SARL and Janus: State of the works and Perspectives

EuSarlCon 2019 - May 2nd 2019

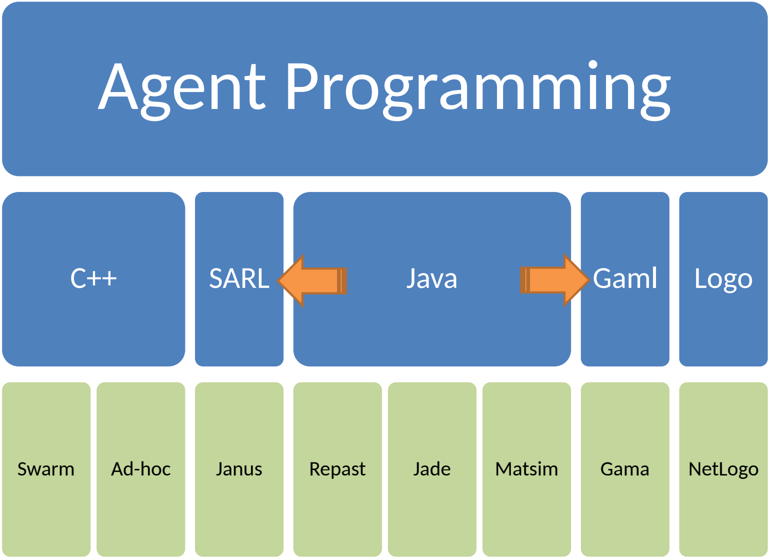
Prof.Dr. Stéphane GALLAND

Université de Technologie de Belfort Montbéliard - UTBM
90010 Belfort cedex, France - http://www.multiagent.fr

OUTLINE

UBFC
UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

OUTLINE



UBFC
UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

DESIGN PRINCIPLES OF SARL

Language

- **All agents are holonic (recursive agents).**
- There is not only one way of interacting but infinite.
- Event-driven interactions as the default interaction mode.
- Agent/environment architecture-independent.
- Massively parallel.
- Coding should be simple and fun.

Execution Platform

- Clear separation between Language and Platform related aspects.
- Everything is distributed, and it should be transparent.
- Platform-independent.

COMPARING SARL TO OTHER FRAMEWORKS

Name	Domain	Hierar.[a]	Simu.[b]	C.Phys.[c]	Lang.	Beginners[d]	Free
GAMA	Spatial simulations		✓		GAML, Java	**[*]	✓
Jade	General		✓	✓	Java	*	✓
Jason	General		✓	✓	Agent-Speaks	*	✓
Madkit	General		✓		Java	**	✓
NetLogo	Social/ natural sciences		✓		Logo	***	✓
Repast	Social/ natural sciences		✓		Java, Python, .Net	**	
SARL	General	✓	✓[e]	✓	SARL, Java, Xtend, Python	**[*]	✓

a Native support of hierarchies of agents.

b Could be used for agent-based simulation.

c Could be used for cyber-physical systems, or ambient systems.

d *: experienced developers; **: for Computer Science Students; ***: for others beginners.

e Ready-to-use Library: Jaak Simulation Library

This table was done according to experiments with my students.

UBFC UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

OVERVIEW OF SARL CONCEPTS

Multiagent System in SARL

A collection of agents interacting together in a collection of shared distributed spaces.

4 main concepts

- Agent
- Capacity
- Skill
- Space

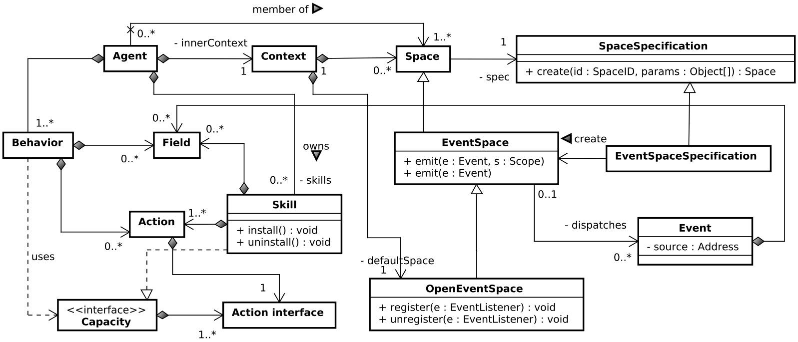
3 main dimensions

- Individual:: the Agent abstraction (Agent, Capacity, Skill)
- Collective:: the Interaction abstraction (Space, Event, etc.)
- Hierarchical:: the Holon abstraction (Context)

SARL: a general-purpose agent-oriented programming language. Rodriguez, S., Gaud, N., Galland, S. (2014) Presented at the The 2014 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IEEE Computer Society Press, Warsaw, Poland. (Rodriguez, 2014)

`http://www.sarl.io`

SARL METAMODEL

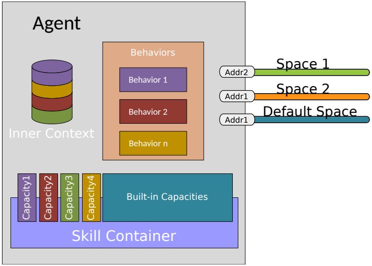


UBFC
UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

AGENT

Agent

- An agent is an autonomous entity having some intrinsic skills to implement the capacities it exhibits.
- An agent initially owns native capacities called Built-in Capacities.
- An agent defines a Context.




```
agent HelloAgent {
  on Initialize {
    println("Hello World!")
  }
  on Destroy {
    println("Goodbye World!"
)
  }
}
```

CAPACITY AND SKILL

Action

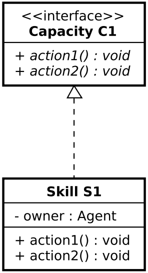
- A specification of a transformation of a part of the designed system or its environment.
- Guarantees resulting properties if the system before the transformation satisfies a set of constraints.
- Defined in terms of pre- and post-conditions.

Capacity

Specification of a collection of actions.

Skill

A possible implementation of a capacity fulfilling all the constraints of its specification, the capacity.



Enable the separation between a generic behavior and agent-specific capabilities.

Space

Support of interaction between agents respecting the rules defined in various Space Specifications.

Space Specification

- Defines the rules (including action and perception) for interacting within a given set of Spaces respecting this specification.
- Defines the way agents are addressed and perceived by other agents in the same space.
- A way for implementing new interaction means.

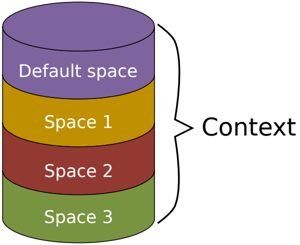
.

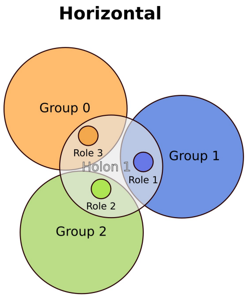
The spaces and space specifications must be written with the Java programming language

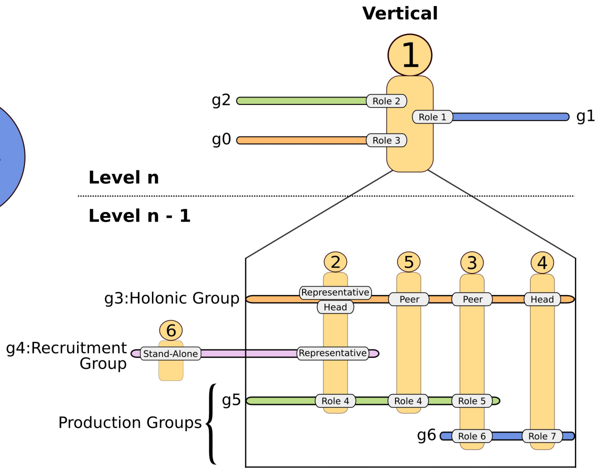
CONTEXT AND SPACES

Context

- Defines the boundary of a sub-system.
- Collection of Spaces.
- Every Context has a Default Space.
- Every Agent has a Default Context, the context where it was spawned.

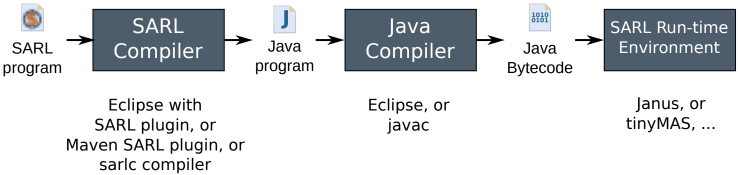



Horizontal
Group 0
Group 1
Group 2
Role 3
Holon 1
Role 1
Role 2


Vertical
1
g2
Role 2
Role 1
g1
g0
Role 3
Level n
Level n - 1
2
5
3
4
g3:Holonic Group
Representative
Head
Peer
Peer
Head
6
g4:Recruitment Group
Stand-Alone
Representative
g5
Role 4
Role 4
Role 5
Production Groups
g6
Role 6
Role 7

COMPATIBILITY BETWEEN SARL AND JAVA

SARL is 100% compatible with Java



- Any Java feature or library could be included and called from SARL.
- A Java application could call any public feature from the SARL API.

Runtime Environment Requirements

- Implements SARL concepts.
- Provides Built-in Capacities.
- Handles Agent's Lifecycle.
- Handles resources.

Janus as a SARL Runtime Environment

- Fully distributed.
- Dynamic discovery of Kernels.
- Automatic synchronization of kernels' data (easy recovery).
- Micro-Kernel implementation.
- Official website: `http://www.janusproject.io`

Other SREs may be defined.

METAMODEL AND LANGUAGE EVOLUTION 1/2

Action Selection Architecture ≈ BDI

- BDI-like framework: Goals, Believes, Actions.
- Definition of statements.

Environment Metamodel ≈ Artifact

- Artifact-like framework: artifact, use interaction, endogeneous dynamics.
- Definition of specific statements.

Organizational Modeling ≈ CRIO

- Definition of the mapping between the CRIO concepts and the SARL concepts.
- Definition of statements for roles and interaction definitions.

METAMODEL AND LANGUAGE EVOLUTION 2/2

Action Selection Architecture $\approx$ BDI

- BDI-like framework: Goals, Believes, Actions.
- Definition of statements.

Design by contract with SARL

- Formal properties into the SARL concepts: invariant, post-, pre-conditions.
- Formal properties for interaction protocols.
- Enforcement of the property validation during run-time.

UBFC UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

SIMULATION FRAMEWORK

Time Management

- Management of the simulation time by the run-time framework.
- Management of the simulation time over a network of computers.

Agent Environment

- Definition of tools for defining the agent environment: artifacts, smart objects...
- Addition of modules for agent-based simulation of drones, road traffic, crowd, autonomous cars, IOT...

User Interface

- UI tools for simulators, like Netlogo of Gama.

Janus for Embedded Systems

- Real-time implementation of Janus for embedded systems.

New Run-time Environments

- Akka for creating a new SARL run-time Environment dedicated to Cloud computing.
- GAMA for running SARL agents.
- Extending MATSIM with SARL capabilities.

Thank you for your attention…

Appendix

IMPLICIT CALLS TO GETTER AND SETTER FUNCTIONS

- Calling getter and setter functions is verbose and annoying.
- Syntax for field getting and setting is better.
- SARL compiler implicitly calls the getter/setter functions when field syntax is used.

```
class Example {
private var a : int

def getA : int {
this.a
}
def setA(a : int) {
this.a = a
}
}

class Caller {
def function(in : Example) {
// Annoying calls
in.setA(in.getA + 1)
// Implicit calls by SARL
in.a = in.a + 1
}
}
```

- With call: `variable.field`; SARL seach for:
  1. the function `getField` defined in the variable's type,
  2. the accessible field `field`.
- If the previous syntax is left operand of assignment operator, SARL seach for:
  1. the function `setField` defined in the variable's type,
  2. the accessible field `field`.

- Goal: Extension of existing types with new methods.
- Tool: Extension methods.
- Principe: The first argument could be externalized prior to the function name.

- Standard notation: `function(value1, value2, value3)`

- Extension method notation: `value1.function(value2, value3)`

```
class Example {

// Compute the Leivenstein
// distance between two
// strings of characters
def distance(s1 : String,
s2 : String)
: int {
// Code
}

def standardNotation {
var d = distance("abc", "abz
")
}

def extensionMethodNotation
{
var d = "abc".distance("abz"
)
}
}
```

- Lambda expression: a piece of code, which is wrapped in an object to pass it around.
- Notation:
  `[ paramName : paramType, ... | code ]`
- Parameters' names may be not typed. If single parameter, `it` is used as name.
- Parameters' types may be not typed. They are infered by the SARL compiler.

```
class Example {
def example1 {
var lambda1 = [
a : int , b : String |
a + b.length ]
}

def example2 {
var lambda2 = [ it.length ]
}
}
```

TYPE FOR A LAMBDA EXPRESSION

- Type for a lambda expression may be written with a SARL approach, or a Java approach.
- Let the example of a lambda expression with:
  - two parameters, one int, one String, and
  - a returned value of type int.

```
class Example {
def example1 :
(int , String) => String {
return [
a : int , b : String |
a + b.length ]
}

def example2 :
Function2<Integer , String ,
Integer> {
return [
a : int , b : String |
a + b.length ]
}
}
```

- SARL notation: `(int, String) => int`
- Java notation: `Function2<Integer, String, Integer>`

- Problem: Giving a lambda expression as function's argument is not friendly (see `example1`).
- Goal: Allow a nicer syntax.

- Principle: If the last parameter is a lambda expression, it may be externalized after the function's arguments (see `example2`).

```
class Example {

def myfct(a : int , b :
String ,
c : (int) => int) {
// Code
}

def example1 {
myfct(1, "abc", [ it * 2 ])
}

def example2 {
myfct(1, "abc") [ it * 2 ]
}
}
```

SPECIAL INSTANCE VARIABLES

- Usually, the OO languages provide special instance variables.
- SARL provides:
  - this: the instance of current type declaration (class, agent, behavior...)
  - super: the instance of the inherited type declaration.
  - it: an object that depends on the code context.

```
class Example extends
SuperType {

 var field : int

 def thisExample {
 this.field = 1
 }

 def superExample {
 super.myfct
 }

 def itExample_failure {
 // it is unknown in this
 // context
 it.field
 }

 def itExample_inLambda {
 // it means: current
parameter
 lambdaConsumer [ it + 1 ]
 }

 def lambdaConsumer((int) =>
int)
 {}
 }
```

- Type: Explicit naming a type may be done with the optional operator:
  typeof(TYPE).
- Casting: Dynamic change of the type of a variable is done with operator:
  VARIABLE as TYPE.
- Instance of: Dynamic type testing is supported by the operator:
  VARIABLE instanceof TYPE.

  If the test is done in a if-statement, it is not neccessary to cast the variable inside the inner blocks.

```
class Example {

def typeofExample {
var t : Class<?>
t = typeof(String)
t = String
}

def castExample {
var t : int
t = 123.456 as int
}

def instanceExample(t:Object
) {
var x : int
if (t instanceof Number) {
x = t.intValue
}
}

}
```

- SARL provides special operators in addition to the classic operators from Java or C++:

Operator	Semantic	Java equivalent
a == b	Object equality test	a.equals(b)
a != b	Object inequality test	!a.equals(b)
a === b	Reference equality test	a == b
a !== b	Reference inequality test	a != b
a <=> b	Compare a and b	Comparable interface
a .. b	Range of values $[a, b]$	n/a
a ..< b	Range of values $[a, b)$	n/a
a >.. b	Range of values $(a, b]$	n/a
a ** b	Compute a^b	n/a
a -> b	Create a pair (a, b)	n/a
a ?: b	If a is not null then a else b	a == null ? b : a
a?.b	If a is not null then a.b is called else a default value is used	a == null ? defaultValue : a.b
if (a) b else c	Inline condition	a ? b : c

OPERATOR DEFINITION AND OVERRIDING

- SARL allows overriding or definition operators.
- Each operator is associated to a specific function name that enables the developper to redefine the operator's code.
- Examples of operators in SARL:

Operator	Function name	Semantic
`col += value`	operator_add(Collection, Object)	Add an value into a collection.
`a ** b`	operator_power(Number, Number)	Compute the power b of a.

```
class Vector {
var x : float
var y : float
new (x : float , y :
float) {
this.x = x ; this.y = y
}
def operator_plus(v:
Vector)
: Vector {
new Vector(this.x + v.x,
this.y + v.y)
}
}
```

```
class X {
def fct {
var v1 = new Vector(1,
2)
var v2 = new Vector(3,
4)

var v3 = v1 + v2
}
}
```

OUTLINE

Full Professor

Université de Bourgogne Franche-Comté

Université de Technologie de Belfort-Montbéliard, France

Topics: Multiagent systems, Agent-based simulation, Agent-oriented software engineering, Mobility and traffic modeling

Web page: `http://www.multiagent.fr/People:Galland_stephane`
Email: stephane.galland@utbm.fr

Open-source contributions:

- `http://www.sarl.io`
- `http://www.janusproject.io`
- `http://www.aspecs.org`
- `http://www.arakhne.org`
- `https://github.com/gallandarakhneorg/`

Rodriguez, S., Gaud, N., and Galland, S. (2014). SARL: a general-purpose agent-oriented programming language. Warsaw, Poland. IEEE Computer Society Press.