Analysis and Design of Collective Behavior by Aggregate Computing

Mirko Viroli

ALMA MATER STUDIORUM—Università di Bologna, Italy
mirko.viroli@unibo.it

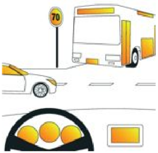
Open Problems in Concurrency Theory
Bertinoro, 30/6/2023

Building resilient distributed systems for the “complex IoT”

Elements of variability

- heterogeneity of devices: wearable / mobile / embedded / flying devices
- heterogeneity of connectivity: BT, wifi, 4G/5G (via brokers, p2p, ...)
- heterogeneity of computational resources: edge-cloud continuum
- pervasive change/dynamism: faults, delays, changing conditions




Problem statement, and Research Goal

Problem statement

Devise a minimal computational model to express distributed/collective systems in a way completely independent of the underlying platform (scheduling, displacement of devices, connectivity, platform, scale, . . .)

Key idea: "program of the entire system, as a whole, not the individual device"

Research Goals

Conceive the full stack:

- key abstraction, core calculus, properties, programming language
- simulation support, execution platforms
- algorithms, applications

Motto: "the platform can change, but the program remains the same"

An analogy with stream programming

Averaging the size of lines in an iterable of strings (in Scala)

```
val op = (dataset: Iterable[String]) =>
    dataset.map(_.size)
           .map((_, 1.0))
           .reduceOption((x, y) => (x._1 + y._1, x._2 + y._2))
           .map{case (sum, size) => sum/size}
           .get
```

How is this executed? Which assumptions? . . . It doesn't matter!

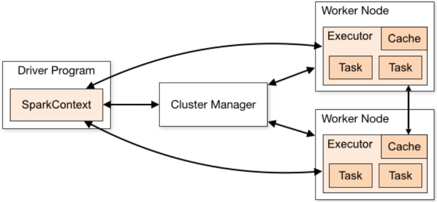
- Could be an in-memory list, a text-file, a sensor stream, a big-data on a cluster
- Could compute by a single-thread, multiple-threads, in a cluster, in a distributed and faulty database

Same motto: "the platform can change, but the program remains the same"

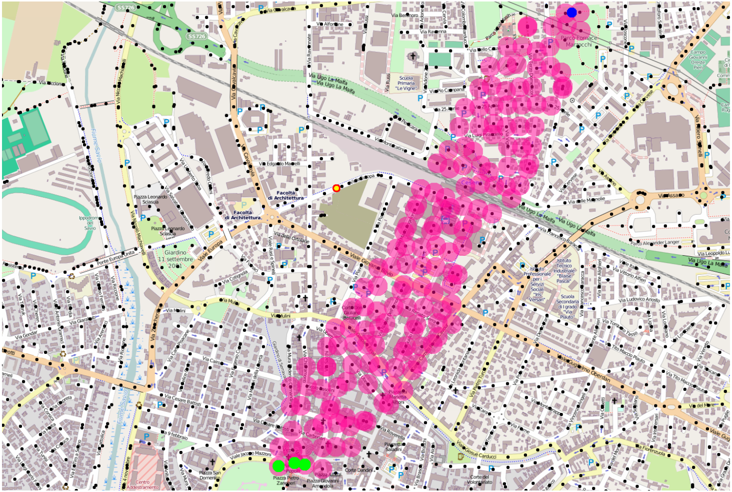
Stream programming in Apache Spark

Averaging the length of lines in an iterable of strings (in Scala)

```
val op = (dataset: Iterable[String]) =>
    dataset.map(_.size)
           .map((_, 1.0))
           .reduceOption((x, y) => (x._1 + y._1, x._2 + y._2))
           .map{case (sum, size) => sum/size}
           .get
```



A case: computing a redundant route in a smart-city



Dynamically and continuously adapting: avoiding traffic, road construction, ...

Macro-programming

Programming "group interaction in space"[1]

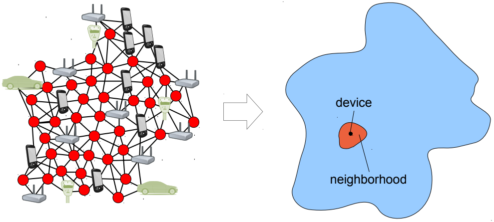
[1] Roberto Casadei. "Macroprogramming: Concepts, State of the Art, and Opportunities of Macroscopic Behaviour Modelling". In: *ACM Comput. Surv.* (2023)

- *Device abstractions* – make interaction implicit
  NetLogo, Hood, TOTA, Gro, MPI, and the SAPERE approach
- *Pattern languages* – supporting composability of spatial behaviour
  Growing Point, Origami Shape, various selforg pattern langs
- *Information movement* – gathering in space, moving elsewhere
  TinyDB and Regiment
- *Spatial computing* – program space-time behaviour of systems
  Proto, MGS
- *Aggregate computing* – programming functional composition of computational fields
  Field calculus and ScaFi

Aggregate computing

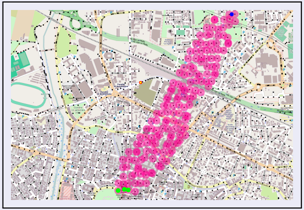
Key principles

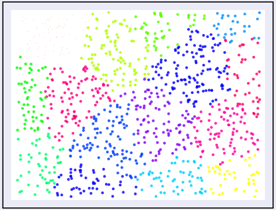
1. The reference computing machine
   ⇒ an aggregate of devices as single "body"
2. The metaphor/methodology
   ⇒ could abstract "body" to the actual *space* where the system runs
3. The computational model
   ⇒ iterative and distributed evolution of a (computational) field
4. Key programming mechanism
   ⇒ stream programming "against the neighbourhood"



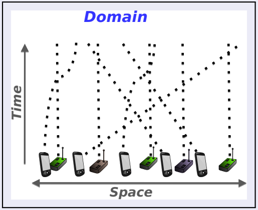
Computational Fields

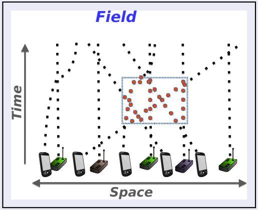
Static view: *Devices* ↦ *Values* (abst. to *Space* ↦ *Values*)





Dynamic view: *Events* ↦ *Values* (abst. to *SpaceTime* ↦ *Values*)



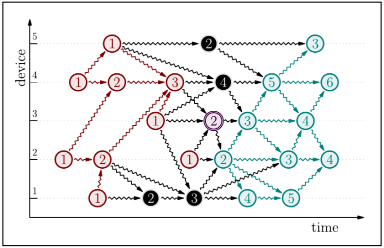


Field denotation, over event structures

Augmented event structure (a situated DAG of events) [2]

[2] Giorgio Audrito et al. "A Higher-Order Calculus of Computational Fields". In: *ACM Transactions on Computational Logic* 20.1 (Jan. 2019), 5:1–5:55

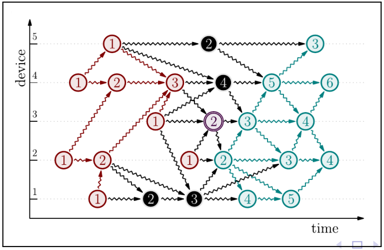
- events: devices that perform a computation and send messages
- arrows between events of different devices: (message) causation
- arrows between events of the the same device: state persistence
- denotation of field: a map from an ES to values



Programming fields, operational semantics

Round-based semantics of a program P

- the platform manages the neighbourhood relation (which is dynamic)
- only the latest message from a neighbour is retained
- at each event, P is used to turn input messages and sensor data to an output message
- operational semantics schema [2]: $\delta; \Theta; \sigma \vdash \mathrm{e}_P \Downarrow \theta$
  Read "at device δ, with messages Θ and sensor data σ, evaluation of e_P gives result/message θ"

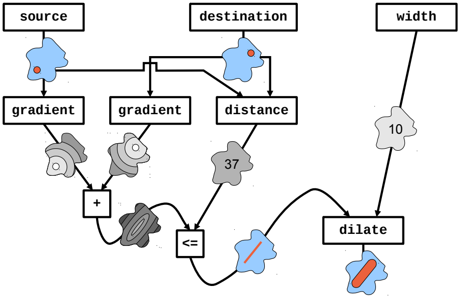


Aggregate programming as a functional approach

Sought features for a programming language (or core calculus) for P [3]

[3] Jacob Beal, Danilo Pianini, and Mirko Viroli. "Aggregate Programming for the Internet of Things". In: *IEEE Computer* 48.9 (2015), pp. 22–30

- Purely functional: it turns fields (/sensors) into a field (/actuator)
- Composable: function composition as modularisation/reuse mechanism
- Declarative (stream-oriented) constructs to deal with space/time

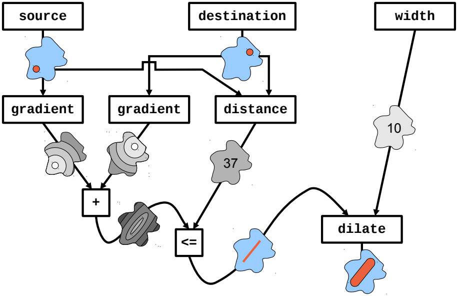


Preview

How we want that computation to be expressed?

- `source`, `dest` and `width` as (typed) inputs
- `gradient`, `distance` and `dilate` as reusable functions

⇒ note the "global-level composition" feeling




```
def channel(source: Boolean, dest: Boolean, width: Double): Double =
  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 (slightly refactored, semi-formal version of papers')

$$\begin{array}{lr}
\texttt{e} ::= \texttt{x} \mid \texttt{v} \mid \texttt{e}(\texttt{e}_1,\ldots,\texttt{e}_n) \mid \texttt{rep}(\texttt{e}_0)\{\texttt{e}\} \mid \texttt{nbr}\{\texttt{e}\} & \text{(expr)} \\
\texttt{v} ::= < \text{standard-values} > \mid \lambda & \text{(value)} \\
\lambda ::= \texttt{f} \mid \texttt{o} \mid (\overline{\texttt{x}})\texttt{=>e} & \text{(functional value)} \\
\texttt{F} ::= \texttt{def f}(\overline{\texttt{x}})\ \{\texttt{e}\} & \text{(function definition)}
\end{array}$$

Few explanations

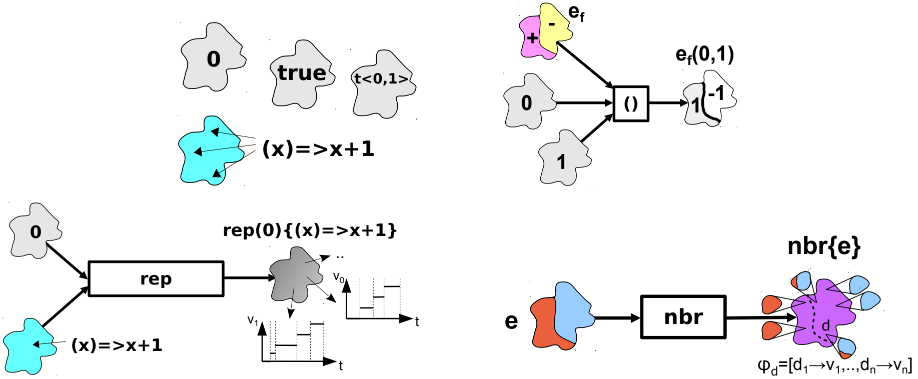
- `v` includes numbers, booleans, strings,..
  ..tuples/vectors/maps/any-ADT (of expressions)
- `f` is a user-defined function (the key aggregate computing abstraction)
- `o` is a built-in local operator (pure math, local sensors,..)

Intuition of global-level (denotational) semantics

The four main constructs at work

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

- $\texttt{e} ::= \ldots \mid \texttt{v} \mid \texttt{e}(\texttt{e}_1, \ldots, \texttt{e}_n) \mid \texttt{rep}(\texttt{e}_0)\{\texttt{e}\} \mid \texttt{nbr}\{\texttt{e}\}$



A mini-tutorial

```
// values
1: 1
2: 2 + 3
3: (10,20)
4: random()
```

```
// sensors
5: sense(1)
6: sense(1) ? 10 : 20
7: mid()
8: minHood(nbrRange)
```

```
// time-iteration
9: rep(0){ (x) => x + 1 }
10: rep(random()){ (x) => x }
```

```
// space-interaction
12: maxHood( nbr{ sense(1) } )
13: sumHood( nbr{ 1 } )
```

```
// space-time
14: rep(0){ (x) => max( sense(1), maxHood( nbr{ x } ) ) }
15: rep(Infinity) { (d) => sense(1) ? 0 : minHood( nbr{d} + 1 ) }
16: rep(Infinity) { (d) => sense(1) ? 0 : minHood( nbr{d} + nbrRange ) }
17: branch(sense(2)){Infinity}{ rep(Infinity) {
   (d) => sense(1) ? 0 : minHood( nbr{d} + nbrRange ) }}
```

A preview: the channel pattern

```
def gradient(source){ ;; reifying minimum distance from source
  rep(Infinity) { ;; distance is infinity initially
    (distance) => source ? 0 : minHood( nbr{distance} + nbrRange )
} }

def distance(source, dest) { ;; propagates minimum distance between source and dest
  snd(         ;; returning the second component of the pair
   rep(pair(Infinity, Infinity)) { ;; computing a field of pairs (distance,value)
    (distanceValue) => source ? pair(0, gradient(dest)) :
      minHood( ;; propagating as a gradient, using for first component of the pair
        pair(fst(nbr{distanceValue}) + nbrRange, snd(nbr{distanceValue})))
} ) }

def dilate(region, width) {  ;; a field of booleans
  gradient(region) < width
}

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

Field calculus, is it expressive?

Practically, we can express:

- complex spreading / aggregation / decay functions [3]
- spatial leader election, partitioning, consensus [4]
- distributed spatio-temporal sensing [5][6]
- splitting in parallel independent subprocesses [7][8]
- runtime verification of spatial properties [9][10]

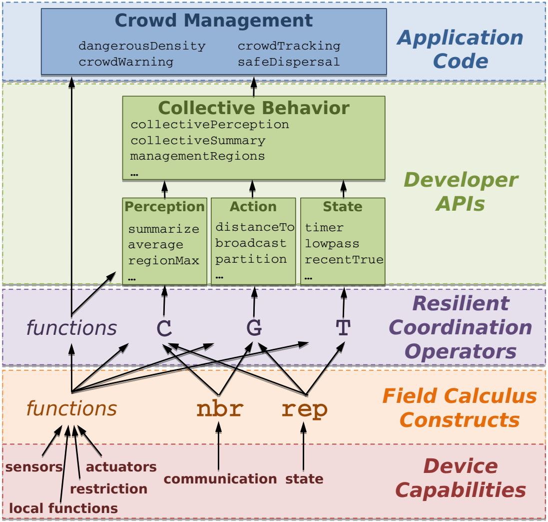
On its theory

- few selection of constructs evaluated, e.g., in XC calculus [11]
- universality [12]
- identification of a self-stabilising fragment [13]

---

[11] Giorgio Audrito et al. "Functional Programming for Distributed Systems with XC". In: *ECOOP 2022*. 2022, 20:1–20:28

Layers of Aggregate Computing



Tooling

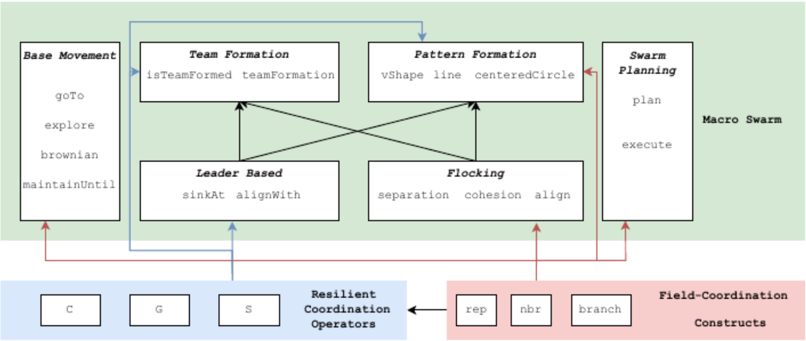
Several open-source projects

- ScaFi: a Scala-hosted DSL (`https://scafi.github.io/`)
- ScaFi-web: a Web playground for ScaFi (`https://github.com/scafi/scafi-web`)
- Alchemist: a simulator with ScaFi plugin (`https://alchemistsimulator.github.io/`)
- PulvReaKT: a platform for flexible deployment (`https://github.com/pulvreakt/pulvreakt`)

Open directions

- learning collective behaviour
- federated learning with aggregate computing
- programming/managing the cloud-edge continuum
- programming/managing swarms
- filling the gap with traditional program/concurrency approaches
- formally proving/enforcing properties

The MacroSwarm library



Involved people/groups

Main contributors

- Mirko Viroli, Univ. Bologna, Italy
  - Danilo Pianini, Roberto Casadei, Gianluca Aguzzi
- Ferruccio Damiani, Univ. Torino, Italy, and colleagues
- Jake Beal, IOWA University, USA, and colleagues

Other contributors

- Franco Zambonelli, Univ. Modena e Reggio Emilia, Italy
- Guido Salvaneschi, St.Gallen, Switzerland
- Simon Dobson, St.Andrews, UK
- Giancarlo Fortino, Univ. della Calabria, Italy
- Danny Weyns, Univ. Leuven, Belgium
- Volker Stolz, Univ. Oslo, Norway
- Lukas Esterle, Aarhus University, Denmark
- . . .

Bibliography I

[1] Roberto Casadei. "Macroprogramming: Concepts, State of the Art, and Opportunities of Macroscopic Behaviour Modelling". In: *ACM Comput. Surv.* (2023).

[2] Giorgio Audrito et al. "A Higher-Order Calculus of Computational Fields". In: *ACM Transactions on Computational Logic* 20.1 (Jan. 2019), 5:1–5:55.

[3] Jacob Beal, Danilo Pianini, and Mirko Viroli. "Aggregate Programming for the Internet of Things". In: *IEEE Computer* 48.9 (2015), pp. 22–30.

[4] Danilo Pianini et al. "Partitioned integration and coordination via the self-organising coordination regions pattern". In: *Future Generation Computer Systems* 114 (2021), pp. 44–68.

[5] Danilo Pianini et al. "Time-Fluid Field-Based Coordination through Programmable Distributed Schedulers". In: *Logical Methods in Computer Science* Volume 17, Issue 4 (Nov. 2021).

Bibliography II

[6] Giorgio Audrito et al. “Optimal resilient distributed data collection in mobile edge environments”. In: *Computers & Electrical Engineering* (2021), p. 107580. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2021.107580. URL: https://www.sciencedirect.com/science/article/pii/S0045790621005140.

[7] Roberto Casadei et al. “Engineering collective intelligence at the edge with aggregate processes”. In: *Engineering Applications of Artificial Intelligence* 97 (2021), p. 104081.

[8] Gianluca Aguzzi et al. “Dynamic Decentralization Domains for the Internet of Things”. In: *IEEE Internet Computing* 26.06 (2022), pp. 16–23. ISSN: 1941-0131. DOI: 10.1109/MIC.2022.3216753.

[9] Giorgio Audrito et al. “Adaptive distributed monitors of spatial properties for cyber?physical systems”. In: *Journal of Systems and Software* 175 (2021), p. 110908. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2021.110908. URL: https://www.sciencedirect.com/science/article/pii/S0164121221000054.

Bibliography III

[10] Giorgio Audrito et al. "Distributed runtime verification by past-CTL and the field calculus". In: *Journal of Systems and Software* 187 (2022), p. 111251. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2022.111251. URL: https://www.sciencedirect.com/science/article/pii/S0164121222000243.

[11] Giorgio Audrito et al. "Functional Programming for Distributed Systems with XC". In: *ECOOP 2022*. 2022, 20:1–20:28.

[12] Giorgio Audrito et al. "Space-Time Universality of Field Calculus". In: *Coordination Models and Languages - 20th IFIP WG 6.1 International Conference, COORDINATION 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018. Proceedings*. Ed. by Giovanna Di Marzo Serugendo and Michele Loreti. Vol. 10852. Lecture Notes in Computer Science. Springer, 2018, pp. 1–20. DOI: 10.1007/978-3-319-92408-3_1. URL: https://doi.org/10.1007/978-3-319-92408-3_1.

[13] Mirko Viroli et al. "Engineering Resilient Collective Adaptive Systems by Self-Stabilisation". In: *ACM Transaction on Modelling and Computer Simulation* 28.2 (Mar. 2018), 16:1–16:28.