



GAMA

*Gis & Agent-based Modeling Architecture*

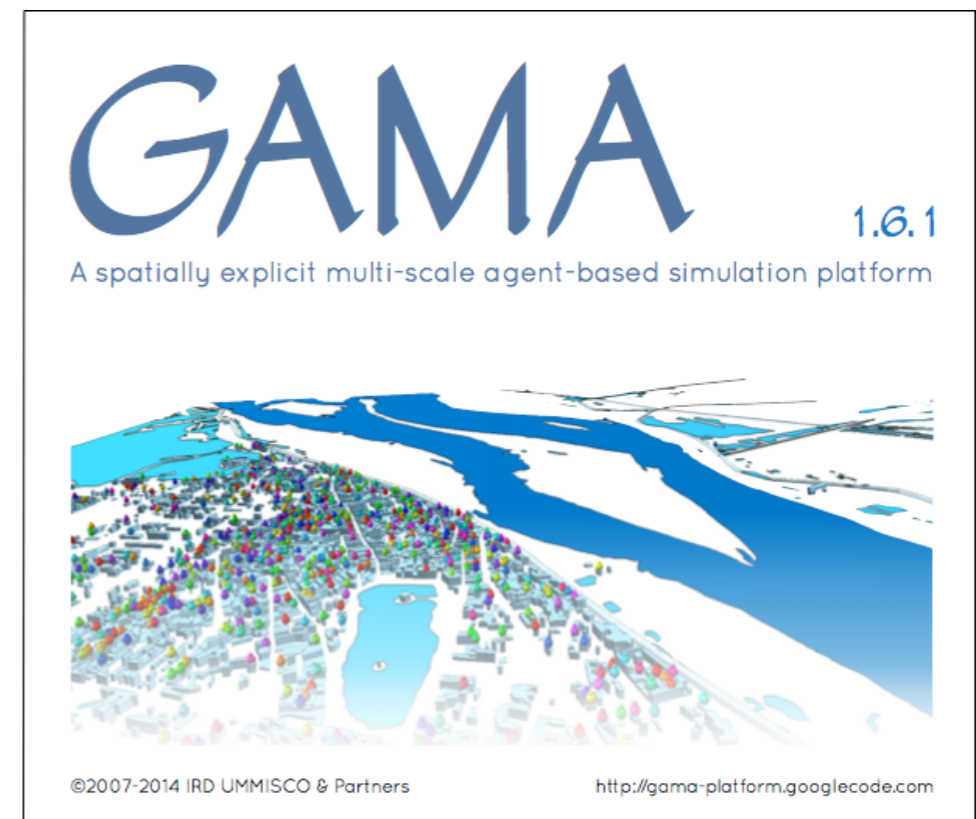
JADE

*JAVA Agent DEvelopment Framework*



# Introduction - GAMA

- Gis & Agent-based Modeling Architecture
- Agent-based, spatially explicit, modeling and simulation platform.
- Free and Open Source tool.
- GAMA webpage



# Introduction - JADE

- JAVA Agent DEvelopment Framework.
- Multi-agent systems through a middle-ware that complies with the Fipa specifications.
- Free and Open Source tool.
- JADE webpage

# Features



# Features

- Initially developed as an Eclipse plug-in, now is an independent tool.
- GAML (Gis & Agent-based Modeling Language) agent-oriented language, close to Java.
- Instantiation of agents from any kind of dataset, including Gis data (e.g.: road traffic model).

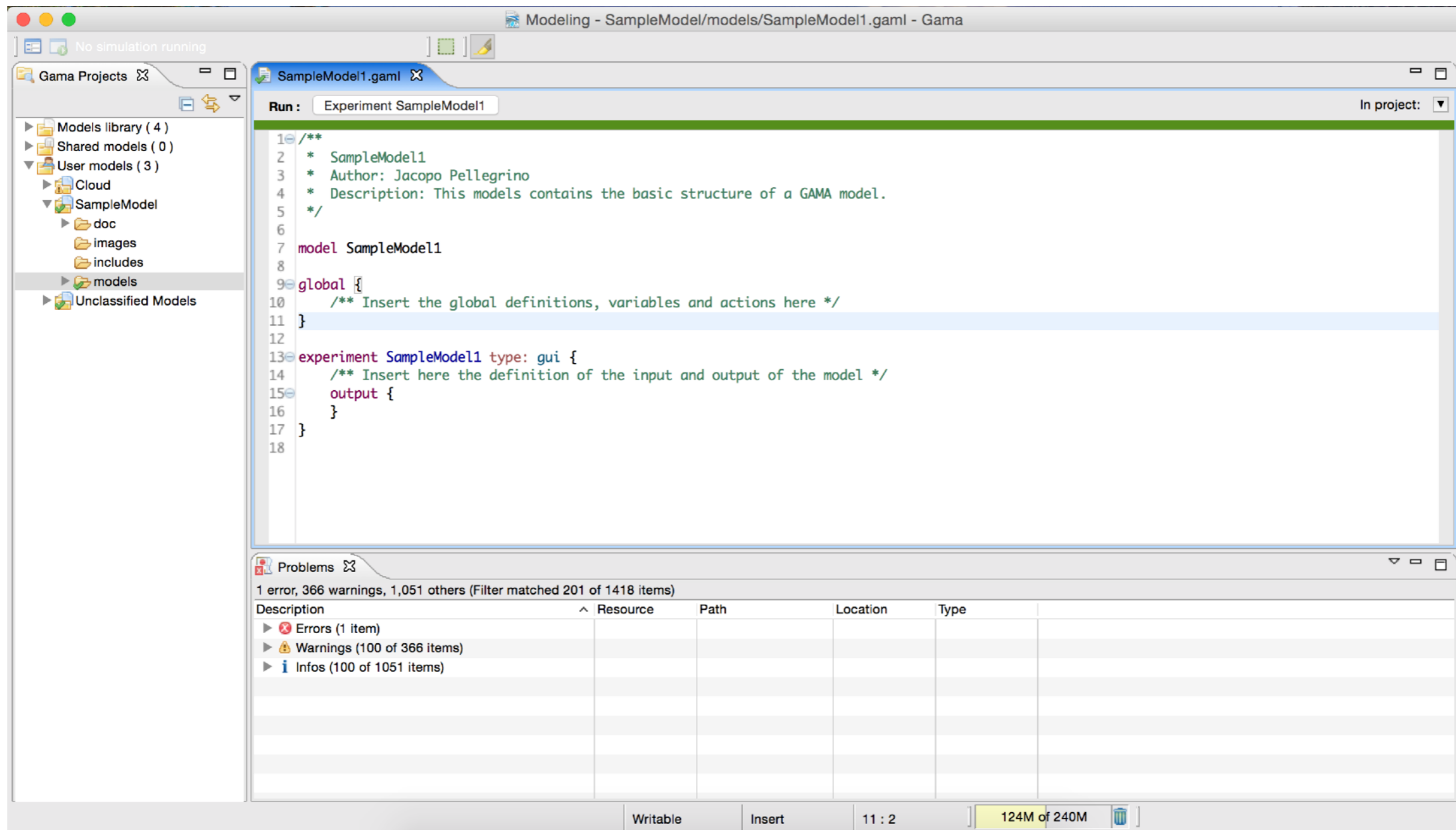
# Features

- Discrete or continuous topological layers.
- Multiple levels of agency: micro and macro species, inheritance.
- Multiple paradigms such as mathematical equations, control architectures, finite state machines.

# Features

- Possibility to define several experiments.
- Development environment with different perspectives.
- User-friendly interface for both development and model simulation.

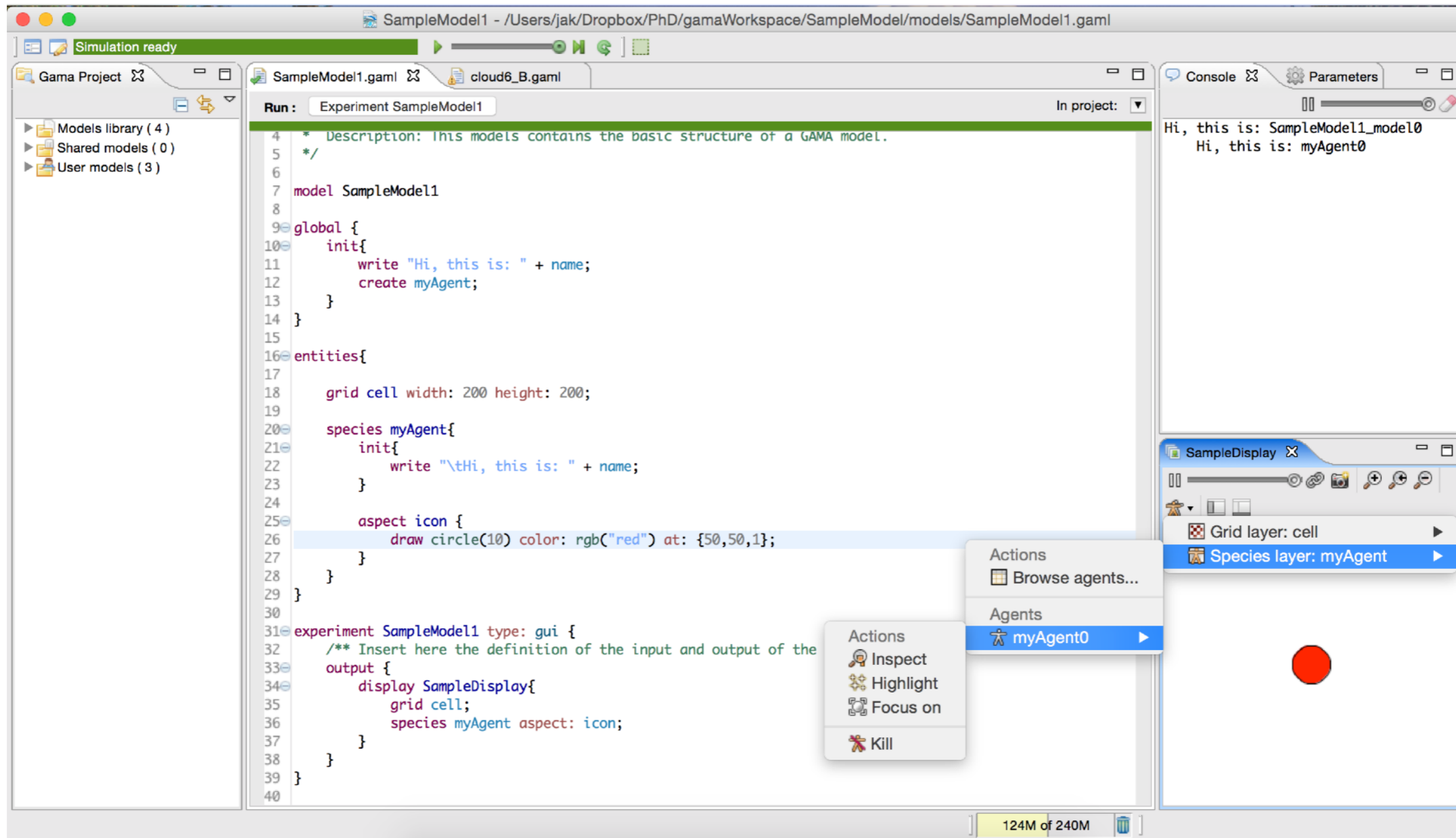
# Features



Modeling perspective



# Features



Simulation perspective

# Modeling



# Modeling Introduction

Model file made up of three main parts:

- global
- entities
- experiment

```
7 model SampleModel1
8
9 global {
10
11 }
12
13 entities{
14
15 }
16
17 experiment SampleModel1 type: gui {
18     output {}
19 }
20
```

# Species Relationship

Species can be related to each other:

- **Nesting:** a species can be defined within another one. The enclosing one is referred as *macro species*, the enclosed one as *micro species*.
- **Inheritance:** a *child* species extends behavior from the *parent*, close to what happens in Java.

# Agent Monitoring

It is possible to monitor agents:

- Agent Browser: browse population of agent species, highlight one, monitor a species.
- Agent Inspector: retrieve information related to one or more specific agent(s), e.g. position, speed, internal variables and the like.

# Agent Monitoring

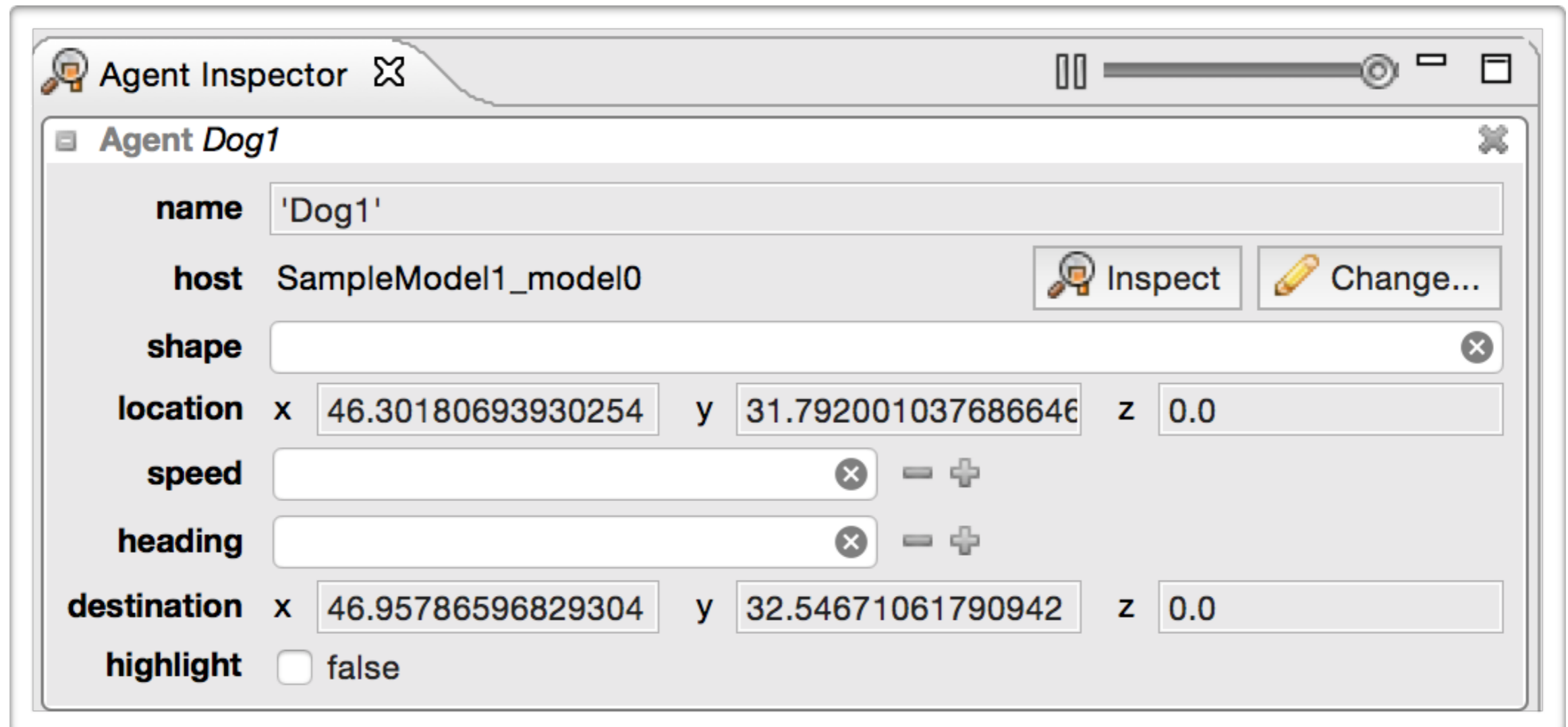
## Agent Browser

The screenshot displays the GAMA software interface for agent monitoring. The main window is titled "SampleModel1" and shows a 3D visualization of a simulation environment. A context menu is open over the 3D window, showing options for "World", "Agents", and "Micro-populations". The "Agents" menu is expanded to show a list of agents from Dog0 to Dog14, with "Dog1" selected. The "Micro-populations" menu is also expanded, showing "Population of Dog" selected. The "Quantities" window shows a pie chart titled "How many?" with a red slice representing "Cats = 10 (40%)" and a blue slice representing "Dogs = 15 (60%)". The code editor at the bottom shows the following code:

```
Run : Experiment SampleModel1  
25 aspect icon {  
26   draw circle(10) color: rgb("red") at: {25,50,0};  
27 }  
28  
29 reflex sayMiaow when: tired{  
30   write "Miaow";  
31   hunger <- hunger +1;  
32 }  
33 }  
34  
35 species Dog skills: {Fighting}
```

# Agent Monitoring

## Agent Inspector



The screenshot shows the 'Agent Inspector' window with the following details for 'Agent Dog1':

- name:** 'Dog1'
- host:** SampleModel1\_model0
- shape:** (empty field)
- location:** x: 46.30180693930254, y: 31.792001037686646, z: 0.0
- speed:** (empty field with minus and plus buttons)
- heading:** (empty field with minus and plus buttons)
- destination:** x: 46.95786596829304, y: 32.54671061790942, z: 0.0
- highlight:**  false

Buttons for 'Inspect' and 'Change...' are visible next to the host field.

# Data input and output

Data I/O:

- Data can be imported and exported in and from the model.
- The project folder allows to gather data to be imported to be accessible to the code.
- Several common formats can be read in: .txt, .csv, .png, etc.

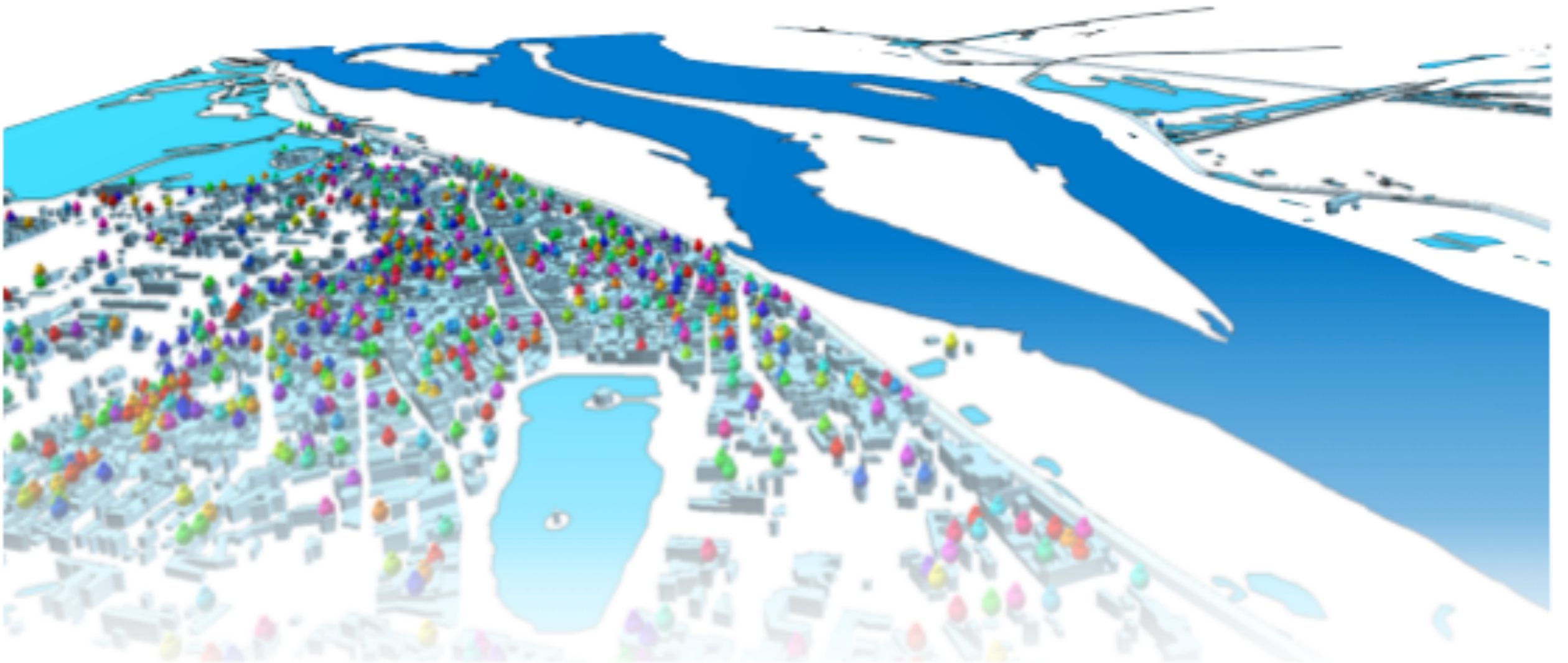


# Errors Detection

Warnings and Errors:

- As most of the common development tools, GAMA notifies the user about bad code or mistakes with warnings and errors (at compile time), e.g.: bad syntax.
- In case of run time errors the simulation stops and a description of the problem is provided, e.g.: null reference, out-of-bound.

# Variables, Actions, Reflexes



# Modeling

In the following we will take a closer look to the implementation of a MAS in GAMA focusing on:

- The GAML language (structures, operators, etc.)
- The display of agents and data
- The FIPA communication protocol

Further information can be found in the documentation.

# GAML - Variables

Variables Access:

- **global**: can be directly accessed by any agent in every part of the model.
- **species variables**: can be directly accessed by the agent of the corresponding species in every part of the model. They can be accessed remotely, by other agents, with the syntax:  
`type varName <- remote_Agent.remoteVariable;`  
`int dogAge <- one_of(Dog).age;`
- **temporary variables**: can only be accessed directly and within the statement block. They stop existing when the statement is completed.

# GAML - Actions

Actions Definition:

- An action embodies a **capability of an agent**, it can take from 0 to many arguments and return 0 or one variable. It is declared as follows:

```
action noArgNoReturn{
}
action noArgReturn{
  return returnVar;
}
action argNoReturn(type1 arg1, type2 arg2){
}
```

- It is possible to assign a return variable directly to a variables as follows:

```
int myVar <- argReturn(arg1::val1, arg2::"val2");
```

# GAML - Reflexes

Reflexes Definition:

- A reflex can be considered as an action that the **agent automatically performs** at any time step or when a given condition occurs. In reflexes action can be called. A reflex is defined as follows:

```
reflex everyTime{  
    //is executed at every time step  
}
```

```
reflex someTimes when: booleanExpression{  
    //is executed only when the boolean expression is true  
}
```

- The `init` is a special kind of reflex that is executed when the agent is created.

# GAML - Control Structures

- Actions and Reflexes have been defined. Now how to tell agents what to do?
- GAMA provides the most common control structures to **control the flow of execution** of the code. The most used are:
  - Loop Statements
  - Conditional Statements

# Graphical Environment





# GAML - Graphics

- A graphical representation can be useful in several modeling scenarios.
- GAMA allows the display of agents within an environment referred as the grid.
- Layers of agents can be displayed separately.
- The output of the experiments can be displayed too.

# GAML - Aspects

Aspects Definition:

- The aspect defines the way agents will be displayed. Each species can have more than one aspects, in the experiment the user indicates which one will be used for the display. Aspects are defined as follows:

```
aspect aspectName {  
    draw shape color: rgb("aRGBColor") at: {position};  
}  
aspect default {  
    draw circle(5) color: rgb("red") at: {25,50,0};  
}
```

- As indicated, it is possible to add facets like the color, the position and the like. It is also possible to use .jpg, .gif, or .png image as icon for the agent.

# GAML - Grid

## Grid Definition:

- The grid can be considered a particular set of agents that share a topology. These agents are automatically created and are mainly used for the implementation of the environment where the other species reside.

```
grid cell width: xSize height: ySize neighbors: neighNb;
```

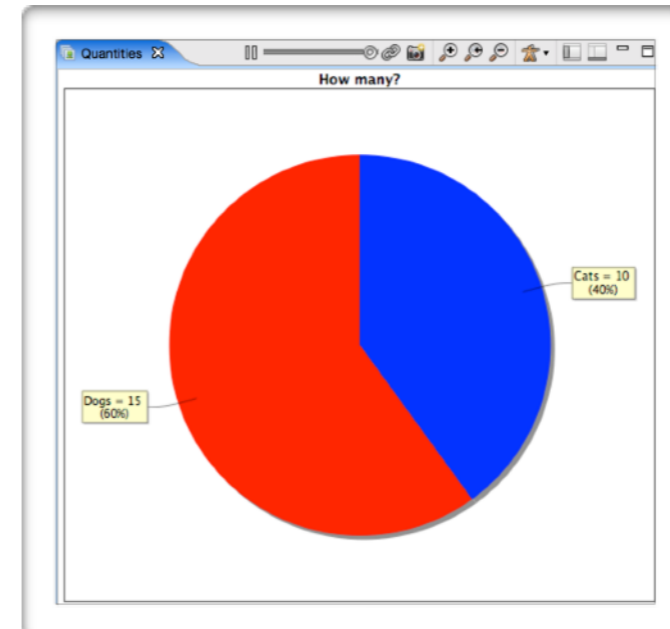
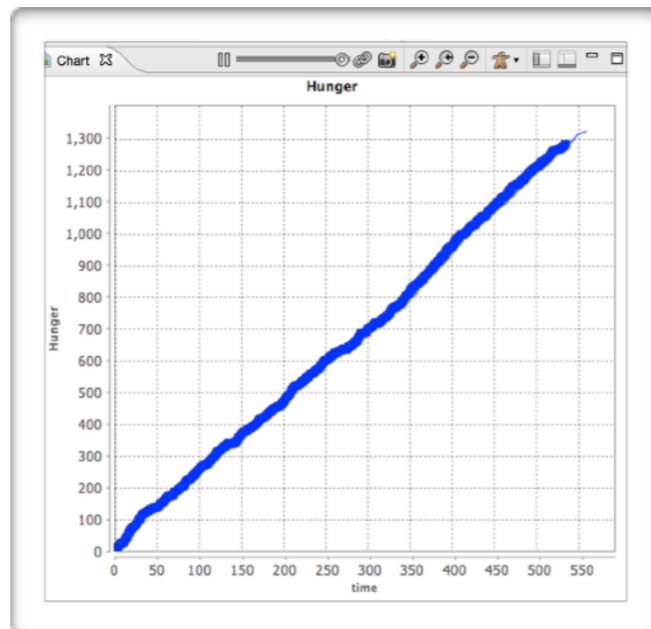
- It is also possible to create the grid from a given file such as a GIS file, see the Road Traffic example.

# GAML - Experiments

## Display Examples:

```
display Chart{  
  chart "Hunger" type: series position: {0.0, 0.0} background: rgb("white") size: {1.0, 1.0}{  
    data "Hunger" value: hunger color: rgb('blue');  
  }  
}
```

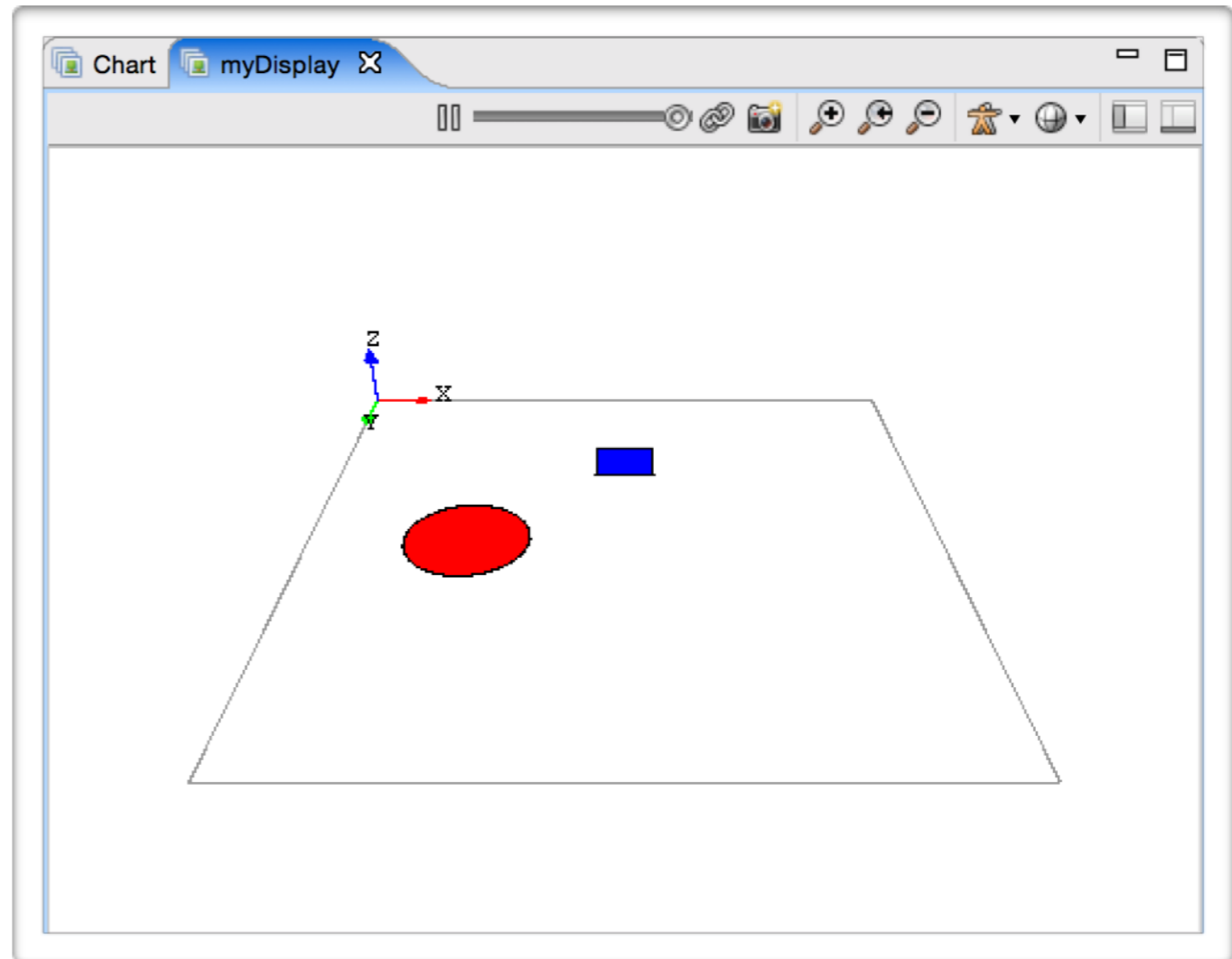
```
display Quantities{  
  chart "How many?" type: pie position: {0.0, 0.0} background: rgb("white") size: {1.0, 1.0}{  
    data "Cats" value: length(Cat) color: rgb('blue');  
    data "Dogs" value: length(Dog) color: rgb('red');  
  }  
}
```



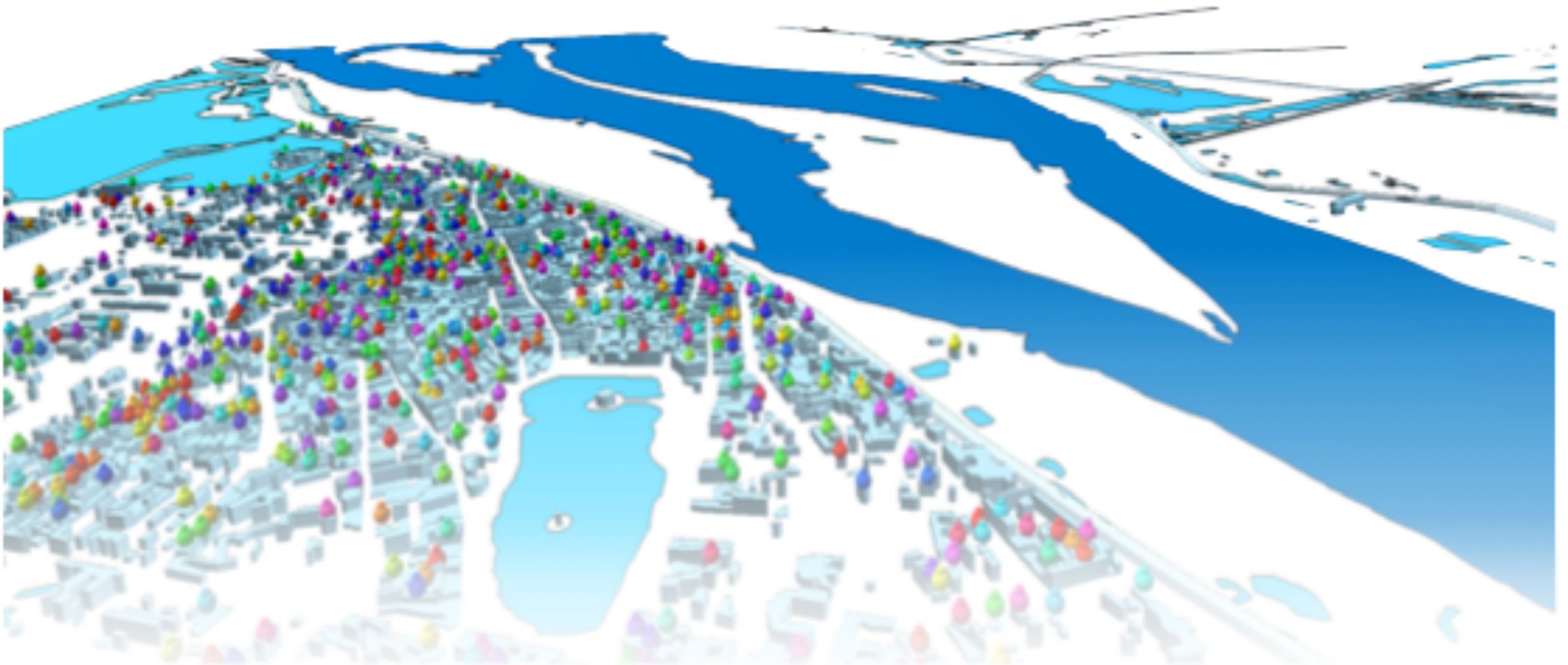
# GAML - Experiments

Display Examples:

```
display myDisplay type:opengl {  
  grid cell;  
  species Cat aspect: icon;  
  species Dog aspect: icon;  
}
```



# Communication



# Communication

- The communication has to be guaranteed by **standards**.
- Agents involved have to be compliant to the standard.
- Definition of **ACL** (Agent Communication Languages) with an explicit, general and well-defined semantics.
- Ability of software systems of **exchanging information** and of **automatically interpreting** its meaning.

# GAML - Communication

- In GAMA the communication is based upon the FIPA Agent Communication Language.
- FIPA messages are labeled with a **performative** that specifies the type of message in terms of purpose.
- Thanks to the performatives it is possible to build **interaction protocols** (patterns of behavior).



# GAML - Communication

List of FIPA performatives:

performative	passing info	requesting info	negotiation	performing actions	error handling
accept-proposal			x		
agree				x	
cancel		x		x	
cfp			x		
confirm	x				
disconfirm	x				
failure					x
inform	x				
inform-if	x				
inform-ref	x				
not-understood					x
propose			x		
query-if		x			
query-ref		x			
refuse				x	
reject-proposal			x		
request				x	
request-when				x	
request-whenever				x	
subscribe		x			

# Communication



# Prey / Predator Model

- The prey / predator model is provided in GAMA as tutorial.
- There are several models with increasing complexity to let the beginner understand the features of GAMA and the GAML syntax.
- In the following the model will be explained and analyzed in detail.

# Prey / Predator Model

The aim of this model is to simulate a natural environment in which two species of animals coexist.

- The environment is made up of a grid of cells representing the soil with grass.
- Preys look around for grass to eat.
- Predators look around for preys to eat.

# Features

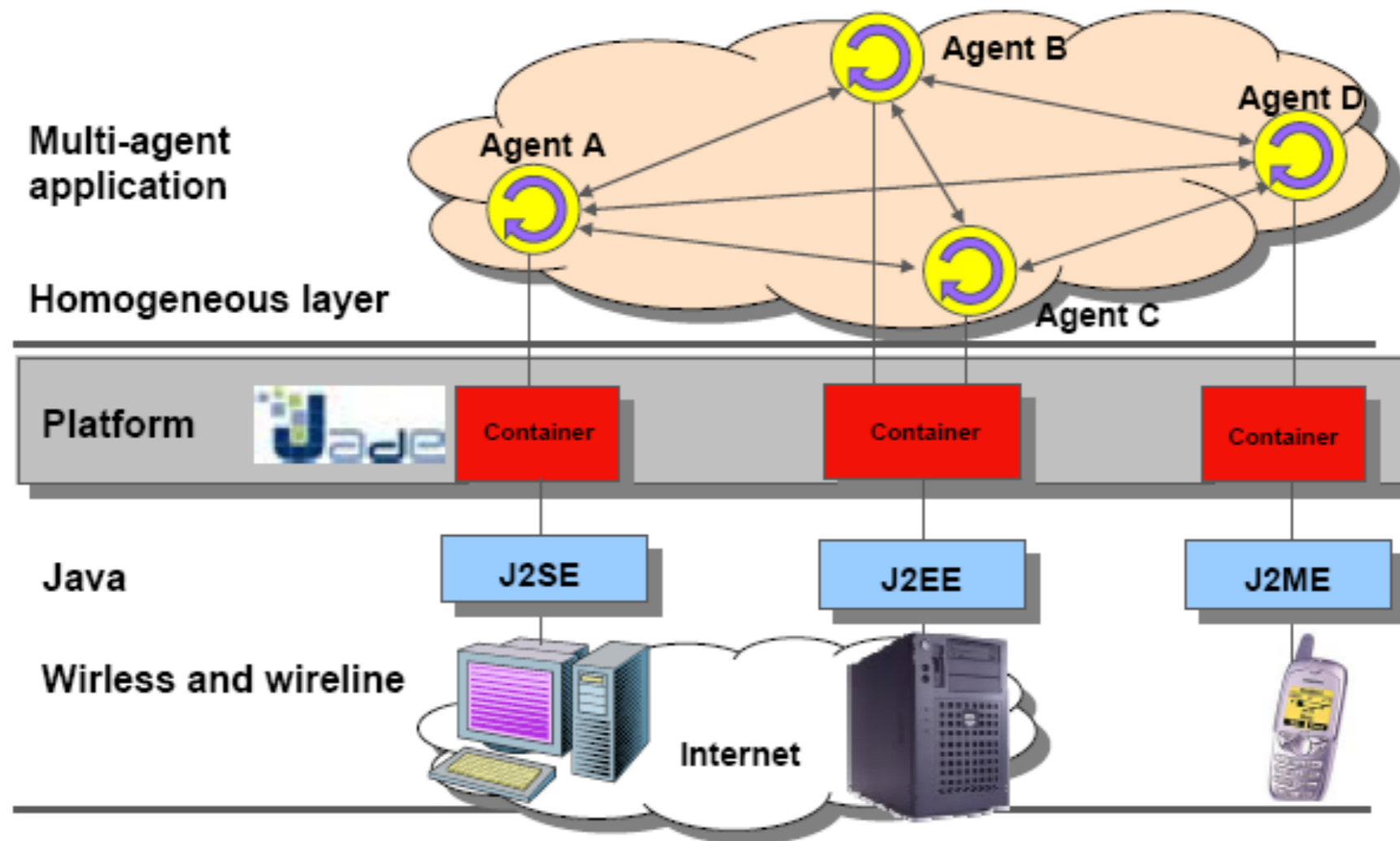


# Features

- Flexible and efficient messaging.
- Java Language.
- Agents are implemented as one thread per agent.
- Graphical User Interface (GUI).
- Can be distributed across machines.

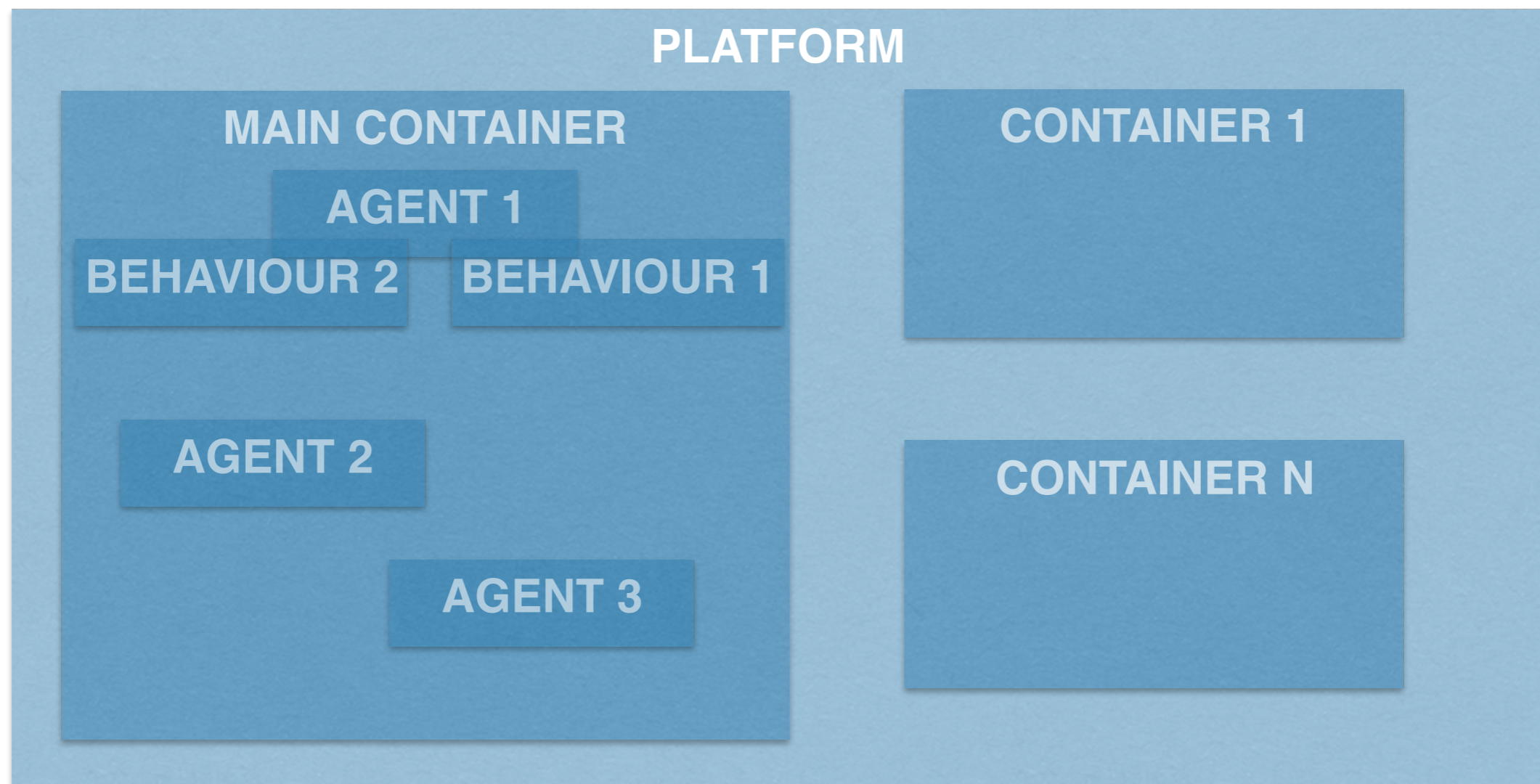
# Features

- Can be distributed across machines.



# Structure of JADE

The platform is made up of:





# Structure of JADE

The platform is made up of:

- a **main container**.
- other (remote) containers.
- each agent is a **peer** living in its container.

# Structure of JADE

The **main container** contains two “special agents”:

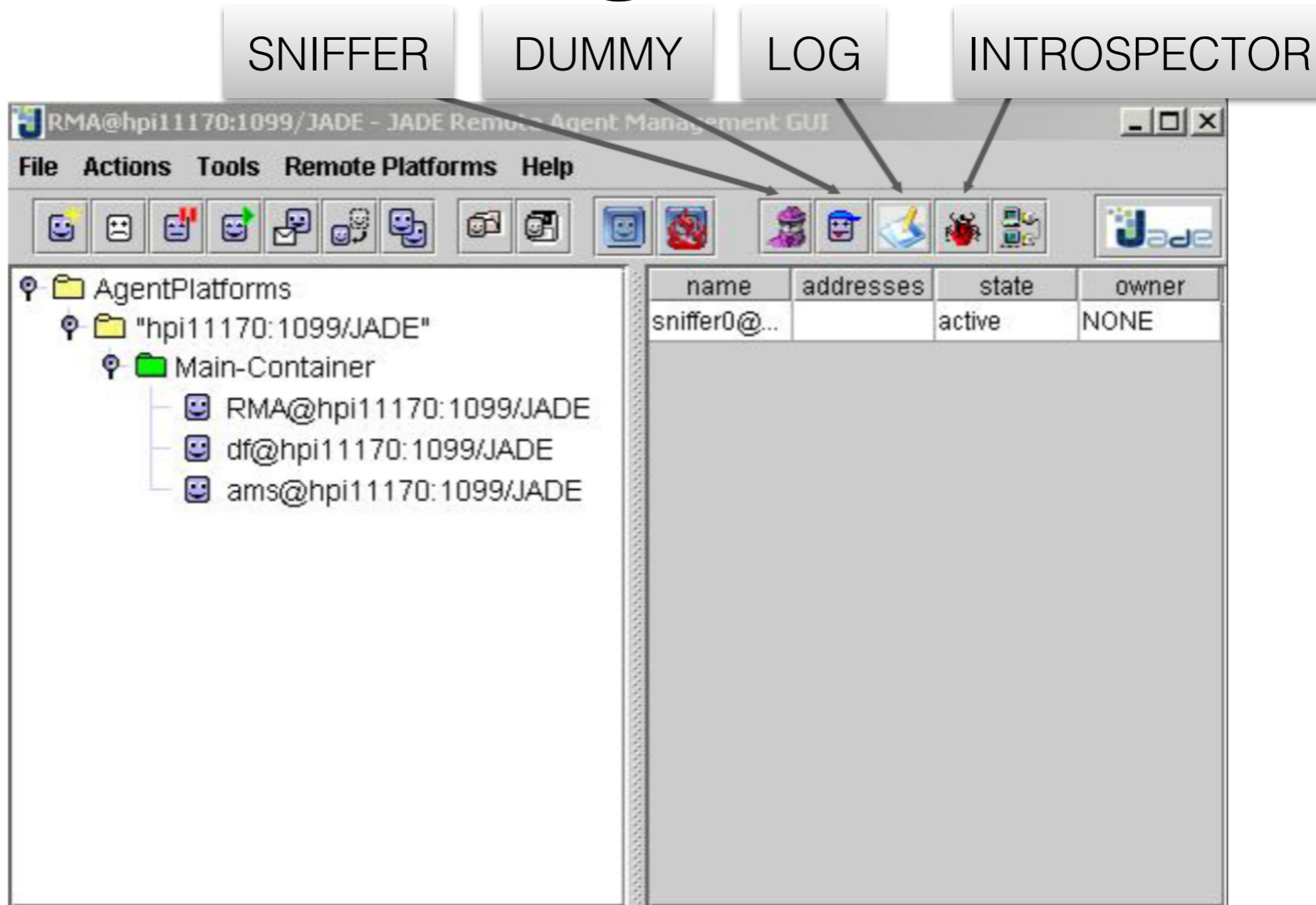
- The **AMS** (Agent Management System):
  - provides the naming service (unique names).
  - represents the authority in the platform (create / delete).
- The **DF** (Directory Facilitator)
  - provides a Yellow Pages service (agent/service).

# Structure of JADE

Other “special” agents are provided by default:

- RMA (Remote Monitoring Agent).
- Dummy Agent.
- Sniffer Agent.
- Introspector Agent.
- Log Manager Agent.
- DF (Directory Facilitator) GUI.

# Starting JADE



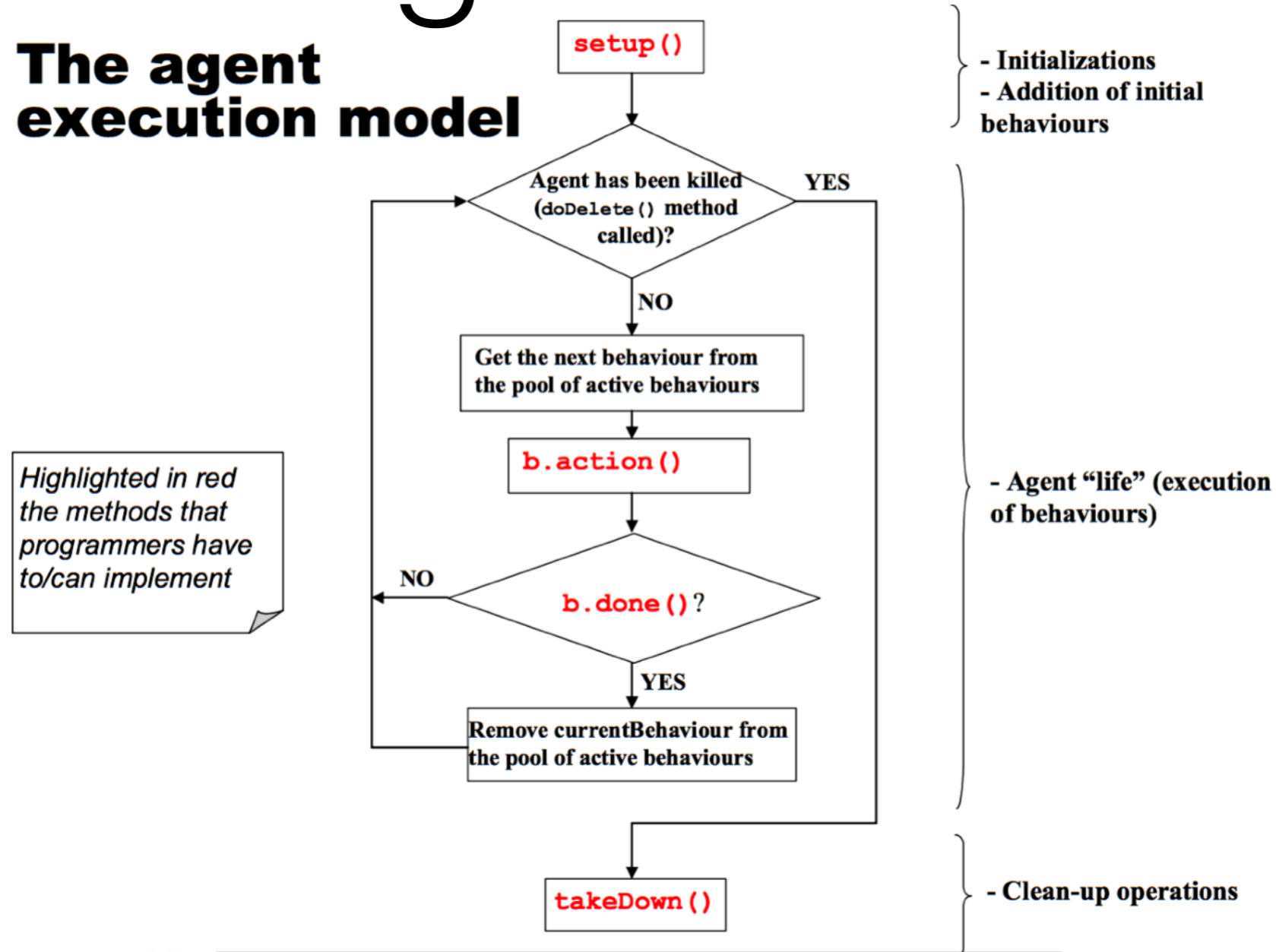
*Screenshot of JADE gui.*

# Modeling



# Modeling Introduction

## The agent execution model



Slide from JADE Tutorial for beginners

# Species Relationship

Species can be related to each other:

- **Inheritance**: a *child* species extends behavior from the *parent*, as happens in Java.
  - All agents inherit from `jade.core.Agent`.
- **NO nesting**: all agents live inside the container at the same level.
  - Each agent can create other agents.

# Data input and output

Data I/O:

- Common Java I/O libraries.
- It is possible to make agents write to a **database**.
- It is a good idea to initialize the model using a **configuration file** (csv, xml, ...).



# Errors Detection

Warnings and Errors:

- Running JADE from the shell errors/exceptions will be prompted and agents may die.
- A good practice is to use an IDE (Eclipse, NetBeans) which helps a lot.

# JAVA - Control Structures

- Behaviours have been defined. Now how to tell agents what to do?
- JAVA provides the most common control structures to **control the flow of execution** of the code. The most used are:
  - Loop Statements.
  - Conditional Statements.
- Keep code in methods.

# Graphics and Communication



# JADE - Graphics

- JADE does not provide a graphical environment like GAMA does.
- Users may create their own with external tools.
- JADE is focused on the communication rather than graphics.

# JADE - Communication

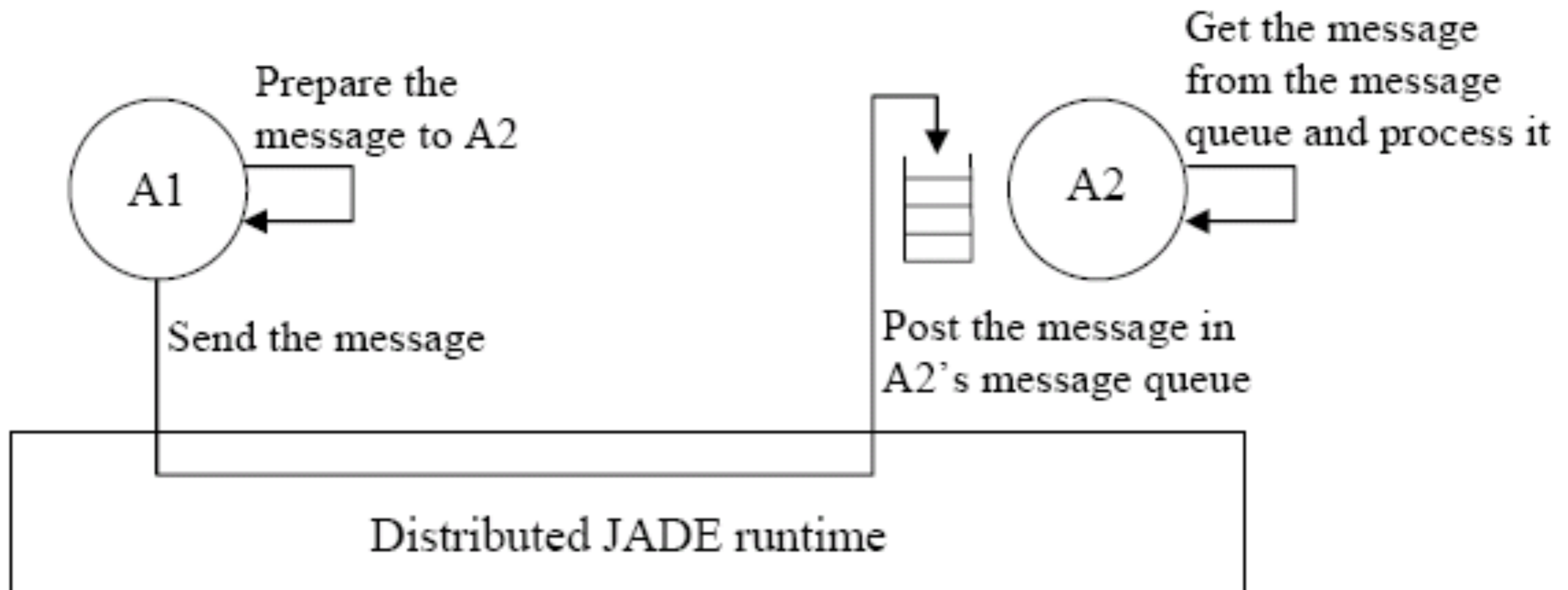
- In JADE, as in GAMA the communication is based upon the FIPA Agent Communication Language.
- FIPA messages are labeled with a **performative** that specifies the type of message in terms of purpose.
- Thanks to the performatives it is possible to build **interaction protocols** (patterns of behavior).

# JADE - Communication

- **Asynchronous message passing**
- When a message is sent the platform takes care of putting it in the **queue** of the receiver agent.
- There are method to easily reply.
- Pay attention to blocking/non-blocking receive.

# JADE - Communication

What happens behind the scenes:



# Thanks for your attention

