An Agent-Based Model Formalization to Support Distributed Execution

Attila Szabó, Vivien Tóth, Richárd Legéndi

ELTE, Faculty of Informatics, Department of Programming Languages and Compilers

AITIA International Zrt.

ELTE-Soft Kft.

CSCS 2010., Szeged

Outline

1. Motivation & context of the research
2. Agent-based modeling (ABM)
3. Formalization and transformation example
4. Conclusions

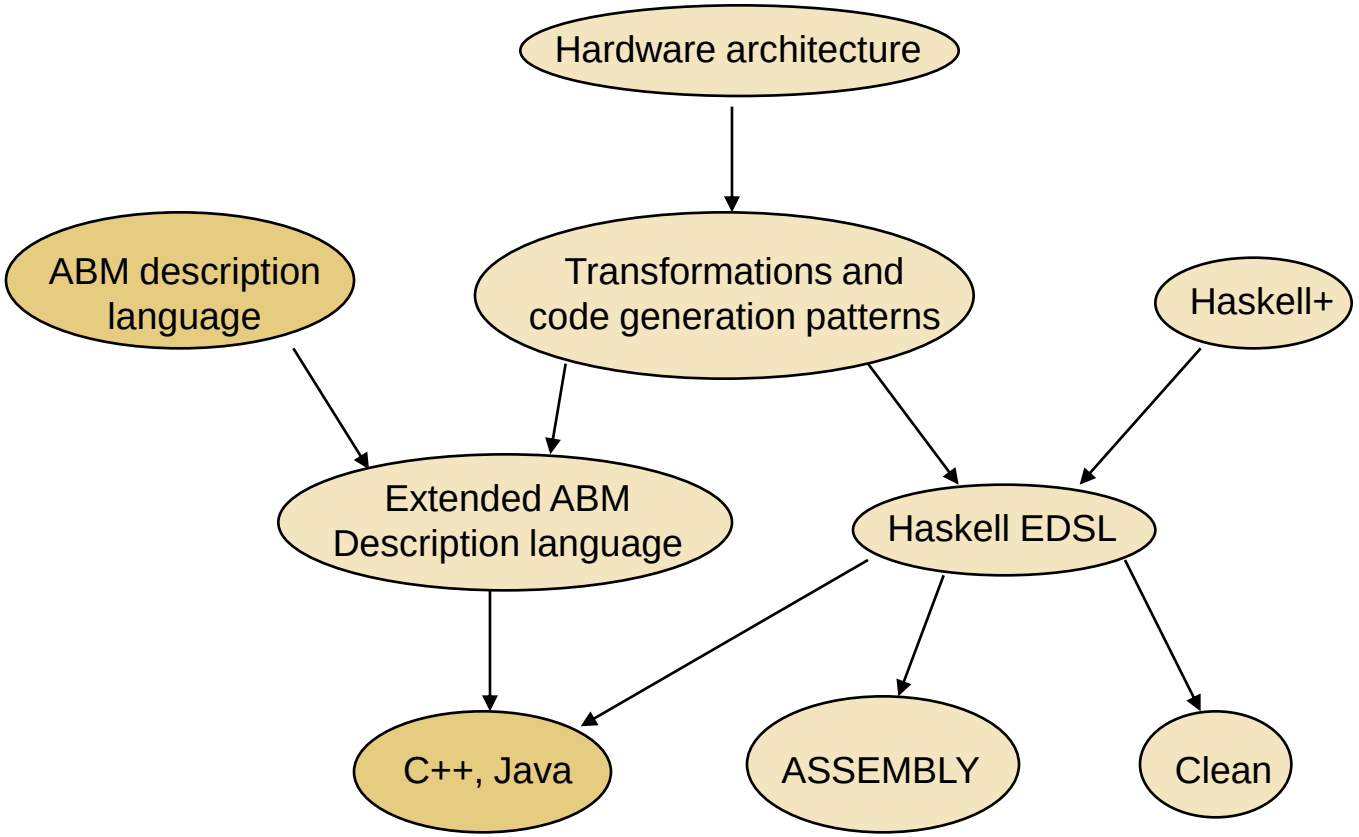
Motivation & Context

Motivation

- We can found free CPU capacity everywhere
- Nobody wants to wait days for experimental results
- Let’s run on multiply computers!
  - However development of distributed software needs a new focus on problem formalization

Context of the research

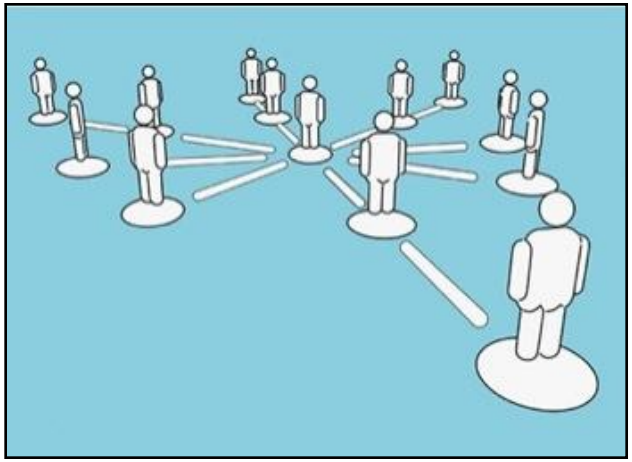
- Define a model once, and run on cloud, multi-core, single core



Agent-Based Modeling

Agent-Based Modeling I.

- Particularly: Agent-Based Social Simulation (ABSS)
- Bottom-up technique
- Helps to find emergent macro level processes by implementing micro level rules
- Computational model for simulating the interactions of agents



Agent-Based Modeling II.

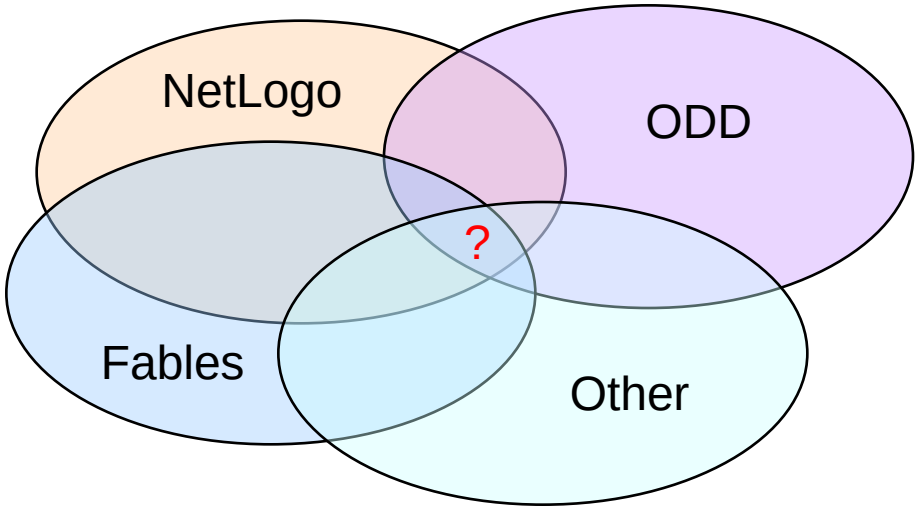
- Agents are usually:
  - autonomous
  - make own decisions
  - learn, adapt to the changes of the environment
  - interact through a network
- Discrete time simulation
  - Agents actions may be executed sequentially within one time step

Existing Formalizations

- Dozens of software libraries and languages exist
  - each contain a formalization or „design pattern” of model execution
  - each support visualization, data recording, and other usual related tasks
  - Platforms: *Netlogo, Repast, Swarm, Mason, Fables etc.*
  - Programs are unambiguous definitions, but hard to understand
- There are also model description languages
  - Like the *ODD (Overview-Design-Details)* protocol: informal, uncompleted

Formalization “chaos”

- Nearly all software tool has its own way to define models
- What’s common in them?



Goal of Formalization

- Our goal is to improve model execution performance by utilizing multi-core computer architectures (including clusters, etc.)
- Two ways:
  - Parallel execution of simulation instances (more runs in the same time)
  - **Distributed execution of one simulation** (more agents in a model)
    - We need a simple, but unambiguous formalization

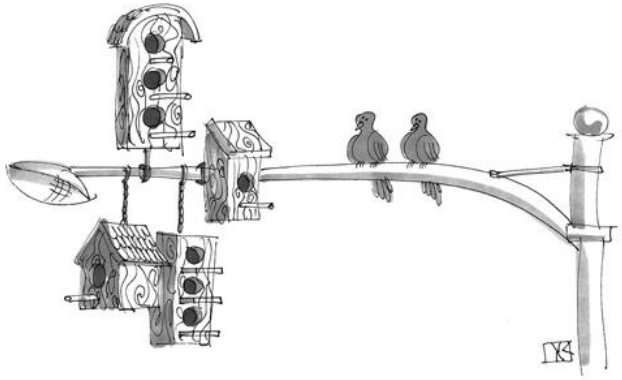
The Urban Sprawl model example

Formalization and distributed simulation

The Urban Sprawl model[1]

Informal model description

- birds live in a changing environment
- a city grows in their territory, that changes the food productivity in the affected areas
- actors are: *birds, built-ups, greenbelts, land areas + environment*
- birds' activities: *move, breed, eat and die*
- built-ups 'reproduce' themselves
- plus many behavioral rules…



[1] *http://www.openabm.org/model-archive/bird_urbansprawl*

The formalized model

Bird

- **energy level**, **location**, *birth probability, death probability, vision, max. number of birds in the same location, reproduction energy level*, rural preference, urban preference
- Die, Reproduce, Move, Metabolism
- Dynamics:
  - reproduce before move,
  - move all before die,
  - die before metabolism

Land

- **location**

Built-up

- **location,** *growth rate, max. number of built-up agents in the same location, urban area radius level*
- Reproduce, Move
- Dynamics:
  - reproduce before move

Environment

- *create greenbelt probability*
- Create greenbelt
- Dynamics:
  - create greenbelt all before Built-up.reproduce

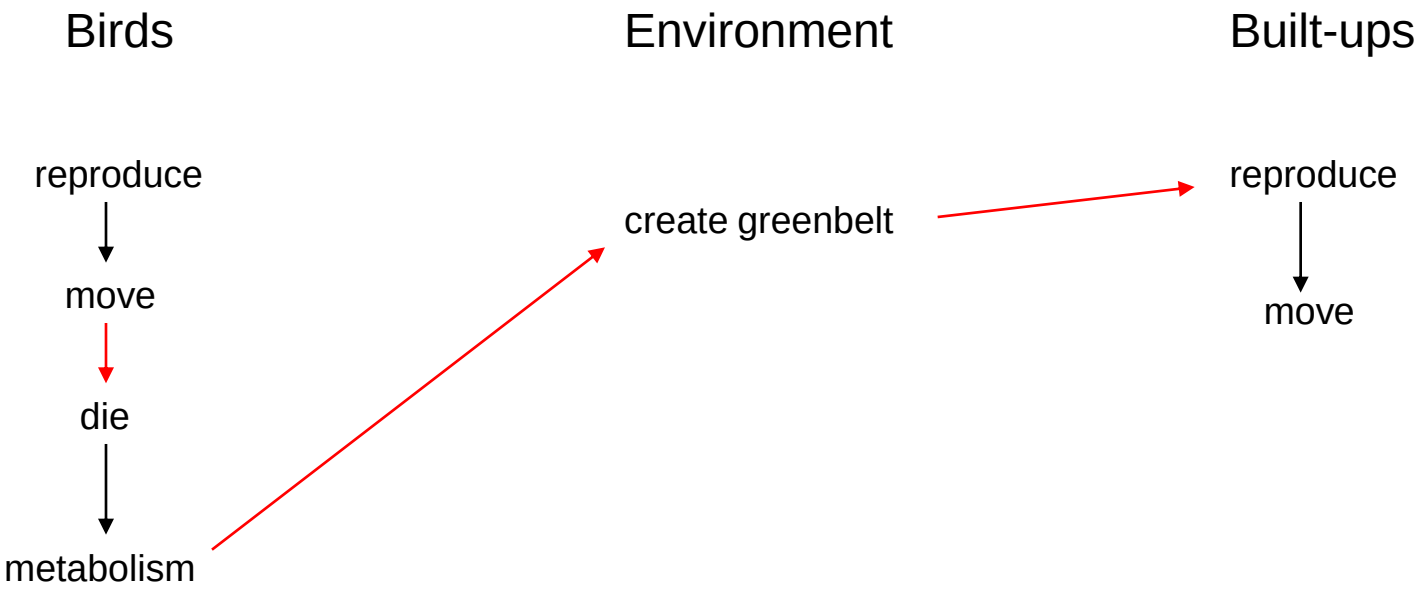
Greenbelt

- **location**

Activity relations

- Before: describes activity relation *within* an agent's step
  - transitive: eat precedes all other activity
- All before: describes activity relation for all agents' of the specified type(s) step
  - transitive: every agent performed reproduce and move activities before anyone decides to die
  - can be applied to any two activities, e.g.: Environment.create_greenbelt and Built-up.reproduce
- Cycles must be avoided!

The Graph



red arrows: all before, black arrows: before

How to obtain distributed code?

- Unrelated agent activities can be performed in the same time (on different CPUs)
  - Note that e.g. $bird_i$.move is unrelated to $bird_j$.move ($i \neq j$)
- The "all before" relations slice the execution (synchronization points)

Code example

```
while( !isEnd ){
        for(Bird b: birds){
                b.reproduce();
                b.move();
        }

        //synchronize
        for(Bird b: birds){
                b.die();
                b.eat();
        }
        //other agents' activities
}
```

Code example

```
while( !isEnd ){
        for(Bird b: birds){
                b.reproduce();
                b.move();
        }

        //synchronize
        for(Bird b: birds){
                b.die();
                b.eat();
        }
        //other agents' activities
}
```

may be executed on multiply CPUs

"move all before die"

may be executed on multiply CPUs

Conclusions and Further Work

Conclusions

- ABM formalization is easy – many solutions are available…
- Our approach motivates a higher abstraction level
  - Static model description (attributes)
  - Dynamic model description (activities)
    - Relation of activities (→ parallel execution)

Further work

- Formalize & implements as many types of models as we can
  - Optimize implementation techniques
  - Extend the formalization if necessary
  - Identify all problem classes where our approach is useful
- Include hardware description in code generation decisions

Thank you for your attention!