Why Interactions Matter
And how to use them in Agent-Based Simulations

P. Mathieu
philippe.mathieu@lifl.fr
S. Picault, Y. Kubera, F. Gaillard, D. Panzoli

Université Lille 1 – Sciences et Technologies
SMAC research team – LIFL – CNRS-UMR 8022

Invited Talk
The advantages of individual-based simulations

- simulation entities and behaviors are designed by analogy with those of the target problem = improved knowledge acquisition
- simulation results are more detailed than in equational models e.g. individual behavior analysis, discrete entities...
- not only predictive, but also explicative models
- individual models can easily be turned to population-based models
Individual-based simulations

Classical approaches

- the design focusses on agents
- behaviors are aimed at specific targets
- behaviors have no software existence separated from agents

Consequences:

- lack of genericity in behavior design
- mixture between declarative and procedural features (behavior description / action selection)
- strong coupling between entities (structure of the system) and behavior (function)
- implicit implementation choices that can lead to biased simulation results
Towards a non ambiguous, domain independant framework

Our goal

Provide a generic and domain independent simulation methodology and framework

This requires:

- the identification of all functional units underlying the architecture of any simulation
- the identification of implementation choices for each unit
- a fine setting of these implementation choices as explicit parameters of the architecture
Individual-based simulations

Short example

A simple Netlogo simulation

to go
  ask turtles [go-turtle]
end
to go-turtle
  ifelse any? other turtles-here [ hatch 1 [ forward 1 ] ]
  right random 60
  left random 60
  forward 1
end

3 implicit modules:
agent activation process

definition of the behaviors

Reproduction

Random walk

action selection process
Individual-based simulations

Critical issues

- any simulation involves (at least) 3 functions:
  1. the definition of the behavior of agents
  2. an action selection process in each agent
  3. the agent activation process

- usually tangled in the code
- impediment to model validation and revision

→ the code should make a clear separation between them:
  1. software separation between behaviors and agents
  2. a generic behavior selection engine
  3. a generic agent activation mechanism
Towards Interaction

From behaviors to interactions

The actual behavior of agents running in a MAS is a complex combination:

- **general interaction abilities between agents**
  → interaction library

- **specific perception/action abilities in agents**
  → agents with primitives

- **simulation engine** → makes agents choose and perform interactions through their own capabilities
  → interaction selection process (generic)

Separation is crucial!  

- simplified design
- bias avoidance (implementation)  

[Kubera & al., 2009]
Towards Interaction

From behaviors to interactions

The actual behavior of agents running in a MAS is a complex combination:

- **general interaction abilities between agents**
  → interaction library

- **specific perception/action abilities in agents**
  → agents with primitives

- **simulation engine** → makes agents choose and perform interactions through their own capabilities
  → interaction selection process (generic)

Separation is crucial!

- simplified design
- bias avoidance (implementation)

[Kubera & al., 2009]
Towards Interaction

From behaviors to interactions

The actual behavior of agents running in a MAS is a complex combination:

- **general interaction abilities between agents**
  → interaction library

- **specific perception/action abilities in agents**
  → agents with primitives

- **simulation engine** → makes agents choose and perform interactions through their own capabilities
  → interaction selection process (generic)

Separation is crucial! [Kubera & al., 2009]

- simplified design
- bias avoidance (implementation)
Our approach

- focus on interactions rather on agents
- software reification of interactions
- complete process from analysis and design to implementation and result analysis

Four layers:

- **IODA** Interaction-Oriented Design of Agent simulations
  - **methodology and modelling**
- **JEDI** Java Environment for the Design of agent Interactions
  - **highly adjustable platform**
- **JEDI-Builder** Code generator
  - IODA model → JEDI simulation
- **LEIA** Simulation space explorer
The IODA approach

3 key ideas

- Every relevant entity is an agent, i.e. an entity endowed with interaction capacities.
- Interactions are more relevant than only agents.
- Interactions must be reified independently.

Definition

An interaction is a coherent sequence of actions between a source agent and target agent(s). It responds to explicit or implicit goals in the simulation and must fulfill logical prerequisites to occur.
A simple interaction: “Eat (X, Y)”

**Trigger:** Implicit motivations/explicit goals intended by the actions

**Conditions:** Logical prerequisites for running actions

**Actions:** List of action primitives to perform

X: source of the interaction

Y: target should be edible?

[Kubera & al., 2008]
Example of an interaction

A simple interaction: “Eat (X, Y)”

X: source
Y: target
of the interaction
Y should be edible?

**Trigger:** Implicit motivations/explicit goals intended by the actions

**Conditions:** Logical prerequisites for running actions

**Actions:** List of action primitives to perform

[Kubera & al., 2008]
Example of an interaction

A simple interaction: “Eat (X, Y)”

Hunger(X) → Has(X,Y)

X.energy++
destroy(Y)

Trigger: Implicit motivations/explicit goals intended by the actions

Conditions: Logical prerequisites for running actions

Actions: List of action primitives to perform

X: source
Y: target
of the interaction
Y should be edible?

[Kubera & al., 2008]
A simple interaction: “Eat (X, Y)”

Trigger: Implicit motivations/explicit goals intended by the actions

Conditions: Logical prerequisites for running actions

Actions: List of action primitives to perform

X: source of the interaction
Y: target
Y should be edible?

[Kubera & al., 2008]
Example of an interaction

A simple interaction: “Eat (X, Y)”

Hunger(X)                      Has(X,Y)
    ↓
X.energy++
       ↓
destroy(Y)

X: source
Y: target
of the interaction

Trigger: Implicit motivations/explicit goals intended by the actions
Conditions: Logical prerequisites for running actions
Actions: List of action primitives to perform

[Kubera & al., 2008]
Example of an interaction

A simple interaction: “Eat (X, Y)”

Trigger: Implicit motivations/explicit goals intended by the actions

Conditions: Logical prerequisites for running actions

Actions: List of action primitives to perform

X: source
Y: target of the interaction

Y should be edible?

[Kubera & al., 2008]
An interaction-based approach

An agent is characterized by the interactions that it can perform or undergo

Mathieu & al. 2001

An interaction $\mathcal{I}$ can occur if and only if $\exists x, y \in \text{Agents} \ (x: \text{source, } y: \text{target})$ so that $x$ can perform $\mathcal{I}$ and $y$ can undergo $\mathcal{I}$. 
An interaction-based approach

An agent is characterized by the interactions that it can perform or undergo

[Mathieu & al. 2001]

An interaction $\mathcal{I}$ can occur if and only if

$\exists x, y \in Agents \ (x: \text{source, } y: \text{target})$

so that $x$ can perform $\mathcal{I}$ and $y$ can undergo $\mathcal{I}$. 
Design of an interaction matrix

The interaction matrix

**Context**: Labyrinth with doors (open, closed or locked), keys, and apples.

<table>
<thead>
<tr>
<th>Sources</th>
<th>Targets</th>
<th>∅</th>
<th>Key</th>
<th>Apple</th>
<th>Dedalus</th>
<th>Door</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unlock (1,0)</td>
</tr>
<tr>
<td>Apple</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dedalus</td>
<td>Die (5,0)</td>
<td>Take (2,1)</td>
<td>Eat (4,0)</td>
<td>Take (3,1)</td>
<td></td>
<td>Open (1,1)</td>
</tr>
<tr>
<td>Door</td>
<td>Walk (0,0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Example of an interaction matrix*

IODA also defines an **update matrix** for changing the state of agents before performing or undergoing regular interactions.
A unified entity representation

Classical multi-agent approaches use different types of entities (agents, objects, artifacts, resources...)

The IODA approach allows a uniform representation of entities:

- each relevant entity is an agent
- the activity of agents is characterized dynamically, through 3 independent abilities:
  - **active agents** act on others or on the environment
    - = non-empty row in the interaction matrix
  - **passive agents** undergo the actions of another entity
    - = non-empty column in the interaction matrix
  - **labile agents** change their own state without acting or undergoing actions
    - = non-empty row in the update matrix
- this characterization is used to optimize the simulation engine
**The simulation engine**

## A reactive action selection process

At each time step, each agent:

1. looks for neighbors (surrounding passive agents)
2. retains only potential targets for interaction it can perform
   i.e. agents that can undergo those interactions
3. evaluates the triggers of available interactions
4. tests their execution conditions
5. selects the realizable interaction with higher priority
6. performs corresponding actions
The simulation engine

A generic simulation engine

The simulation, in a discrete-time scheduler, repeats successive steps:

1. mark all agents as operative (“ready to work”)
2. ask all labile agents to update their state
   i.e. performs realizable interactions from the update matrix
3. ask all operative, active agents to select and perform an interaction according to the interaction matrix
Adjustable knowledge representation level

“kind-of” relationships taken into account in the interaction matrix:

<table>
<thead>
<tr>
<th>Sources</th>
<th>Targets</th>
<th></th>
<th>Grass</th>
<th>Sheep</th>
<th>Goat</th>
<th>Wolf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grass</td>
<td></td>
<td></td>
<td>+ (Breed; 0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Animal</td>
<td></td>
<td></td>
<td>+(Die; 3)</td>
<td>+(Move; 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Herbivore</td>
<td></td>
<td></td>
<td></td>
<td>+(Eat; 2; 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sheep: Animal, Herbivore</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+ (Breed; 1; 1)</td>
<td></td>
</tr>
<tr>
<td>Goat: Animal, Herbivore</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+ (Breed; 1; 1)</td>
</tr>
<tr>
<td>Wolf: Animal</td>
<td></td>
<td></td>
<td>*((Die; 3); 4)</td>
<td>+(Eat; 2; 0)</td>
<td>+(Eat; 3; 0)</td>
<td>+(Breed; 1; 1)</td>
</tr>
</tbody>
</table>

Example of an interaction matrix with inheritance
JEDI features

- exact implementation of the IODA methodology
- each entity is an agent
- each behavior is an interaction
- deterministic algorithms to ensure **reproducibility** in experiments
- specific choices:
  - 2D multiagent simulation platform
  - discrete time
  - sequential evaluation of the agents at each time step
  - multi-scale representation
  - reactive moves
JEDI: principles

- an agent is instance of a class that extends Agent, with:
  - a list of Affectations representing the interactions it can perform, and on which agents
  - a method defining how the neighborhood is computed
  - perception and action primitives (methods) used by the interactions it can perform or undergo

- an interaction is a class extending Interaction, where:
  - 2 methods specify the conditions and trigger
  - 1 method specifies the actions to run
JEDI-Builder

**IODA’s IDE**
- tool for the design of models according to the IODA methodology
- automatic code generation for JEDI
- graphical view of agents and their behaviors through the interaction matrix
Code generator

Given a IODA model, JEDI-Builder automatically generates a code skeleton:

- all Java classes required for the simulation
- the affectation of interactions to agents
- the signature of all action/perception primitives that must be implemented by each agent class
- a default implementation of neighborhood perception in agents
- the simulation mainloop
What remains to write

The developer has still to write everything that is specific to a given experiment:

- the body of action/perception primitives (for new agent classes)
- the condition, trigger and actions of new interaction classes
- the initialization of the simulation (location and state of agents)

... i.e. what defines the domain ontology and the initial conditions
From model to code

JEDI-Builder

- Termites
  - classes
  - src
    - agents
      - interfaces
        - PeutEffectuerPoser.java
        - PeutEffectuerPrendre.java
    - Bois.java
    - Termite.java
- experiences
  - Experience.java
  - ExperienceGUI.java
- interactions
  - Errer.java
  - Poser.java
  - Prendre.java

Readme.txt
Termites.bat
Termites.config
Termites.sh
Termites.xml
The interest of reverse engineering

- forward engineering =
  1. build a model of a phenomenon
  2. implement the model in a simulation
  3. interpret the results

- reverse engineering =
  1. observe simulation results
  2. relate them to implementation
  3. revise the model

Both are helpful to test hypotheses and models!
LEIA lets you Explore Interactions for your Agents
= reverse engineering tool for IODA/JEDI-based simulations

- automatic simulation generator from an ontology of the domain (agents and interactions libraries)
- automatic analysis of interesting simulation results
  - intrinsic criteria (e.g. spatial organization)
  - domain-dependent criteria (e.g. “fitness function”)
- automatic model combinations / transformations
Towards automatic model design

1. define the ontology of the domain = interactions and agents classes

2. LEIA start with an initial interaction matrix (possibly random) = main model

3. variations of the main model are produced through transformation operators

4. resulting simulations are run and watched in parallel by analysis tools

5. a new main model is built
   - one of existing model is chosen
   - or several existing models are combined
LEIA: Principles

Domain ontology

Interactions:
- Kill
- Clone
- ...

Agent families:
- Red
- Blue
- Green
- Yellow

Fill with interactions

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Envir.</th>
<th>Red</th>
<th>Blue</th>
<th>Green</th>
<th>Yellow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>Run</td>
<td></td>
<td></td>
<td></td>
<td>Infect</td>
<td></td>
</tr>
<tr>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Green</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Follow</td>
</tr>
<tr>
<td>Yellow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LEIA: Principles

The diagram illustrates the LEIA process, which involves simplification of model, interesting model, transformation tools, analysis, simulation, and iteration. The process starts with simplification of the model, leading to an interesting model. This interesting model is then transformed using tools, which in turn leads to various models (model 0 to model n). These models are iteratively analyzed and simulated (simulation 0 to simulation n), with the analysis and simulation processes feeding back to refine the model.
Analysis tools

Our aim: to analyse a simulation during runtime

- activity of agents (interactions)
- evolution, density of the population
- evolution of the environment...

Important points:

- full use of the separation between interactions and agents
- works over any JEDI simulation and any ontology from any domain
- collects data from each family of agents and interactions
Transformation tools

Automatic model transformations are performed:

- add/remove interactions randomly
- modifications of priorities or guard distance
- combination of interaction matrices
- initial number of instances for each agent class
- ... + user-defined transformation tools (for domain-specific needs)
LEIA: Overview

LEIA: Principles
An interesting result

An infection model found by LEIA:

<table>
<thead>
<tr>
<th>Sources</th>
<th>Targets</th>
<th>()</th>
<th>Red</th>
<th>Blue</th>
<th>Green</th>
<th>Yellow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>Infect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blue</td>
<td>Infect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Green</td>
<td>Infect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yellow</td>
<td>Infect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Corresponds to the behavior of an “excitable medium”

Cyclic cellular automaton
[Griffeath 93]

Belousov-Zhabotinsky reaction
[Belousov 59, Zhabotinsky 50]
Conclusion on LEIA

- a simulation generator without any code generation
- possible thanks to the IODA approach; takes place in an integrated framework (IODA, JEDI, JEDI-Builder)
- interactive or automatic iterative exploration of the simulation space in order to create or discover new models
- guidelines to improve simulations through intrinsic/domain-dependent metrics
- can be applied to any simulation ontology
Exploration of simulation space

Applications of LEIA

- find new (interesting) models from scratch
- test the robustness of existing models
- test the coherence/relevance of an ontology of domain kind of “brain stimulator”

[Pachet 07]
Client behavior modelling

Supermarket

- real, statistical data → individual behaviors
- easy behavior revision
- global and individual predictions
## Behaviors modeling

### Typical interaction matrix

<table>
<thead>
<tr>
<th>source</th>
<th>target</th>
<th>Client</th>
<th>Article</th>
<th>Checkout</th>
<th>Entrance</th>
<th>Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Ø</td>
<td>+(Talk;2;1;1)</td>
<td>+(Take;4;1;1)</td>
<td>+(MoveTowards;6;1;5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+(SeekCheckout;5)</td>
<td>+(SeekTarget;1)</td>
<td>+(MoveTowards;3;1;3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+(ChooseNextItem;2)</td>
<td>+(MoveToTarget;0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Article</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checkout</td>
<td>+(Open;2)</td>
<td>+(DealWith;1;1;0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+(Close;2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entrance</td>
<td>+(CreateClient;0)</td>
<td>+(Inform;1;1;10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sign</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Behaviors modeling

Example
The FormatStore Project

with Idées-3Com (3D technologies society) and Enaco (business school)

- “Serious Game”: simulation and gameplay issues
- pedagogical simulation platform for vendor training
- interactive simulation (vendor students)
- large-scale simulation (each entity = 1 agent)
- immersion of the player:
  - realistic 3D scene
  - convincing simulated clients with diversified behaviors
  - dynamic environment with unexpected events to handle
Example

see http://www.lifl.fr/SMAC/projects/formatstore
Advantages of IODA/JEDI

The interaction-oriented approach:

- reifies not only agents but also their interactions from design step to implementation step
- builds reusable libraries of interactions:
  - generic, abstract description of behavior
  - interactions are independent from agent ontologies
- represents the functional part of the simulated system and not only the entities (=structure) of the system
- makes explicit several implementation choices thus reduces the risk of simulation biases

see http://www.lifl.fr/SMAC/projects/ioda
Advantages of IODA

Towards plug-and-play simulations

Interactions from libraries are affected to agents, through the interaction matrix

run the generic simulation engine... then watch resulting behaviors!
Conclusion

- flexible methodology for individual-based simulation
  - high abstraction level in behavior modeling
  - arbitrary tuning of cognitive abilities
  - incremental design available
  - uniform handling of all relevant entities
  - easily accessible to non-computer scientists

- integrated development framework from model design (IODA) to simulation execution (JEDI) and result analysis (LEIA), through automatic code generation (JEDI-Builder)

- work in progress: multi-environment modelling

see http://www.lifl.fr/SMAC/projects/ioda
That’s all!

Thank you for your attention!

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