

Programming (and Learning) Self-Adaptive & Self-Organising Behaviour with ScaFi

for Swarms, Edge-Cloud Ecosystems, and More

Roberto Casadei roby.casadei@unibo.it
Gianluca Aguzzi gianluca.aguzzi@unibo.it
Danilo Pianini danilo.pianini@unibo.it
Mirko Viroli mirko.viroli@unibo.it

Alma Mater Studiorum – Università di Bologna

Talk @ International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)



28/09/2023

Contents

1 Tutorial Quickstart

2 Introduction

- Context – Collective Adaptive Systems
- Aggregate Computing

3 Playing with ScaFi!

Tutorial references

- ***Where do I start?***

- <https://github.com/TODO>: repository with the examples/exercises proposed in the tutorial
Follow the README.md to run examples and setup a development environment.

Frequently Asked Questions (FAQs)

- **Where do I start?** Slide 3
- **Where do I learn more about the tools (ScaFi, Alchemist)?**
 - <https://scaficode.github.io/>
 - <http://alchemistsimulator.github.io/>
- **Where do I learn more about the theory/research on aggregate computing (AC)?**
Check out relevant presentations and papers.
 - <https://www.slideshare.net/RobertoCasadei/aggregate-computing-research-an-overview> – a quick overview of AC research
 - M. Viroli, J. Beal, F. Damiani, *et al.*, "From distributed coordination to field calculus and aggregate computing," *J. Log. Algebraic Methods Program.*, vol. 109, 2019. doi: 10.1016/j.jlamp.2019.100486 – a survey about AC, its history, and main developments
- **What is the research context?** Check out Slide 7–Slide 15
- **What kind of systems can be programmed/developed with AC?** Check out Slide 20 and Slide 7–Slide 15

Contents

1 Tutorial Quickstart

2 Introduction

- Context – Collective Adaptive Systems
- Aggregate Computing

3 Playing with ScaFi!

Contents

1 Tutorial Quickstart

2 Introduction

- Context – Collective Adaptive Systems
- Aggregate Computing

3 Playing with ScaFi!

Context

Modern IT systems are more and more **complex**

- increasing availability of wearable/mobile/embedded/flying devices
- increasing availability of heterogeneous wireless networks
- increasing availability of computational resources (edge/fog/cloud computing)
- increasing production of data **everywhere** and **anytime**

The challenge

Consider the worst-case possible scenario

- **zillion** of devices *unpredictably* located and moving in a space
- heterogeneous displacement, **pervasive** sensing/actuation
- computational services are **contextual** (proximity-based) and *dynamic*

How can we **program** such systems?

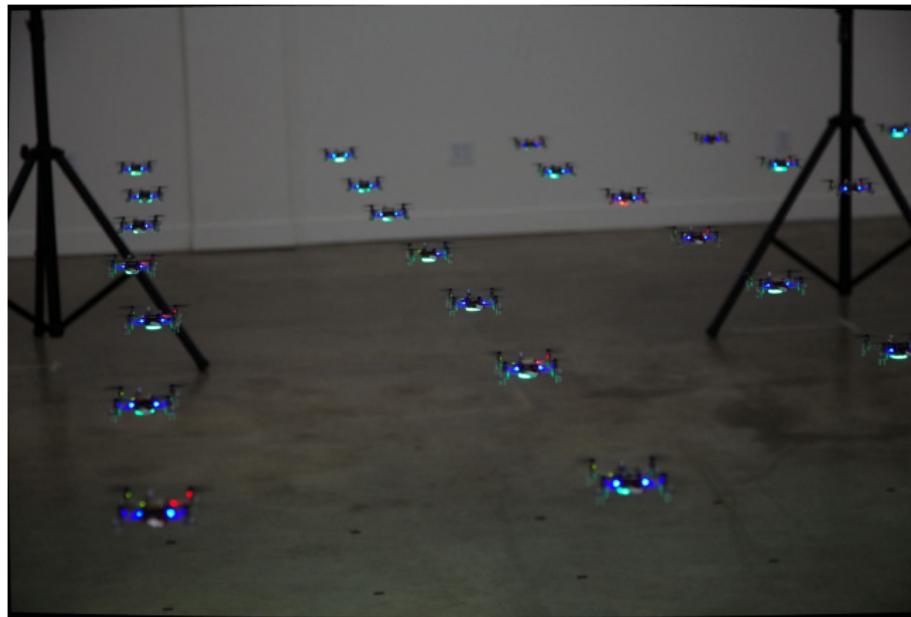
What are the right **abstractions**?

Collective Adaptive Systems (CASs)

Systems composed of *possibly large* set of component executing a **collective** task strongly relying on component **interactions** and showing **inherent adaptivity**.



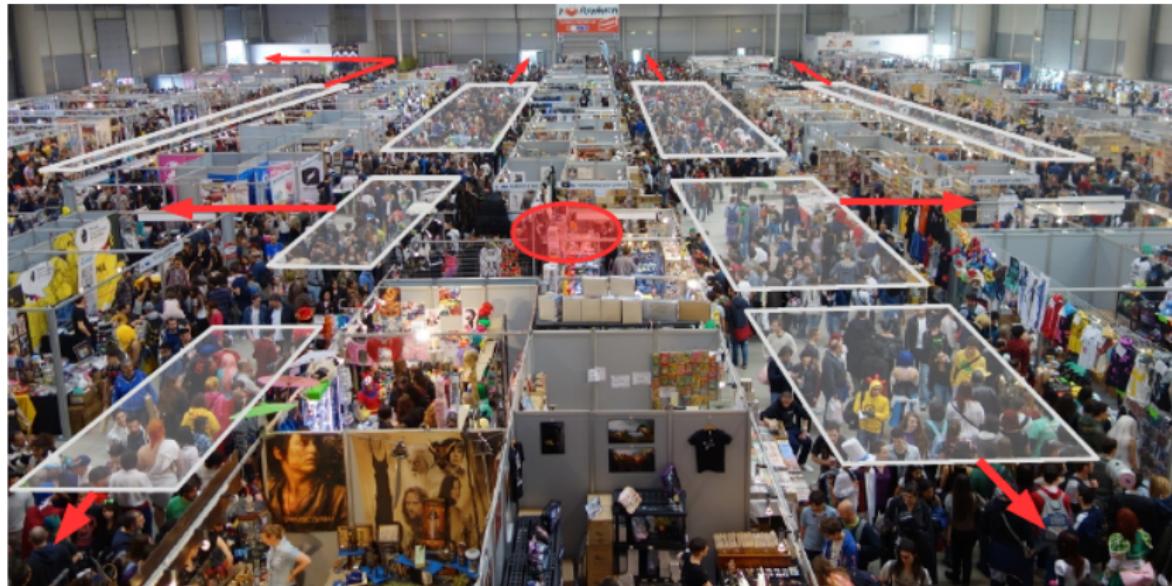
Examples: controlling a fleet of drones



Issues

- Design techniques to **reactively** propagate information across the fleet
- Create algorithms for **higher-level** programming of the fleet

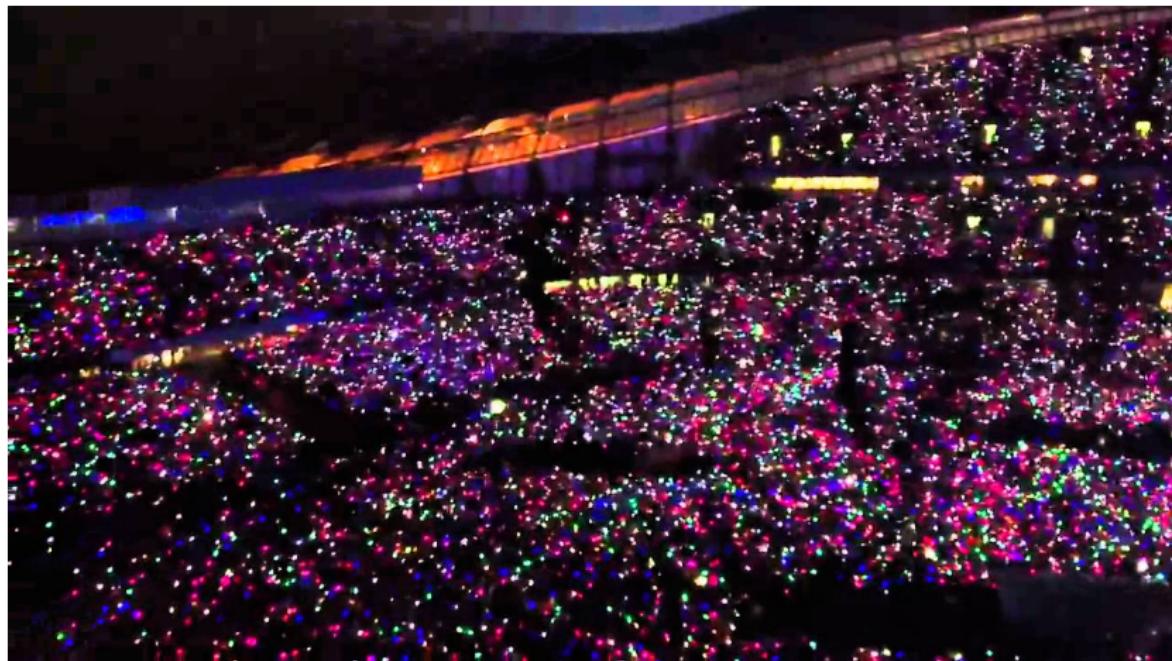
Examples: pedestrian steering in (smart)cities



Issue

- Design **self-organisation** algorithms for people navigation
- More generally, design map-based large-scale applications

Examples: fine control of crowds



Issues

- Evaluating the influence of environment in crowd formation
- Evaluating the influence of ambient sensors, cameras and wearable devices

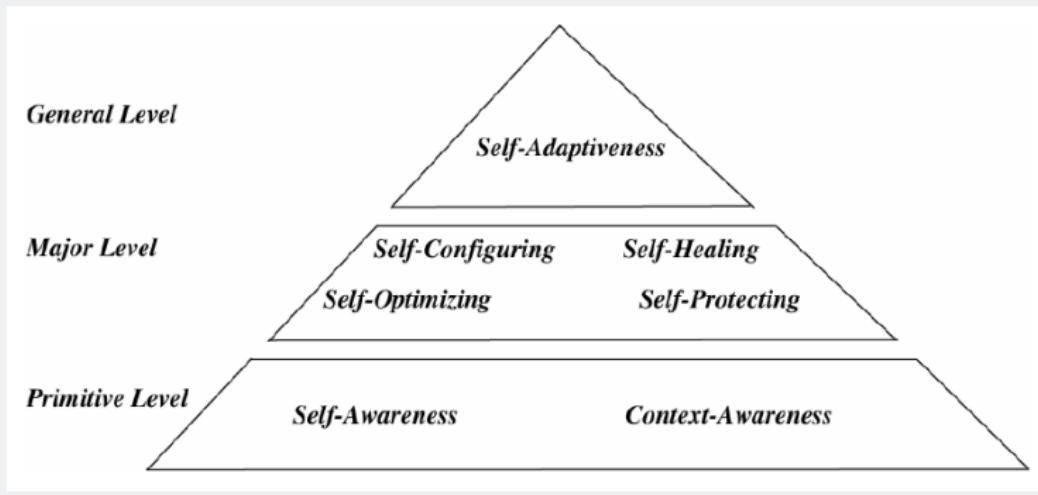
Collective Adaptive Systems

Adaptation

A system is **adaptive** if it can **change** its behaviour depending on **circumstances**, in order to better reach its **goal**

→ Adaptiveness is **intrinsic** to open complex system (like CASs)

Kind of adaptiveness, self-* properties



Kind of adaptiveness

Self-adaptiveness: general level

- **self-adaptiveness** → the ultimate property we perceive from outside
- the terms “self-***” recalls a property achieved in **autonomy**
- subcases: self-managing, -governing, -maintenance, -control

Major level

- internal properties related to overall system management
- subcases: self-configuring, self-healing, self-optimising, self-protecting
- defined in the context of **autonomic** computing

Primitive level

- internal properties related to primitive aspects
- subcases: self-awareness, context-awareness
- means being able to properly perceive the own state, context, ...

The case of self-organisation

Self-organisation

The ability of creating **spatial/temporal** patterns out of the *local interaction* of individuals

Self-adaptive vs. self-organising

- self-adaptive: a top-down way of achieving adaptiveness
- we have the adaptation goal, and accordingly guide components
- self-organising: a bottom-up way of achieving adaptiveness
- the adaptive behaviour emerges from local interactions
 - The typical approach in large-scale CASs
 - Born in the context of **swarm intelligence**

Good Abstraction for CASs?

Ideas

- specify **overall** behaviour, not *individual* device program
- abstract from the actual shape of **interactions**
- **automatically** adapt to environment details
- focus on how the **global output** pattern can be obtained from global inputs
- focus on both **spatial** and **temporal** computing patterns
- ➔ the ideas around **macro-programming** paradigms [2]!
 - Aggregate computing
 - Buzz
 - ...
- Ruccurent abstractions:
 - Ensembles & collective tasks
 - Self-organising information flows
 - Self-healing collective structures (e.g., *gradients*)

[2] R. Casadei, "Macroprogramming: Concepts, state of the art, and opportunities of macroscopic behaviour modelling," *ACM Comput. Surv.*, vol. 55, no. 13s, 2023, issn: 0360-0300. doi: 10.1145/3579353

Contents

1 Tutorial Quickstart

2 Introduction

- Context – Collective Adaptive Systems
- Aggregate Computing

3 Playing with ScaFi!

Aggregate Computing (AC) in 1 Slide

Self-org-like computational model

interaction: **repeated** msg exchange with neighbours

behaviour: **repeated** execution of **async rounds** of sense

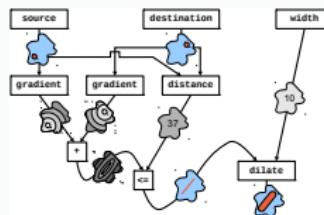
- compute - (inter)act

formal model of executions: event structures

abstraction: **computational fields** ($dev/evt \mapsto V$)

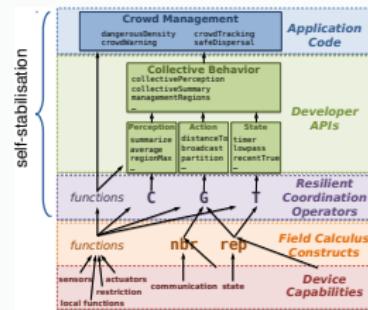
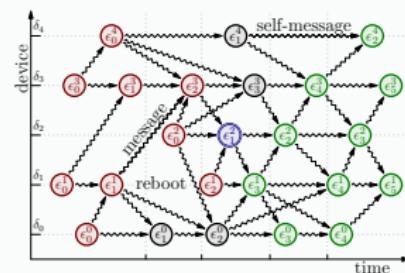
formal core language: **field calculus** [1]

paradigm: **functional, macro-programming**

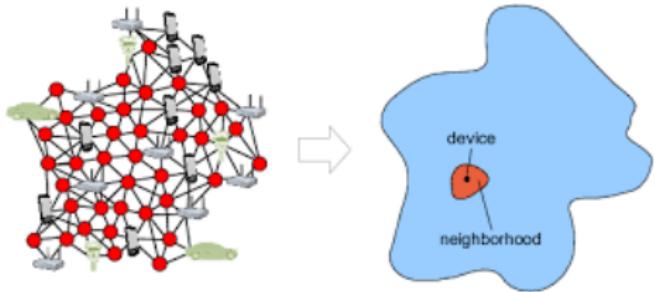


```
def channel(source: Boolean, target: Boolean, width: Double) =
  dilate(gradient(source) + gradient(target)) <=
    distance(source, target), width
```

M. Viroli, J. Beal, F. Damiani, et al., "From distributed coordination to field calculus and aggregate computing," *J. Log. Algebraic Methods Program.*, vol. 109, 2019. doi: 10.1016/j.jlamp.2019.100486



J. Beal, D. Pianini, and M. Viroli, "Aggregate programming for the internet of things," *IEEE Computer*, vol. 48, no. 9, pp. 22–30, 2015. doi: 10.1109/MC.2015.261



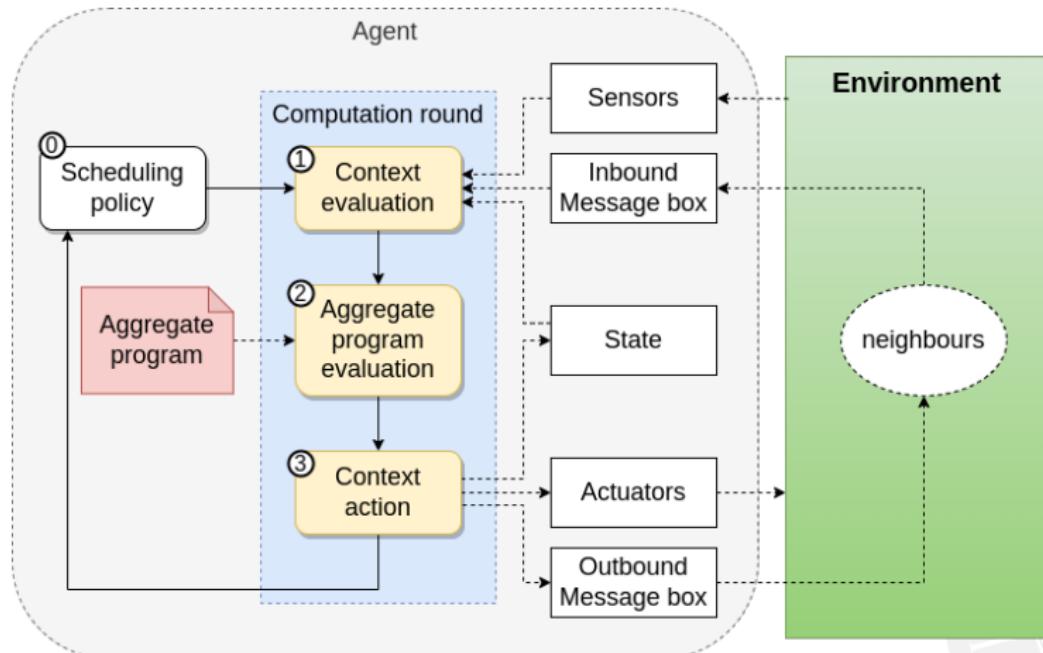
Aggregate Computing

Program the **aggregate**, not the individual device

- *Computing Machine* → an ensemble of devices as a single body, fading the actual space
- *Elaboration process* → atomic manipulation of a *collective* data structure (**computational field**)
- *Networked computation* → a proximity-based self-organising system hidden “under-the-hood”

Computational model – self-organisation-like execution

- *Continuous* communication with **neighbours** only (→decentralisation)
- *Continuous* execution of async **rounds** of sense - compute - communicate



Applicability: in a nutshell

Essentially, this approach can be used to program **any** system that can be recast to the following

- **Structure:** network of locally interacting agents
 - **neighbouring relationship:** e.g. based on network connectivity, physical distance, social relationship, or whatever
 - **dynamicity/openness:** devices may move/fail, enter/exit the system at runtime, and neighbourhoods may change
- **Interaction:**
 - with neighbours: asynchronous message passing
 - with environment: via sensors and actuators
- **Behaviour:** **sense–compute–interact**
 - **sense:** acquire **context** (messages from neighbours, values from sensors)
 - **compute:** mapping context to (inter-)action
 - **interact:** sending messages to neighbours and running actuation

So, best for applications that are:

- **progressive/long-running:** require multiple coordinated steps/rounds of local processing and action

Assuming this **system model**, AC is about how to specify the “compute” part so that **global emergent** outcomes can be eventually attained

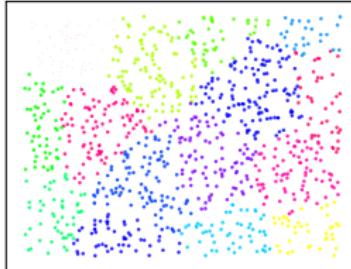
Computational Fields: a static view

Traditionally a map: $Space \mapsto Values$

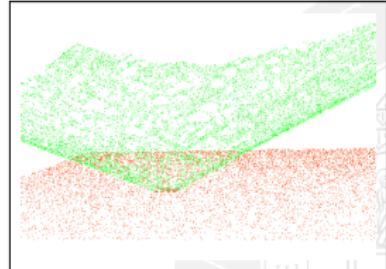
- possibly: evolving over time, dynamically injected, stabilising
- smoothly adapting to very heterogeneous domains
- more easily "*understood*" on continuous and flat spatial domains
- ranging to booleans, reals, vectors, functions



boolean channel in 2D



numeric partition in 2D

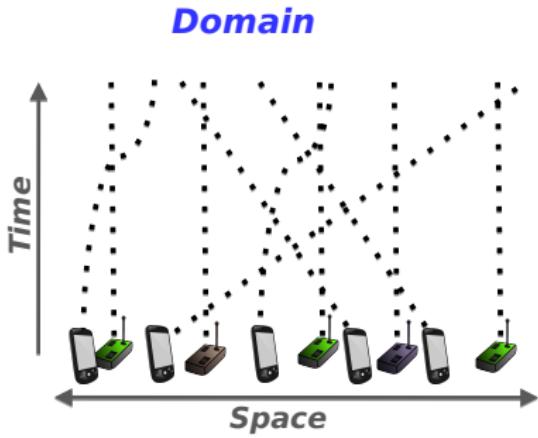


real-valued gradient in 3D

Computational Fields revisited: a dynamic view

A field as a *space-time* structure: $\phi : D \mapsto V$

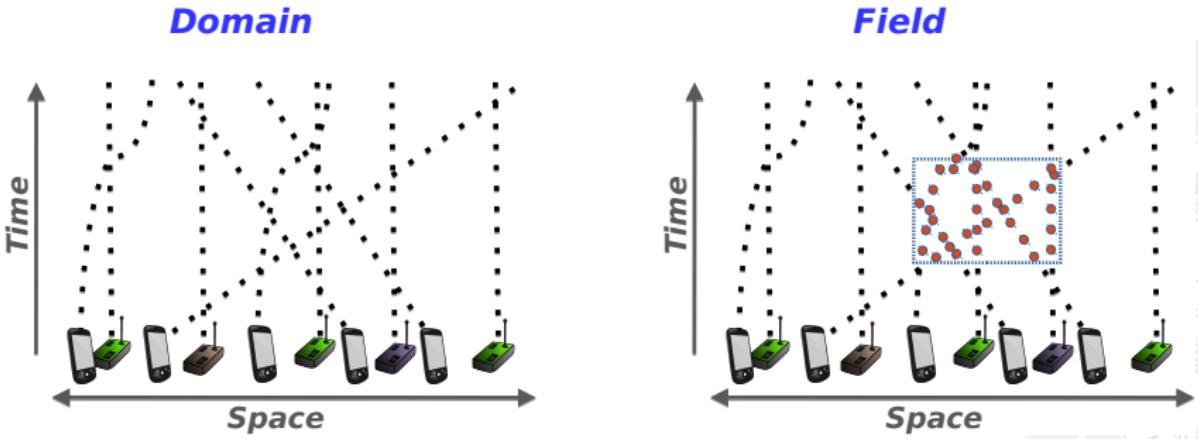
- event E : a triple $\langle \delta, t, p \rangle$ – device δ , “firing” at time t in position p
 - events domain D : a coherent set of events (devices cannot move too fast)
 - field values V : any data value
- computation abstracts from/adapts to the underlying event
- scheduling of events is essentially exogenous



Computational Fields revisited: a dynamic view

A field as a *space-time* structure: $\phi : D \mapsto V$

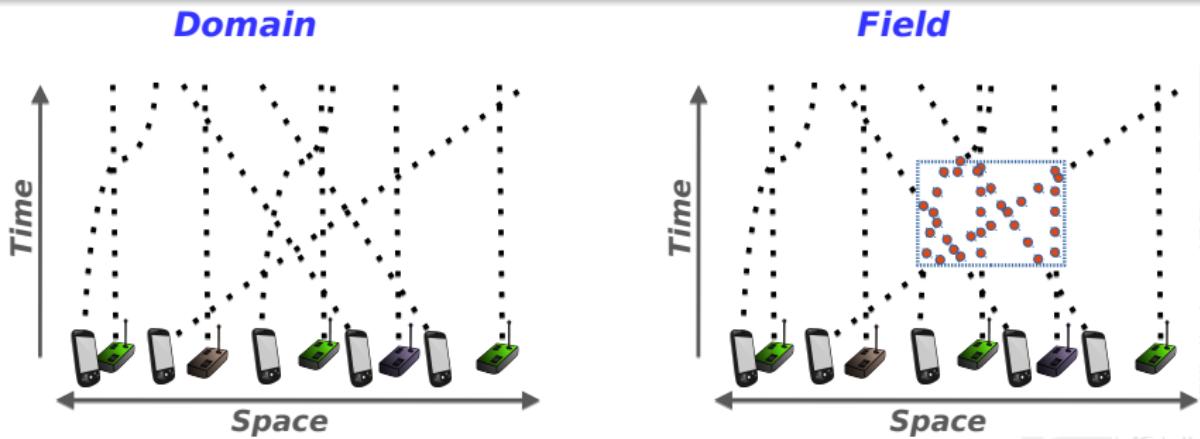
- event E : a triple $\langle \delta, t, p \rangle$ – device δ , “firing” at time t in position p
- events domain D : a coherent set of events (devices cannot move too fast)
- field values V : any data value
- computation abstracts from/adapts to the underlying event
- scheduling of events is essentially exogenous



Computational Fields revisited: a dynamic view

A field as a *space-time* structure: $\phi : D \mapsto V$

- event E : a triple $\langle \delta, t, p \rangle$ – device δ , “firing” at time t in position p
- events domain D : a coherent set of events (devices cannot move too fast)
- field values V : any data value
- computation abstracts from/adapts to the underlying event
- scheduling of events is essentially exogenous

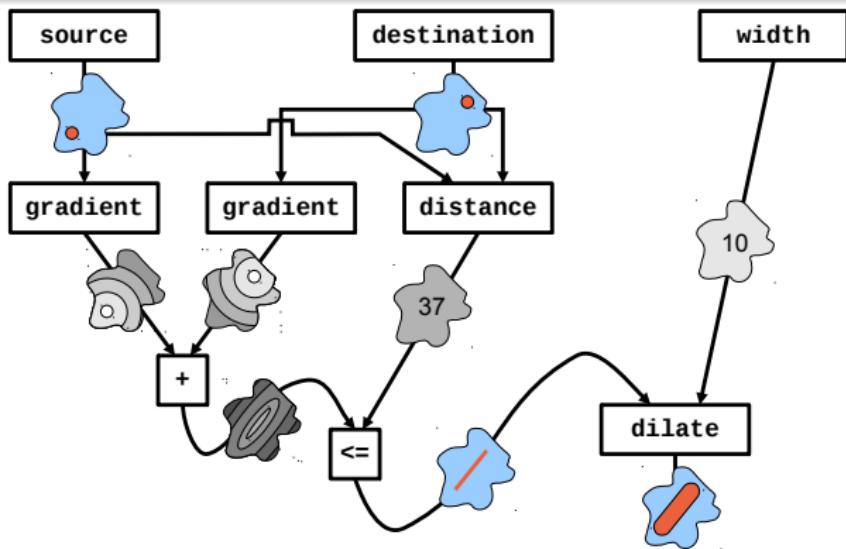


will later show only snapshots of fields in 2D space..

Aggregate programming as a functional approach

Functionally composing fields

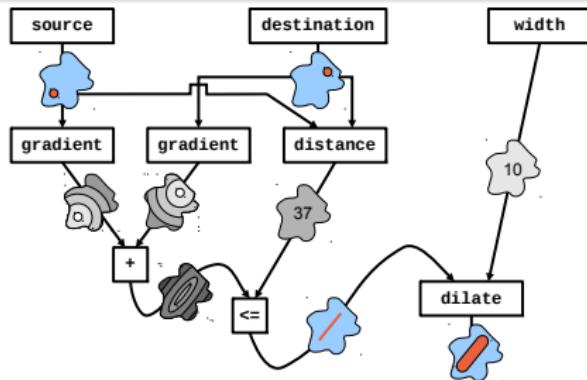
- **Inputs:** sensor fields, Output: actuator field
- Computation is a pure function over fields (time embeds state!)
- for this to be practical/expressive we need a good programming language



Preview

How do we want that computation to be expressed?

- source, dest and width as inputs
- gradient, distance and dilate as reusable functions
- note we are reusing and composing global-level, aggregate specs



```
def channel(source: Boolean, dest: Boolean, width: Double): Boolean = {
  dilate(gradient(source) + gradient(dest) <= distance(source, dest), width)
}
```

Field calculus model

Key idea

- a sort of λ -calculus with “everything is a field” philosophy!

Syntax – Reduced

$e ::= x \mid v \mid e(e_1, \dots, e_n) \mid \text{rep}(e_0)\{e\} \mid \text{nbr}\{e\}$	(expr)
$v ::= < \text{standard-values} > \mid \lambda$	(value)
$\lambda ::= f \mid o \mid (\bar{x})=>e$	(functional value)
$F ::= \text{def } f(\bar{x}) \{e\}$	(function definition)

Few explanations

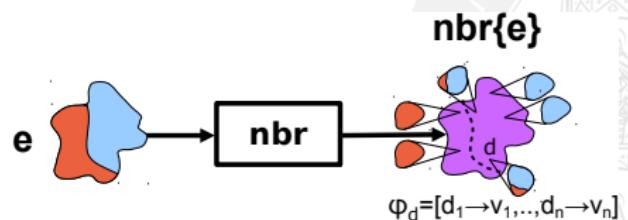
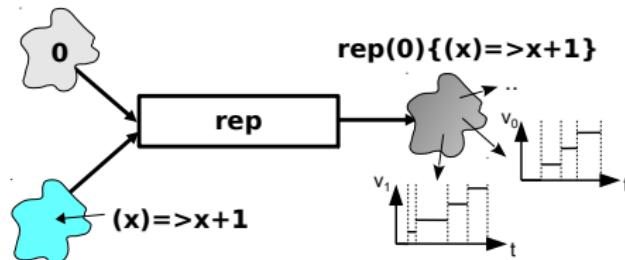
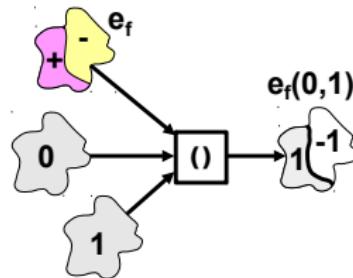
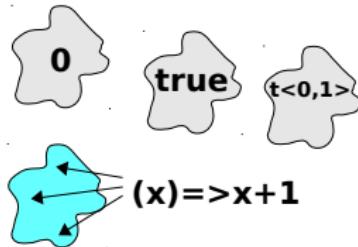
- v includes numbers, booleans, strings,..
..tuples/vectors/maps/any-ADT (of expressions)
- f is a user-defined function (the key aggregate computing abstraction)
- o is a built-in local operator (pure math, local sensors,...)

Intuition of global-level (denotational) semantics

The four main constructs at work

⇒ values, application, evolution, and interaction – in aggregate guise

- $e ::= \dots \mid v \mid e(e_1, \dots, e_n) \mid \text{rep}(e_0)\{e\} \mid \text{nbr}\{e\}$



Intuition of global-level semantics – More

Value v

- A field constant in space and time, mapping any event to v

Function application $e(e_1, \dots, e_n)$

- e evaluates to a field of functions, assume it ranges to $\lambda_1, \dots, \lambda_n$
- this naturally induces a partition of the domain D_1, \dots, D_n
- now, join the fields: $\forall i, \lambda_i(e_1, \dots, e_n)$ restricted in D_i

Repetition $\text{rep}(e_0)\{e_\lambda\}$

- the value of e_0 where the restricted domain “begins”
- elsewhere, unary function e_λ is applied to previous value at each device

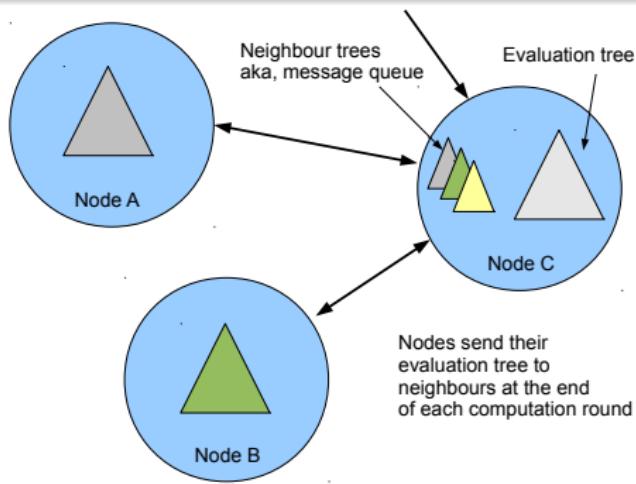
Neighbouring field construction $\text{nbr}\{e\}$

- at each event gathers most recent value of e in neighbours (in restriction)
- ..what is neighbour is orthogonal (i.e., physical proximity)

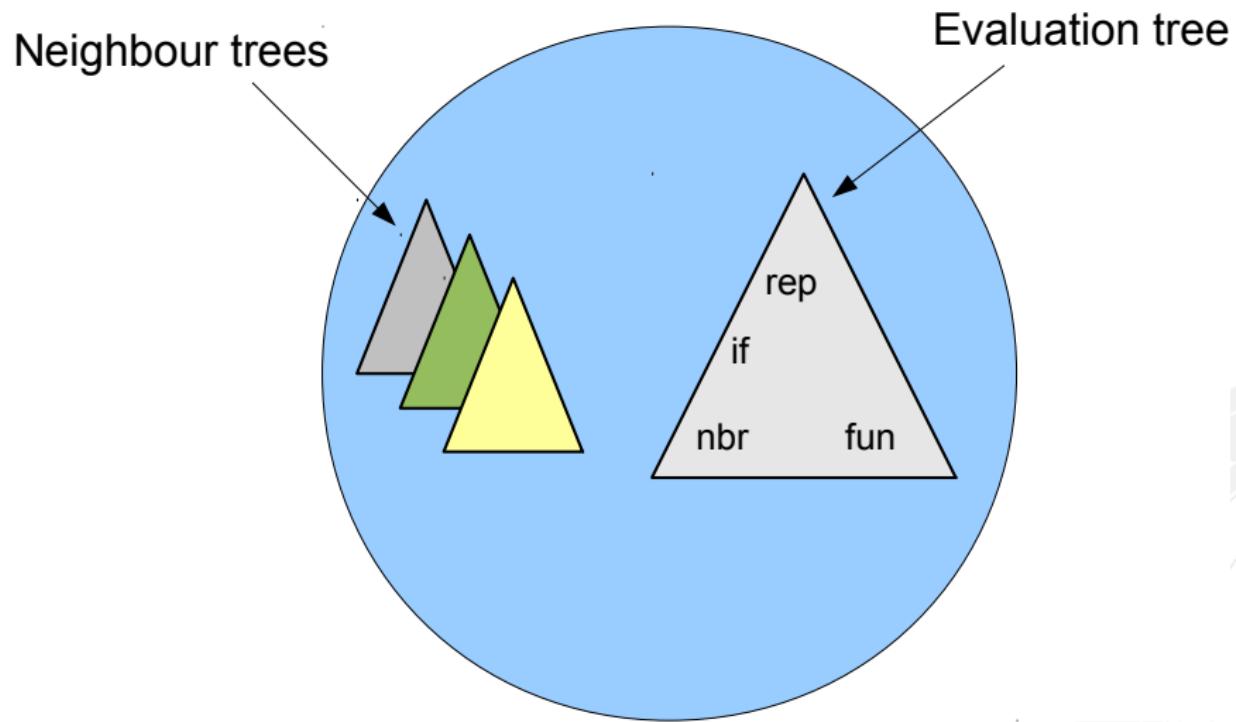
Key aspects of the semantics: network model

Platform abstract model

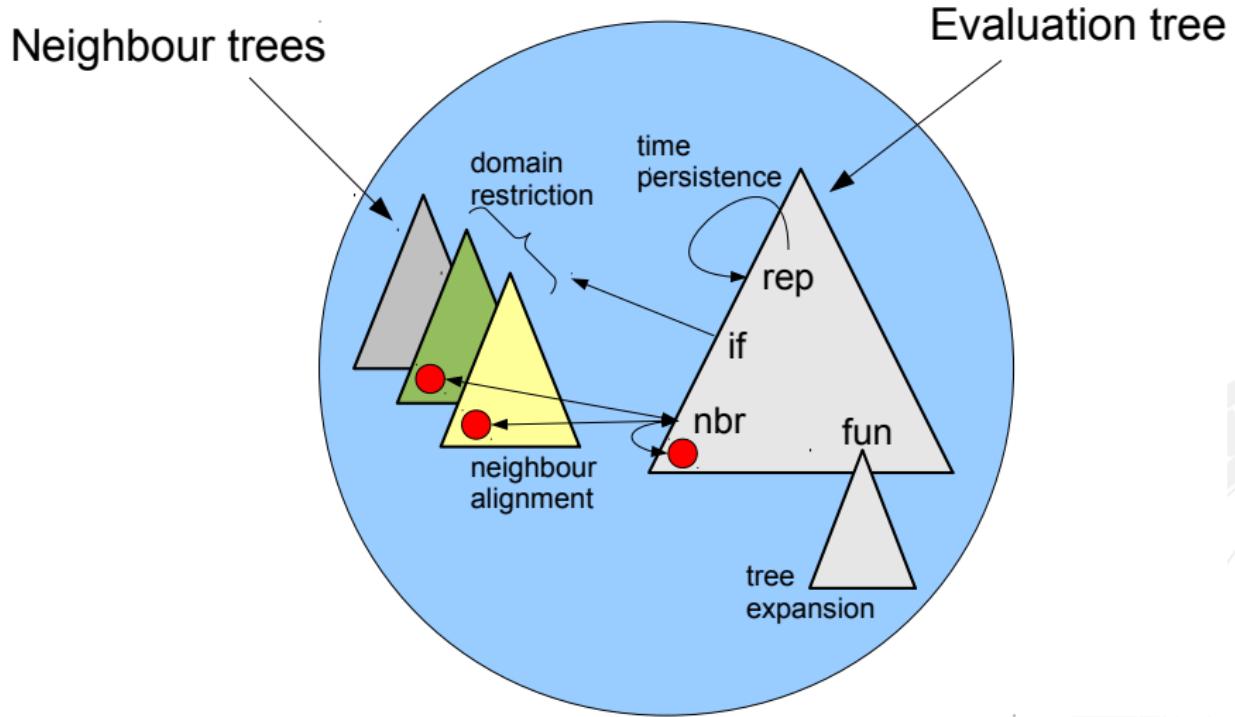
- A node state θ (value-tree) updated at asynchronous rounds
- At the end of the round, θ is made accessible to the neighbourhood
- A node state is updated “against” recently received neighbours’ trees



Tree evaluation: pictorial semantics



Tree evaluation: pictorial semantics



Core mechanisms in the operational semantics

Orthogonally..

- evaluation proceeds recursively on expression and neighbour trees
- neighbour trees may be discarded on-the-fly if not “aligned” (restriction)

Core mechanisms in the operational semantics

Orthogonally..

- evaluation proceeds recursively on expression and neighbour trees
- neighbour trees may be discarded on-the-fly if not "aligned" (restriction)

Function application $e(e_1, \dots, e_n)$

- evaluates the body against a filtered set of neighbours ...
- ..i.e., only those which evaluated e to the same result

Repetition $\text{rep}(e_0)\{e_\lambda\}$

- if a previous value-tree of mine is available, evaluates e_λ on its root
- otherwise, evaluates e_0

Neighbouring field construction $\text{nbr}\{e\}$

- gather values from neighbour trees currently aligned
- add my current evaluation of e

Operational semantics as a blueprint for platform support

Requirements

- a notion of the neighbourhood must be defined — wireless connectivity, physical proximity ...
- nodes execute in asynchronous rounds, and emit a “round result”
- a node needs to have recent round results of neighbours
- by construction we tolerate losses of messages
- by construction we tolerate various round frequencies



Operational semantics as a blueprint for platform support

Requirements

- a notion of the neighbourhood must be defined — wireless connectivity, physical proximity ...
- nodes execute in asynchronous rounds, and emit a “round result”
- a node needs to have recent round results of neighbours
- by construction we tolerate losses of messages
- by construction we tolerate various round frequencies

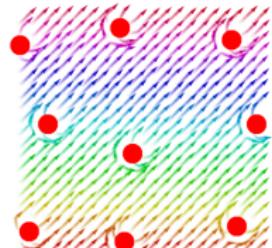
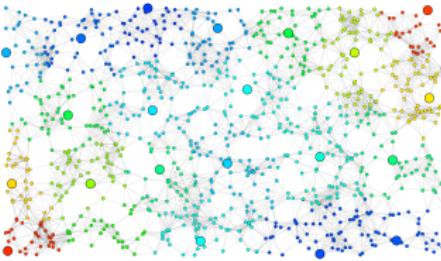
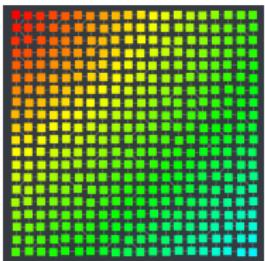
Platform details are very orthogonal to our programming model!

- the above requirements can be met by various platforms
- *programming remains mostly unaltered!*

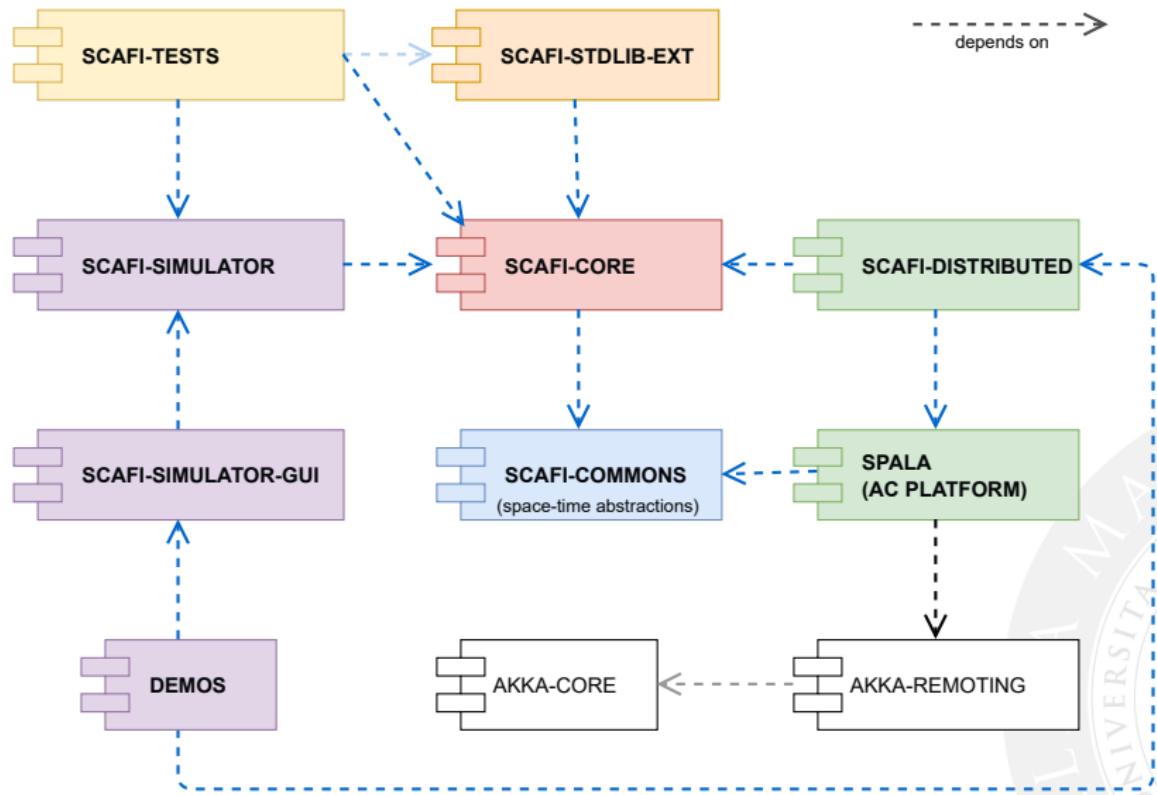


ScaFi (**S**c**a** **F**ields)

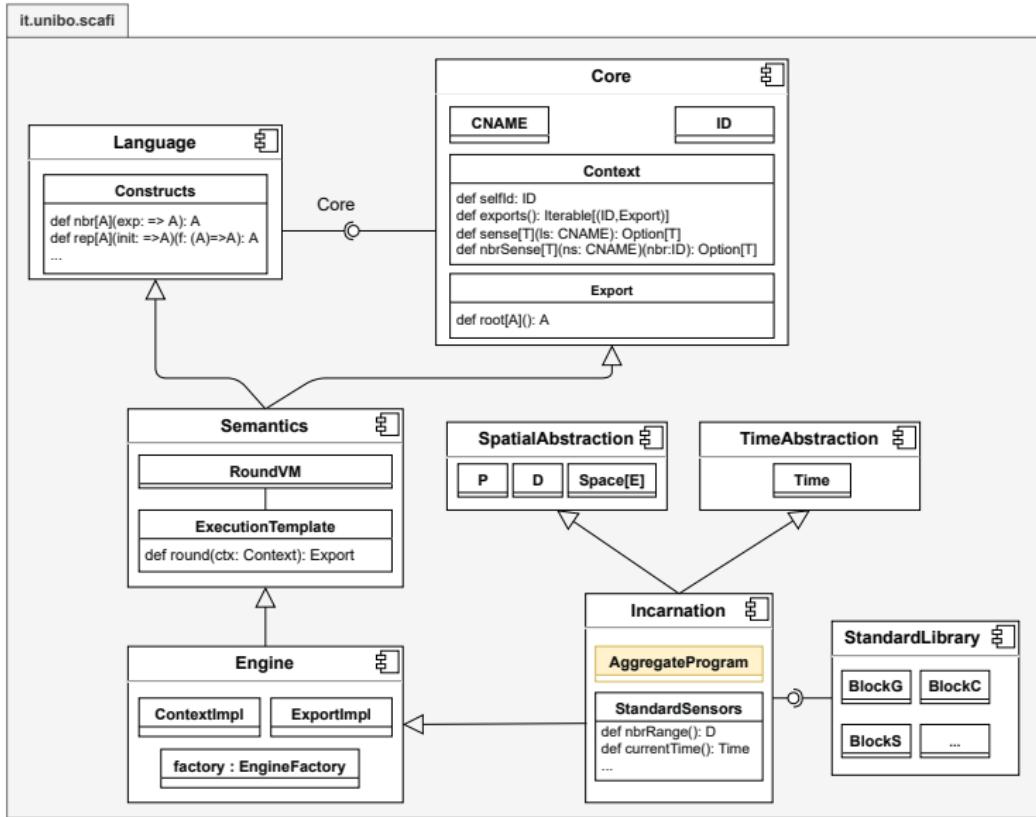
A Scala toolkit providing an *internal domain-specific* language, *libraries*, a *simulation* environment, and *runtime* support for **practical** aggregate computing systems development



ScaFi: Organization



ScaFi: Design



ScaFi: syntax as a core language/API

```
trait Constructs {  
    // the unique identifier of the local device  
    def mid(): ID  
  
    // applies fun to the previous result, or to init at the first call  
    def rep[A](init: A)(fun: (A) => A): A  
  
    // evaluation of expr at the currently-considered neighbour  
    def nbr[A](expr: => A): A  
  
    // accumulates available evaluations of expr, with acc/init monoid  
    def foldhood[A](init: => A)(acc: (A,A)=>A)(expr: => A): A  
  
    // splits computation: th where cond is true, el everywhere/time else  
    def branch[A](cond: => Boolean)(th: => A)(el: => A): A  
  
    // perception of local sensor  
    def sense[A](name: CNAME): A  
  
    // perception of neighbourhood sensor  
    def nbrvar[A](name: CNAME): A  
    ...  
}
```

ScaFi: setup an aggregate application

```
package experiments
// STEP 1 Choose an incarnation (simulated or real)
import it.unibo.scafi.incarnations.BasicSimulationIncarnation._

// STEP 2 Define the aggregate program including the right libraries
class MyProgram extends AggregateProgram with Libs {
    // STEP 2.1 Define main logic of the program
    override def main(): Any = ...
}

// STEP 3 Platform/Node setup (both in simulation/real)

// STEP 3.1 in case of ScaFi simulation:
object SimulationRunner extends Launcher {
    Settings.Sim_ProgramClass = "experiments.MyAggregateProgram"
    Settings.ShowConfigPanel = true
    launch()
}
```

So let us start playing with ScaFi!! 😊

Contents

1 Tutorial Quickstart

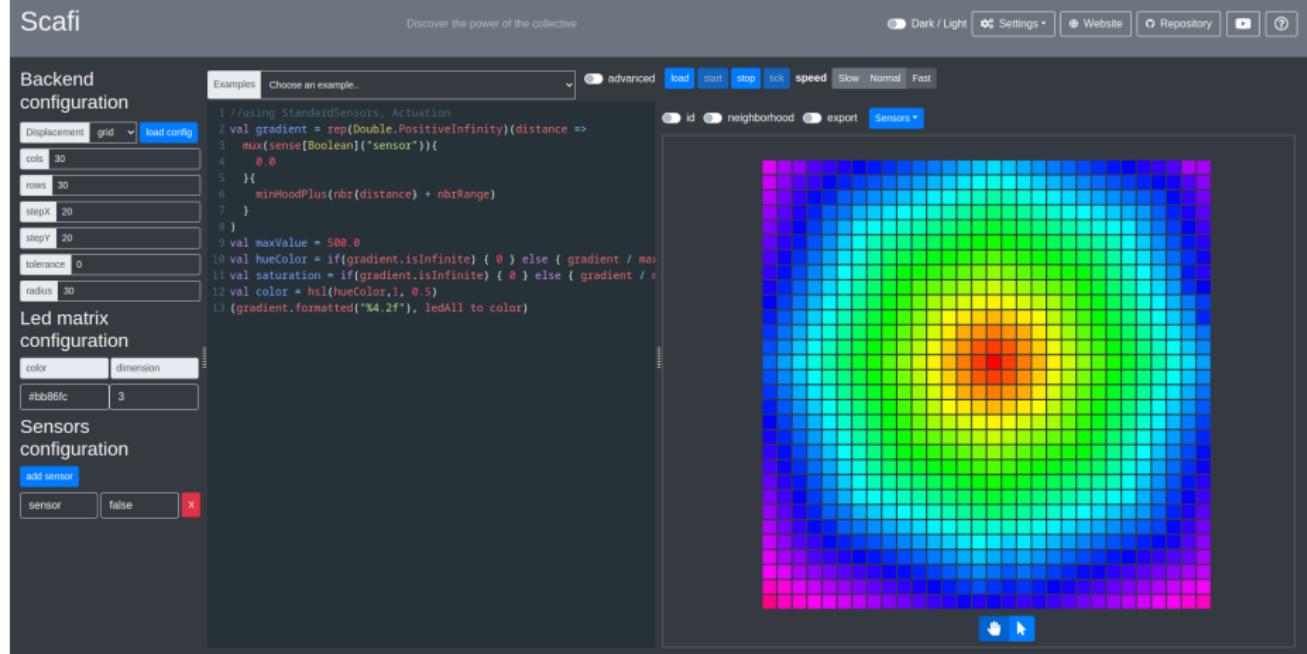
2 Introduction

- Context – Collective Adaptive Systems
- Aggregate Computing

3 Playing with ScaFi!

An aggregate computing playground – ScaFi Web!

<https://scafi.github.io/web/>



The screenshot shows the ScaFi Web interface. On the left, there are three configuration sections: "Backend configuration" (Displacement, grid, load config), "Led matrix configuration" (color, dimension, #bbb8fc, 3), and "Sensors configuration" (add sensor, sensor, false). In the center, there is a code editor with the following Scala code:

```

1 //using StandardSensors, Actuation
2 val gradient = rep[Double.PositiveInfinity](distance =>
3   mux(sense[Boolean]("sensor")){
4     0.0
5   }{
6     minHoodPlus(nbr(distance) + nbrRange)
7   }
8 )
9 val maxValue = 500.0
10 val hueColor = if(gradient.isInfinite) { 0 } else { gradient / maxValue }
11 val saturation = if(gradient.isInfinite) { 0 } else { gradient / radius }
12 val color = hsl(hueColor,1,0.5)
13 (gradient.formatted("%4.2f"), ledAll to color)

```

Below the code editor are buttons for advanced mode, load, start, stop, tick, speed (Slow, Normal, Fast), id, neighborhood, export, Sensors, and a help icon.

To the right, a large 3D perspective view of a 30x30 grid of colored squares is displayed. The colors transition from purple at the edges to yellow in the center, forming a gradient. A cursor icon is visible at the bottom right of the grid area.

Programming (and Learning) Self-Adaptive & Self-Organising Behaviour with ScaFi

for Swarms, Edge-Cloud Ecosystems, and More

Roberto Casadei roby.casadei@unibo.it
Gianluca Aguzzi gianluca.aguzzi@unibo.it
Danilo Pianini danilo.pianini@unibo.it
Mirko Viroli mirko.viroli@unibo.it

Alma Mater Studiorum – Università di Bologna

Talk @ International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)



28/09/2023

References |

- [1] M. Viroli, J. Beal, F. Damiani, G. Audrito, R. Casadei, and D. Pianini, "From distributed coordination to field calculus and aggregate computing," *J. Log. Algebraic Methods Program.*, vol. 109, 2019. doi: 10.1016/j.jlamp.2019.100486.
- [2] R. Casadei, "Macroprogramming: Concepts, state of the art, and opportunities of macroscopic behaviour modelling," *ACM Comput. Surv.*, vol. 55, no. 13s, 2023, issn: 0360-0300. doi: 10.1145/3579353.
- [3] J. Beal, D. Pianini, and M. Viroli, "Aggregate programming for the internet of things," *IEEE Computer*, vol. 48, no. 9, pp. 22–30, 2015. doi: 10.1109/MC.2015.261.