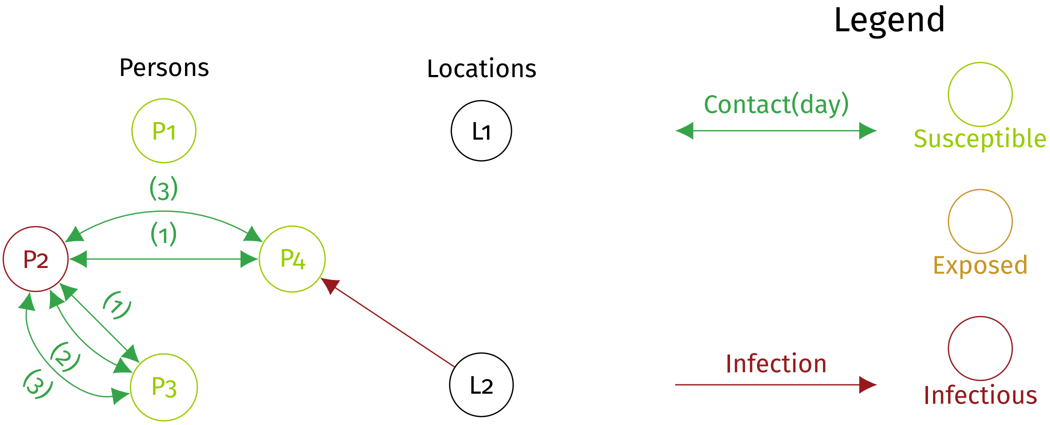
Start of Day 3

Example: A epidemiological model



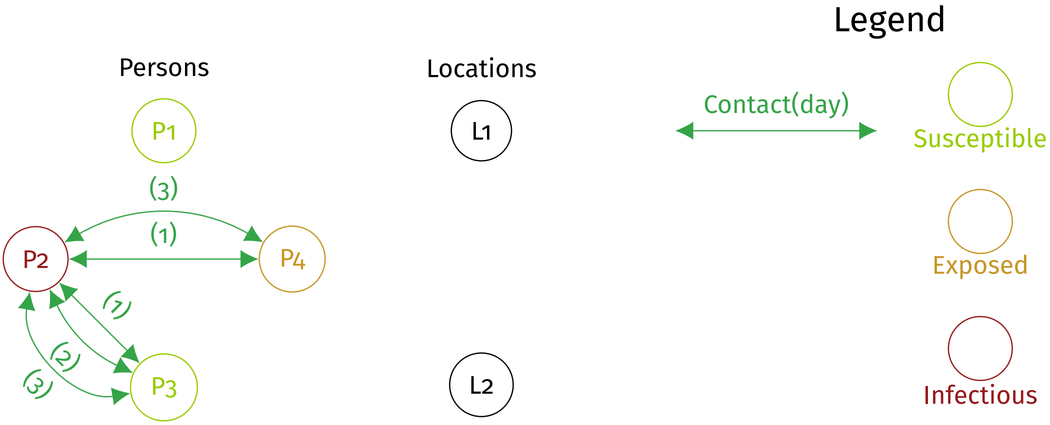
Persons: Plan day

Example: A epidemiological model



Locations: Infect persons & add contacts

Example: A epidemiological model



Persons: Update health status

REPL example

```
julia> show_agent(sim, Person; neighborstate = true)
Id / Local Nr: 0x0100000000085e35 / 548405
State:
    health=susceptible
    age=41
    quarantine=Quarantine(false, -1, -1)
Edge(s):
    MaybeEvent
        from:              edge.state:
        0x020000000007691a MaybeEvent(0, ACTSTART, HOME)
        0x020000000007691a MaybeEvent(32520, ACTEND, HOME)
        ... (16 not shown)
    Home
        from:
        0x020000000007691a
    Knows
        from:              neighborstate:
        0x0400000000000001 capacity=0, traced=0, reported_infections=0,
```

Parallel Simulations

- Vahana.jl uses the **Message Passing Interface** (MPI) for communication across processes.
- But it **shields users** from complexities of MPI.
- At the end of the initialization phase, the actual graph is **automatically partitioned** and distributed to the processes.
- Each **process only knows a subset** of the graph representing simulation state.
- Vahana.jl uses **MPI-3 shared memory** to avoid copies of agent state on same node.

Hegselmann-Krause (HK) Opinion Model

- Finite number n of agents
- Time is **discrete**: t = 0, 1, ...
- Real number $x_i(t)$ represents the **opinion** of agent i
- There is a **confidence bound** $\epsilon > 0$, opinions with a difference greater then ϵ are ignored.
- Opinions are updated **synchronously** according to

$$x_i(t+1) = \frac{1}{|\mathcal{N}_i(t)|} \sum_{j \in \mathcal{N}_i(t)} x_j(t)$$

$$\text{where} \quad \mathcal{N}_i(t) = \{j : \|x_j(t) - x_i(t)\| \leq \epsilon\}$$