A Hands-on IODA Tutorial

Interaction-Oriented Simulation within NetLogo

Sébastien Picault

sebastien.picault@univ-lille1.fr

SMAC team – LIFL – Lille 1 University

http://www.lifl.fr/SMAC/

Social Simulation Conference

September 1st, 2014

Outline of the tutorial

1. The Interaction-Oriented Approach (IODA)
2. The IODA extension for NetLogo
3. Let's code !

The Interaction-Oriented approach (IODA)

Limitation of classical MABS approaches

- excessive focus on individuals
  - strong dependencies agents/behaviors
  - make model revisions difficult
- alternative: separation declarative/procedural :
  - simplifies expertise acquisition
  - enhanced model intelligibility
  - increased reutilisability
  - homogeneity of the concepts

The notion of interaction

behavior = abstract description + modalities

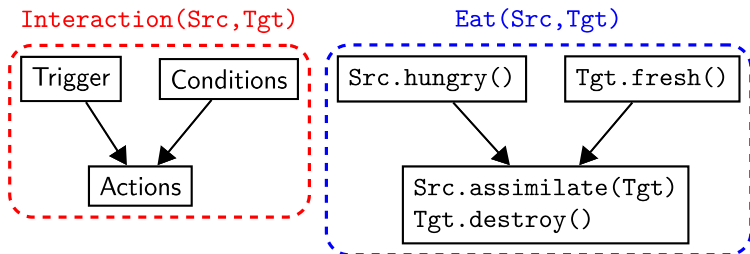
abstract description :

conditions/actions rules = interactions
rely upon generic perception/action capabilities
(**primitives**)

+ execution modalities :

variability in the primitives,
depending on agent families

Structure of an interaction



Trigger: motivations/goals for doing the actions

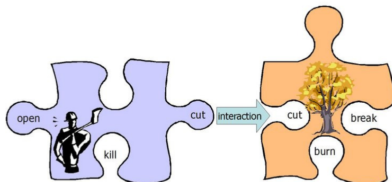
Conditions: prerequisite that allow the realization of the actions

Actions: list of the actions to perform

The interaction-oriented approach

3 key ideas

- each relevant entity is an agent
  = entity with **perception** and **action** capabilities
- each behavior is an interaction
  = **conditions/actions rule** involving several agents
- a generic engine determines what interactions may occur



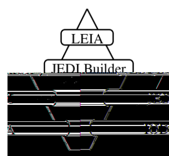
agents characterized by their capability to **perform** (source) or **undergo** (target) an interaction

[Mathieu & *al.* 2001]

The IODA approach

separation between declarative/procedural parts:

- automatization of model implementation and of simulations analysis
- elicitation of simulation biases [Kubera & *al.* 2008]



IODA Interaction-Oriented Design of Agent simulations
methodology and modelling frame

JEDI Java Environment for the Design of agent Interactions
highly tunable platform
[Kubera & *al.* 2011]

JEDI-Builder Code generator
IODA model → JEDI simulation

LEIA LEIA lets you Explore Interactions for your Agents
Explorer for the simulation space
[Gaillard & *al.* 2010]

Interaction and update matrices

Sources \ Targets	∅	Trees	Grass	Sheep	Goats	Wolves
Trees						
Grass	Grow (0) Spread (1)					
Sheep	Die (3) Wander (0)		Eat (2; 0)	Mate (1; 0)		
Goats	Die (4) Wander (0)	Climb (3; 1)	Eat (2; 0)		Mate (1; 0)	
Wolves	Die (4) Wander (0)			Eat (3; 1)	Eat (2; 1)	Mate (1; 0)

	UPDATE
Trees	
Grass	
Sheep	Age (2) BecomeSick (1)
Goats	Age (2) BecomeSick (1)
Wolves	Age (1)

- synthetic view of all behaviors
- actions resulting from decisions vs. spontaneous state changes

An homogeneous handling of entities

- each relevant entity is an agent
- the actitivy of agents is characterized dynamically:

  active agent: acts upon the others
  = no-empty line in the interaction matrix

  passive agent: undergoes actions from other entities
  = non-empty column in the interaction matrix

  labile agent: spontaneous state change
  = non-empty line in the update matrix

- optimization of the simulation engine

[Kubera & *al.* 2010]

“Reactive” IODA engine

during a time step:

1. **Update:** each labile agent performs the realizable interactions from the update matrix
2. **Interaction selection:** each active agent
   1. perceives neighbors (among the passive agents)
   2. filters targets of interactions that it can perform i.e. agents which *can undergo* those interactions
   3. evaluates the triggers and conditions of those interactions
   4. selects one realizable interaction with the highest priority level
   5. performs the corresponding actions

The IODA extension for NetLogo

Overview

The principles and algorithms of IODA can be implemented on any platform

within NetLogo:

- Java extension $\longrightarrow$ required data structures
  + file reading functions
  + consistency checking / primitives to write
- NetLogo include file $\longrightarrow$ reactive simulation engine

Features

- agent families = `turtles`, *`breeds`* or `patches`
- target selection policies
- consistency checking
- easy tuning of perception
- physical/social environnements (`links`)

`http://www.lifl.fr/SMAC/projects/ioda/ioda_for_netlogo/`

Principles

a **IODA NetLogo** simulation =

- classical NetLogo program using the extension
  + definition of the concrete primitives
  for each agents family
- definition of the interactions in a text file
- definition of the interaction and update matrices in a text file
- delegation of the scheduling to a dedicated procedure `ioda:go`

Example (1) – behaviors

- inspired by the classical `Termites` model in the NetLogo library
- everything is an agent: termites + wood chips

Source \ Target	∅	chips	termites
chips			
termites	(MoveRandomly, 0)	(MoveRandomly, 10, 1) (PutDown, 20, 0.3) (PickUp, 30, 1)	

Example (2) – NetLogo template

```
__includes ["../../IODA_2_2.nls"]
extensions [ioda]

breed [ termites termite ]
breed [ chips chip ]
termites-own [ carrying? ]

to setup
  clear-all
  create-termites nb-termites [ init-termite ]
  create-chips nb-chips [ init-chip ]
  ioda:load-interactions "interactions.txt"
  ioda:load-matrices "matrix.txt" " \t"
  ioda:setup
  reset-ticks
end

to go
  ioda:go
  tick
end
```

Example (3) - matrices

- defined in a CSV file (arbitrary separators)
- here: a file "`matrix.txt`"

```
;source       interaction       priority    target    distance
termites      MoveRandomly      0
termites      MoveRandomly      10      chips    1
termites      PutDown           20      chips    0.3
termites      PickUp            30      chips    1
```

Example (4) – interactions

defined in a text file (here: "interactions.txt")

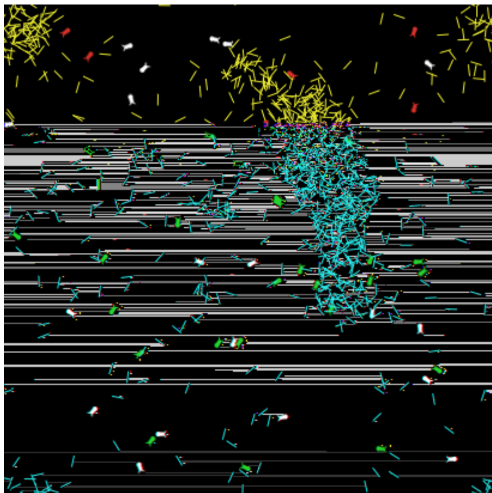
```
interaction MoveRandomly
  actions    wiggle
end

interaction PickUp
  condition          not-carrying?
  actions            take-load get-away
end

interaction PutDown
  condition          carrying?
  actions            drop-load random-turn get-away
end
```

Example (5) – primitives

```
to-report termites::not-carrying?
  report not carrying?
end
to-report termites::carrying?
  report carrying?
end
to termites::filter-neighbors
  ioda:filter-neighbors-in-radius 1
end
to termites::take-load
  set carrying? true
  ask ioda:my-target [ioda:die]
end
to termites::wiggle
  left random 50 right random 50 fd 1
end
to termites::drop-load
  set carrying? false
  hatch-chips 1 [init-chip]
end
to termites::get-away
  fd 20
end
to termites::random-turn
  right random 360
end
```

- each abstract primitive must be concretely defined for the agents family that are likely to use it
- `filter-neighbors` primitive for handling the perception of neighbors
- predefined primitives e.g. `ioda:die`

Example (6) – outcome



— III —

Let's code!

Using the extension

http://www.lifl.fr/SMAC/projects/ioda/SSC2014