Lifting the Fog with Aggregate Computing
..a programming model perspective

Mirko Viroli

Alma Mater Studiorum—Università di Bologna, Italy
mirko.viroli@unibo.it

"Through the Fog" Workshop
Pisa, 15/1/2016

(Internet-of-)Things are getting a bit messy (and foggy)

A plethora of programming models for "mobile/IoT applications"

- client side
  - single-device program: objects + functions + concurrency..
    ..threads/actors/futures/tasks/activities
  - device-centric interactions/protocols: using APIs for MoM/SOA/ad-hoc-communications
- server side
  - same interactions/protocols: MoM/SOA/ad-hoc-communications
  - storage by DB: OO, relational, NoSQL
  - coordination (orchestration, mediation, rules enactment)
  - situation recognition (online/offline, mining, business intelligence, stream processing)
- scalability in the server calls for cloudification
  - not really orthogonal to the whole programming model
  - it often dramatically affects system design

Fog computing has likely nice benefits

..but does not seemingly simplify things

(Internet-of-)Things are getting a bit messy (and foggy)

A plethora of programming models for "mobile/IoT applications"

- client side
  - single-device program: objects + functions + concurrency..
    ..threads/actors/futures/tasks/activities
  - device-centric interactions/protocols: using APIs for MoM/SOA/ad-hoc-communications
- server side
  - same interactions/protocols: MoM/SOA/ad-hoc-communications
  - storage by DB: OO, relational, NoSQL
  - coordination (orchestration, mediation, rules enactment)
  - situation recognition (online/offline, mining, business intelligence, stream processing)
- scalability in the server calls for cloudification
  - not really orthogonal to the whole programming model
  - it often dramatically affects system design

Fog computing has likely nice benefits

..but does not seemingly simplify things

Implications

Where programming effort ends up?

- programs of clients and servers highly depend on
  - the chosen platform / API / communication technology
  - the number and type of involved devices
- IoT systems tend to be very rigid, hard and costly to debug/maintain
- design and deployments hardly tolerate changes

The technological result

- systems can't scale with complexity of behaviour
- very few of the opportunities of large-scale IoT are taken
  - virtually any computational mechanism (sensing, actuation, processing, storage)..
  - ..could involve spontaneous, adaptive cooperation of large sets of devices!
- how many large-scale deployments of adaptive IoT systems around?
- where are the Collective Adaptive Systems?

Implications

Where programming effort ends up?

- programs of clients and servers highly depend on
  - the chosen platform / API / communication technology
  - the number and type of involved devices
- IoT systems tend to be very rigid, hard and costly to debug/maintain
- design and deployments hardly tolerate changes

The technological result

- systems can't scale with complexity of behaviour
- very few of the opportunities of large-scale IoT are taken
  - virtually any computational mechanism (sensing, actuation, processing, storage)..
  - ..could involve spontaneous, adaptive cooperation of large sets of devices!
- how many large-scale deployments of adaptive IoT systems around?
- where are the Collective Adaptive Systems?

What to do? A programming model perspective..

What do we lack in large-scale IoT sytems?

- the plain old platform-independent programming abstraction

  ⇒ fully grounding system design like objects did well.. in the past
  - delegating to the underlying platform virtually *all* deployment issues
  - automagically addressing non-functional issues (resilience, self-*)

The challenge

Just directly consider the worst scenario possible..

- zillion devices unpredictably moving in the environment
- heterogeneous displacement, pervasive sensing/actuation
- abstracting away from the possible multi-layered "server system" (fog++/cloud++) in background

  ⇒ but be ready to exploit the opportunities it creates!

What to do? A programming model perspective..

What do we lack in large-scale IoT sytems?

- the plain old platform-independent programming abstraction

  ⇒ fully grounding system design like objects did well.. in the past
  - delegating to the underlying platform virtually *all* deployment issues
  - automagically addressing non-functional issues (resilience, self-*)

The challenge

Just directly consider the worst scenario possible..

- zillion devices unpredictably moving in the environment
- heterogeneous displacement, pervasive sensing/actuation
- abstracting away from the possible multi-layered "server system" (fog++/cloud++) in background

  ⇒ but be ready to exploit the opportunities it creates!

What to do? A programming model perspective..

What do we lack in large-scale IoT sytems?

- the plain old platform-independent programming abstraction
  - ⇒ fully grounding system design like objects did well.. in the past
    - delegating to the underlying platform virtually *all* deployment issues
    - automagically addressing non-functional issues (resilience, self-*)

The challenge

Just directly consider the worst scenario possible..

- zillion devices unpredictably moving in the environment
- heterogeneous displacement, pervasive sensing/actuation
- abstracting away from the possible multi-layered "server system" (fog++/cloud++) in background
  ⇒ but be ready to exploit the opportunities it creates!

Let's try to program *that* "computational system"!

Abstract of the talk

Systems of interest: collective adaptive situated systems CASS

- (possibly very large scale) collective adaptive systems
- deployed in physical space (situated), i.e., IoT-oriented
- complex (open, dynamic, in need of much self-*)

Aggregate Computing

- The "good" computing/programming model for CASS
- It gives nice abstractions, promoting solid engineering principles
- Simple idea, few constructs, rather tractable, somehow *different*

This talk

1. Motivation and idea of aggregate computing
2. Some semi-technicalities and overview of results
3. State of toolchain and perspectives on platforms and fog

Abstract of the talk

Systems of interest: collective adaptive situated systems CASS

- (possibly very large scale) collective adaptive systems
- deployed in physical space (situated), i.e., IoT-oriented
- complex (open, dynamic, in need of much self-*)

Aggregate Computing

- The "good" computing/programming model for CASS
- It gives nice abstractions, promoting solid engineering principles
- Simple idea, few constructs, rather tractable, somehow *different*

This talk

1. Motivation and idea of aggregate computing
2. Some semi-technicalities and overview of results
3. State of toolchain and perspectives on platforms and fog

Abstract of the talk

Systems of interest: collective adaptive situated systems CASS

- (possibly very large scale) collective adaptive systems
- deployed in physical space (situated), i.e., IoT-oriented
- complex (open, dynamic, in need of much self-*)

Aggregate Computing

- The "good" computing/programming model for CASS
- It gives nice abstractions, promoting solid engineering principles
- Simple idea, few constructs, rather tractable, somehow *different*

This talk

1. Motivation and idea of aggregate computing
2. Some semi-technicalities and overview of results
3. State of toolchain and perspectives on platforms and fog

Outline

1. Aggregate Computing
2. Field Calculus
3. Platform support
4. Field Engineering

Outline

An example opportunity for IoT-based CASS..

Gathering local context

Sensing global patterns of data

Crowd Detection

Crowd Anticipation

Crowd-aware Steering

Crowd dispersal

Crowd evacuation upon alerts

Broad research challenges

Computational/programming model for these services

- Programming as: "describing the problem, not hacking the solution!"
- Hiding complexity and resiliency "under-the-hood"
- How computation carries on is hidden as well, and intrinsically self-*

Grounding an effective tool-chain

- languages, compilers, simulators, scalable execution platforms

Supporting solid engineering principles

- checking/enacting functional/non-functional correctness
- supporting reuse of patterns, substitutability, compositionality

Chasing the true issue

- we should fully escape the single "device" abstraction

Broad research challenges

Computational/programming model for these services

- Programming as: "describing the problem, not hacking the solution!"
- Hiding complexity and resiliency "under-the-hood"
- How computation carries on is hidden as well, and intrinsically self-*

Grounding an effective tool-chain

- languages, compilers, simulators, scalable execution platforms

Supporting solid engineering principles

- checking/enacting functional/non-functional correctness
- supporting reuse of patterns, substitutability, compositionality

Chasing the true issue

- we should fully escape the single "device" abstraction

Broad research challenges

Computational/programming model for these services

- Programming as: "describing the problem, not hacking the solution!"
- Hiding complexity and resiliency "under-the-hood"
- How computation carries on is hidden as well, and intrinsically self-*

Grounding an effective tool-chain

- languages, compilers, simulators, scalable execution platforms

Supporting solid engineering principles

- checking/enacting functional/non-functional correctness
- supporting reuse of patterns, substitutability, compositionality

Chasing the true issue

- we should fully escape the single "device" abstraction

Approaches to "group interaction in space"

Survey of past approaches [Beal et.al., 2013]

- *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
- *Foundation* – giving linguistic means for group interactions in space
  3π, Shape Calculus, bi-graphs, KLAIM, $\sigma\tau$-linda, SCEL
- *Spatial computing* – program space-time behaviour of systems
  Proto, MGS

Our approach

- Combining the above efforts of "macro" programming
- Taking some of those ideas to the extreme consequences

Approaches to "group interaction in space"

Survey of past approaches [Beal et.al., 2013]

- *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
- *Foundation* – giving linguistic means for group interactions in space
  3π, Shape Calculus, bi-graphs, KLAIM, $\sigma\tau$-linda, SCEL
- *Spatial computing* – program space-time behaviour of systems
  Proto, MGS

Our approach

- Combining the above efforts of "macro" programming
- Taking some of those ideas to the extreme consequences

Aggregate programming at [IEEE Computer 48(9), 2015]

Computer | Previous Page | Contents | Zoom in | Zoom out | Front Cover | Search Issue | Next Page

Computer

MULTIMEDIA

16 GUEST EDITORS' INTRODUCTION
Activating the Internet of Things
ROY WANT AND SCHAHRAM DUSTDAR

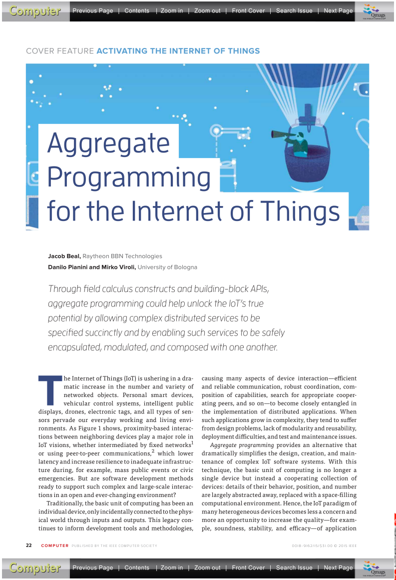
SEPTEMBER 2015
FEATURES

22 Aggregate Programming for the Internet of Things
JACOB BEAL, DANILO PIANINI, AND MIRKO VIROLI

32 Design and Deployment of an IoT Application-Oriented Testbed
LAURA BELLI, SIMONE CIRANI, LUCA DAVOLI, ANDREA GORRIERI, MIRKO MANCIN, MARCO PICONE, AND GIANLUIGI FERRARI

42 Repurposing Web Analytics to Support the IoT
MATEUSZ MIKUSZ, SARAH CLINCH, RACHEL JONES, MIKE HARDING, CHRISTOPHER WINSTANLEY, AND NIGEL DAVIES

Computer | Previous Page | Contents | Zoom in | Zoom out | Front Cover | Search Issue | Next Page



Computer | Previous Page | Contents | Zoom in | Zoom out | Front Cover | Search Issue | Next Page

COVER FEATURE **ACTIVATING THE INTERNET OF THINGS**

Aggregate Programming for the Internet of Things

Jacob Beal, Raytheon BBN Technologies
Danilo Pianini and Mirko Viroli, University of Bologna

Through field calculus constructs and building-block APIs, aggregate programming could help unlock the IoT's true potential by allowing complex distributed services to be specified succinctly and by enabling such services to be safely encapsulated, modulated, and composed with one another.

The Internet of Things (IoT) is ushering in a dramatic increase in the number and variety of networked objects. Personal smart devices, vehicular control systems, intelligent public displays, drones, electronic tags, and all types of sensors pervade our everyday working and living environments. As Figure 1 shows, proximity-based interactions between neighboring devices play a major role in IoT visions, whether intermediated by fixed networks[1] or using peer-to-peer communications,[2] which lower latency and increase resilience to inadequate infrastructure during, for example, mass public events or civic emergencies. But are software development methods ready to support such complex and large-scale interactions in an open and ever-changing environment?

Traditionally, the basic unit of computing has been an individual device, only incidentally connected to the physical world through inputs and outputs. This legacy continues to inform development tools and methodologies, causing many aspects of device interaction—efficient and reliable communication, robust coordination, composition of capabilities, search for appropriate cooperating peers, and so on—to become closely entangled in the implementation of distributed applications. When such applications grow in complexity, they tend to suffer from design problems, lack of modularity and reusability, deployment difficulties, and test and maintenance issues.

Aggregate programming provides an alternative that dramatically simplifies the design, creation, and maintenance of complex IoT software systems. With this technique, the basic unit of computing is no longer a single device but instead a cooperating collection of devices: details of their behavior, position, and number are largely abstracted away, replaced with a space-filling computational environment. Hence, the IoT paradigm of many heterogeneous devices becomes less a concern and more an opportunity to increase the quality—for example, soundness, stability, and efficacy—of application

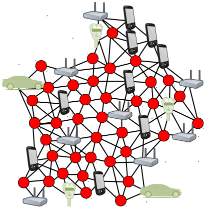
22 COMPUTER PUBLISHED BY THE IEEE COMPUTER SOCIETY 0018-9162/15/$31.00 © 2015 IEEE

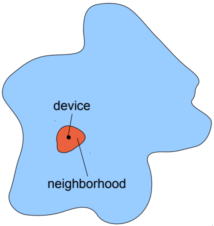
Computer | Previous Page | Contents | Zoom in | Zoom out | Front Cover | Search Issue | Next Page

Manifesto of aggregate computing

Motto: program the aggregate, not individual devices!

1. The reference computing machine
   ⇒ an aggregate of devices as single "body", fading to the actual *space*
2. The reference elaboration process
   ⇒ atomic manipulation of a collective data structure (a field)
3. The actual networked computation
   ⇒ a proximity-based self-org system hidden "under-the-hood"



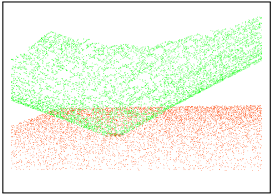


Outline

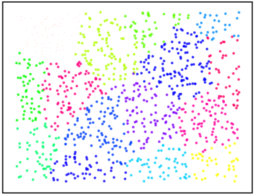
Computational Fields [Mamei et.al., 2009, Beal et.al., 2013]

Traditionally a map: *Space* $\mapsto$ *Values*

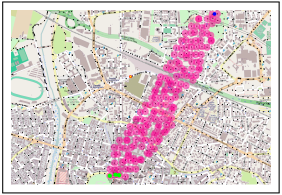
- possibly: evolving over time, dynamically injected, stabilising
- smoothly adapting to very heterogeneous domains
- more easily "understood" on continuous and flat spatial domains
- ranging to: booleans, reals, vectors, functions



real-valued gradient in 3D



numeric partition in 2D



boolean channel in 2D

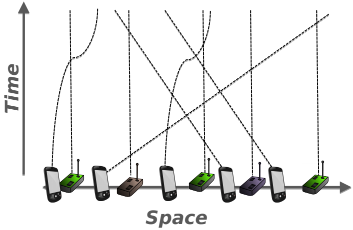
(Computational) Fields revisited [IEEE Computer 48(9), 2015]

A map: *DeviceSet* $\times$ *Space* $\times$ *Time* $\mapsto$ *ValueSet*

- *event E*: a triple $\langle \delta, t, p \rangle$ – device δ, "firing" at time t in position p
- *domain D*: a coherent set of events (devices cannot move too fast)
- *field* $\phi : D \mapsto V$: a map from events to *field values*

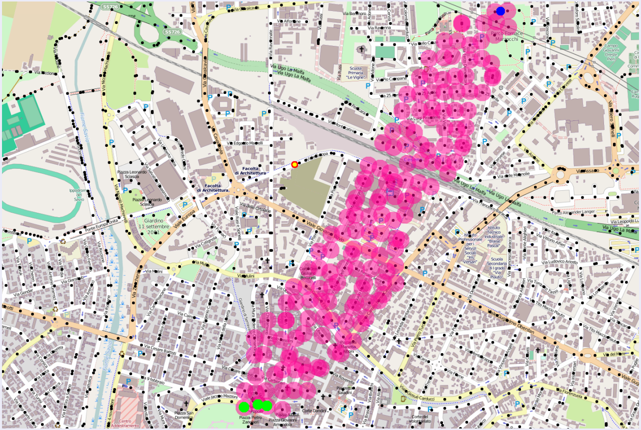
Early intuition: often one will think at fields that..

- "converge" with density of events, and lose track of device identities
- eventually (in time) reach a fixpoint
- so, you can draw (and reason/design) in 2D



The “channel” example: computing a redundant route

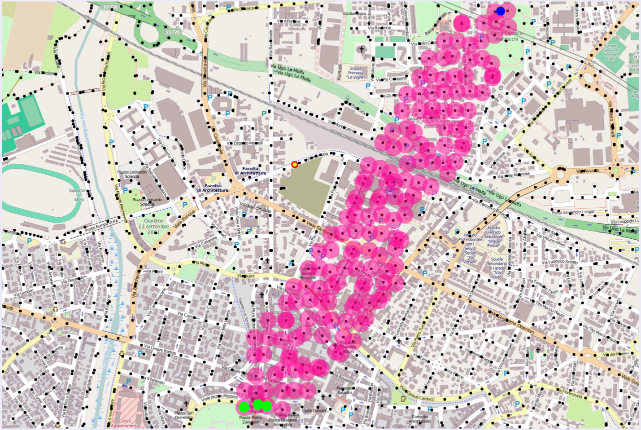
How would you program it?



how could a program be platform-independent,
unaware of global map, resilient to changes, faults,..

The “channel” example: computing a redundant route

How would you program it?



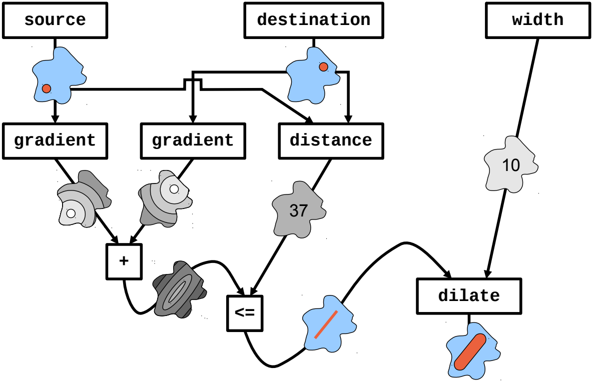
how could a program be platform-independent,
unaware of global map, resilient to changes, faults,..

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



Field calculus [Damiani & Viroli & Beal & Pianini, FORTE2015]

Key idea

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

Syntax (slightly refactored, semi-formal version of FORTE's)

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

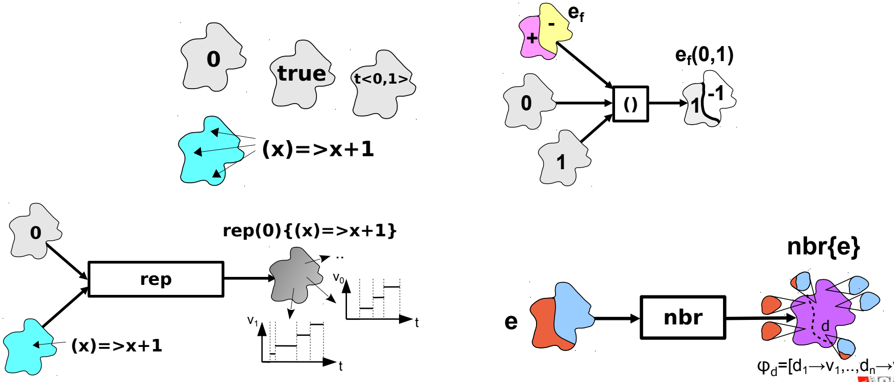
Few explanations

- v includes numbers, booleans, strings,..
  ..tuples/vectors/maps/any-ADT (of expressions)
- f is a user-defined function
- o is a built-in functional operator (mostly pure math or a sensor)

Intuition of global-level semantics

The four main constructs at work
$\Rightarrow$ values, application, evolution, and interaction – in aggregate guise

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



Intuition of field-level semantics

Value v

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

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

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

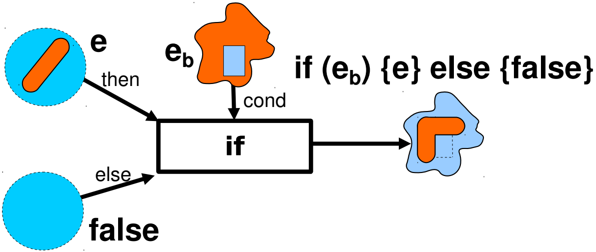
Repetition $\texttt{rep}(e_0)\{e_\lambda\}$

- 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 $\texttt{nbr}\{e\}$

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

The restriction trick: branching behaviour



if as a space-time branching construct

if(e-bool){e-then}else{e-else}

≈

(e-bool ? ()=>{e-then} : ()=>{e-else})()

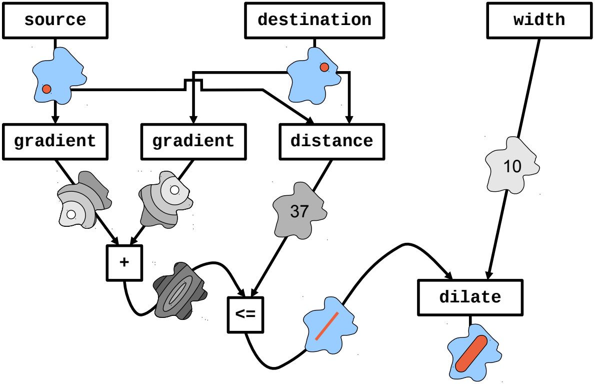
More advanced patterns

- spread code, in different versions in different regions
- have different regions/device run different programs

Aggregate programming as a functional approach

Functionally composing fields

- ...so, is field calculus language practical/expressive?



The channel pattern

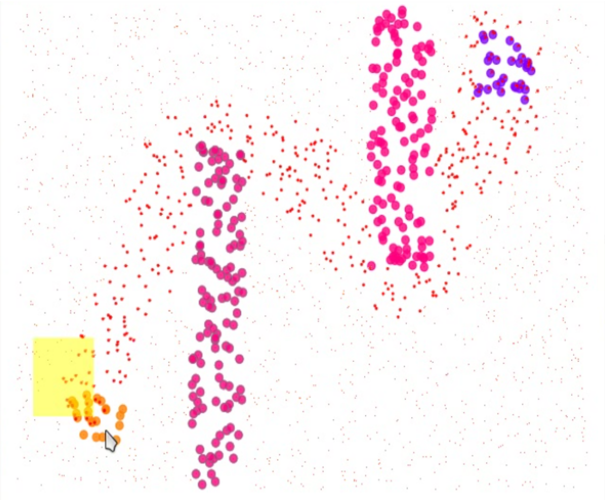
```
def gradient(source){ ;; reifying minimum distance from source
  rep(Infinity) { ;; distance is infinity initially
    (distance) => source ? 0 : minHood( nbr{distance} + nbrRange )
} }

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
}

;; Here the ''aggregate'' nature of our approach gets revealed
def channel(source, dest, width) {
  dilate( gradient(source) + gradient(dest) <= distance(source,dest), width )
}
```

Symbols

Builtin functions exploited

- ?: — Java-like (though, call-by-value) ternary operator
- nbrRange — maps each device to a neighbour field of estimated distances
- minHood — in each device, collapse a neighbour field into its minimum value
- sumHood — in each device, collapse a neighbour field into sum of values
- *,-,*,/,>,... — usual math, applied also pointwise to fields
- pair,fst,snd — construction/selection for pairs

Channel in action: note inherent self-stabilisation



On expressiveness of the field calculus

Practically, we can express:

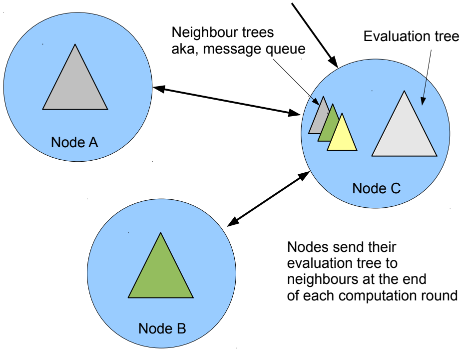
- complex spreading / aggregation / decay functions
- spatial leader election, partitioning, consensus
- distributed spatio-temporal sensing and situation recognition
- dynamic deployment/spreading of code (via lambda)
- implicit/explicit device selection of what code execute
- "collective teams" forming based on the selected code

Outline

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



Single-round operational semantics – pulverization

Main run-time structures

ϕ	::=	$\{\overline{\delta} \mapsto \overline{\mathtt{l}}\}$	field value: mapping nodes to local values
$\mathtt{v}$	::=	$\mathtt{l} \mid \phi$	values: local values or field values
θ	::=	$\mathtt{v}\,(\overline{\theta})$	value-tree: an ordered tree of values
Θ	::=	$\{\overline{\delta} \mapsto \overline{\theta}\}$	value-tree environment: neighbours info

Big-step operational semantics judgment

$$\delta; \Theta \vdash \mathsf{e} \Downarrow \theta$$

Read: at device δ, with environment Θ, evaluation of e gives result θ

$\Rightarrow$ Namely, computation takes input Θ and produces output θ

..an orthogonal "network-level" LTS completes the operational semantics

Current formalisation (under progressive shrinking..)

Auxiliary functions:

$\rho(\mathtt{v}(\overline{\theta})) = \mathtt{v}$

$\pi_i(\mathtt{v}(\theta_1,\ldots,\theta_n)) = \theta_i \quad \text{if } 1 \le i \le n \qquad \pi^{\ell,n}(\mathtt{v}(\theta_1,\ldots,\theta_{n+2})) = \theta_{n+2} \quad \text{if } \rho(\theta_{n+1}) = \ell$

$\pi_i(\theta) = \bullet \quad \text{otherwise} \qquad \pi^{\ell,n}(\theta) = \bullet \quad \text{otherwise}$

$$\text{For } aux \in \rho, \pi_i, \pi^{\ell,n}: \begin{cases} aux(\delta \mapsto \theta) &= \delta \mapsto aux(\theta) & \text{if } aux(\theta) \neq \bullet \\ aux(\delta \mapsto \theta) &= \bullet & \text{if } aux(\theta) = \bullet \\ aux(\Theta, \Theta') &= aux(\Theta), aux(\Theta') \end{cases}$$

$args(\mathtt{d}) = \overline{\mathtt{x}} \quad \text{if } \mathtt{def\ d}(\overline{\mathtt{x}})\ \{\mathtt{e}\} \qquad body(\mathtt{d}) = \mathtt{e} \quad \text{if } \mathtt{def\ d}(\overline{\mathtt{x}})\ \{\mathtt{e}\}$

$args((\overline{\mathtt{x}}) \Rightarrow \mathtt{e}) = \overline{\mathtt{x}} \qquad body((\overline{\mathtt{x}}) \Rightarrow \mathtt{e}) = \mathtt{e}$

Rules for expression evaluation: $\boxed{\delta;\Theta \vdash \mathtt{e} \Downarrow \theta}$

$$\text{[E-LOC]}\ \frac{}{\delta;\Theta \vdash \ell \Downarrow \ell()} \qquad \text{[E-FLD]}\ \frac{\phi' = \phi|_{\mathbf{dom}(\Theta)\cup\{\delta\}}}{\delta;\Theta \vdash \phi \Downarrow \phi'()}$$

$$\text{[E-DATA]}\ \frac{\begin{array}{l}\mathtt{c}\langle \mathtt{e}_1,\ldots,\mathtt{e}_m\rangle \text{ not a value} \\ \delta;\pi_1(\Theta) \vdash \mathtt{e}_1 \Downarrow \theta_1 \quad \cdots \quad \delta;\pi_m(\Theta) \vdash \mathtt{e}_m \Downarrow \theta_m \qquad \ell = \mathtt{c}\langle \rho(\theta_1),\ldots,\rho(\theta_m)\rangle\end{array}}{\delta;\Theta \vdash \mathtt{c}\langle \mathtt{e}_1,\ldots,\mathtt{e}_m\rangle \Downarrow \ell(\theta_1,\ldots,\theta_n)}$$

$$\text{[E-B-APP]}\ \frac{\begin{array}{l}\delta;\pi_{n+1}(\Theta) \vdash \mathtt{e}_{n+1} \Downarrow \theta_{n+1} \qquad \rho(\theta_{n+1}) = \mathtt{b} \\ \delta;\pi_1(\Theta) \vdash \mathtt{e}_1 \Downarrow \theta_1 \quad \cdots \quad \delta;\pi_n(\Theta) \vdash \mathtt{e}_n \Downarrow \theta_n \qquad \mathtt{v} = \varepsilon^{\mathtt{b}}_{\delta;\Theta}(\rho(\theta_1),\ldots,\rho(\theta_n))\end{array}}{\delta;\Theta \vdash \mathtt{e}_{n+1}(\mathtt{e}_1,\ldots,\mathtt{e}_n) \Downarrow \mathtt{v}(\theta_1,\ldots,\theta_{n+1})}$$

$$\text{[E-D-APP]}\ \frac{\begin{array}{l}\delta;\pi_{n+1}(\Theta) \vdash \mathtt{e}_{n+1} \Downarrow \theta_{n+1} \qquad \rho(\theta_{n+1}) = \ell \qquad args(\ell) = \mathtt{x}_1,\ldots,\mathtt{x}_n \\ \delta;\pi_1(\Theta) \vdash \mathtt{e}_1 \Downarrow \theta_1 \quad \cdots \quad \delta;\pi_n(\Theta) \vdash \mathtt{e}_n \Downarrow \theta_n \qquad body(\ell) = \mathtt{e} \\ \delta;\pi^{\ell,n}(\Theta) \vdash \mathtt{e}[\mathtt{x}_1 := \rho(\theta_1) \quad \ldots \quad \mathtt{x}_n := \rho(\theta_n)] \Downarrow \theta_{n+2} \qquad \mathtt{v} = \rho(\theta_{n+2})\end{array}}{\delta;\Theta \vdash \mathtt{e}_{n+1}(\mathtt{e}_1,\ldots,\mathtt{e}_n) \Downarrow \mathtt{v}(\theta_1,\ldots,\theta_{n+2})}$$

$$\text{[E-NBR]}\ \frac{\Theta_1 = \pi_1(\Theta) \qquad \delta;\Theta_1 \vdash \mathtt{e} \Downarrow \theta_1 \qquad \phi = \rho(\Theta_1)[\delta \mapsto \rho(\theta_1)]}{\delta;\Theta \vdash \mathtt{nbr}\{\mathtt{e}\} \Downarrow \phi(\theta_1)}$$

$$\text{[E-REP]}\ \frac{\ell_0 = \begin{cases} \rho(\Theta(\delta)) & \text{if } \Theta \neq \emptyset \\ \ell & \text{otherwise} \end{cases} \qquad \delta;\pi_1(\Theta) \vdash \mathtt{e}[\mathtt{x} := \ell_0] \Downarrow \theta_1 \qquad \ell_1 = \rho(\theta_1)}{\delta;\Theta \vdash \mathtt{rep}(\ell)\{(\mathtt{x}) \Rightarrow \mathtt{e}\} \Downarrow \ell_1(\theta_1)}$$

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 $e(e_1, \ldots, e_n)$

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

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

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

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

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

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 $\mathtt{e}(\mathtt{e}_1, \ldots, \mathtt{e}_n)$

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

Repetition $\mathtt{rep}(\mathtt{e}_0)\{\mathtt{e}_\lambda\}$

- if a previous value-tree of mine is available, evaluates $\mathtt{e}_\lambda$ on its root
- otherwise, evaluates $\mathtt{e}_0$

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

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

Operational semantics as blueprint for platform support

Requirements

- a notion of neighbourhood must be defined — wireless connectivity, physical proximity..
- nodes execute in asynchronous rounds, and emit a "round result"
- a node need 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!*

Operational semantics as blueprint for platform support

Requirements

- a notion of neighbourhood must be defined — wireless connectivity, physical proximity..
- nodes execute in asynchronous rounds, and emit a "round result"
- a node need 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!*

Natural implementations

P2P

- devices see neighbours, and directly broadcast messages (ad-hoc wifi)

$\Rightarrow$ in principle possible, but interferences might be an issue

Server-mediated communication

- a single server mediates communications
- holding topology info and enacting a fully-custom topology

$\Rightarrow$ not hard to handle 10K devices firing at 1Hz

Dealing with (mobile) cloud

Cloud implementation

- we use devices only as physical containers of sensors / actuators
- the server as mediator of communications *and* running computations
- cloudification is easy due to our *pulverization* semantics
- a cloud-DB holds field maps, rounds can be executed in clusters

Advantages of the conceptual concentration

- vertical optimisation: decide what to compute in the cloud and what on device/edge
- horizontal optimisation: decide which device computation can be slowed down

⇒ both explicit (programmed) or implicit (dynamically activated)

Dealing with fog computing

Explicit approach: edge devices as part of the "aggregate machine"

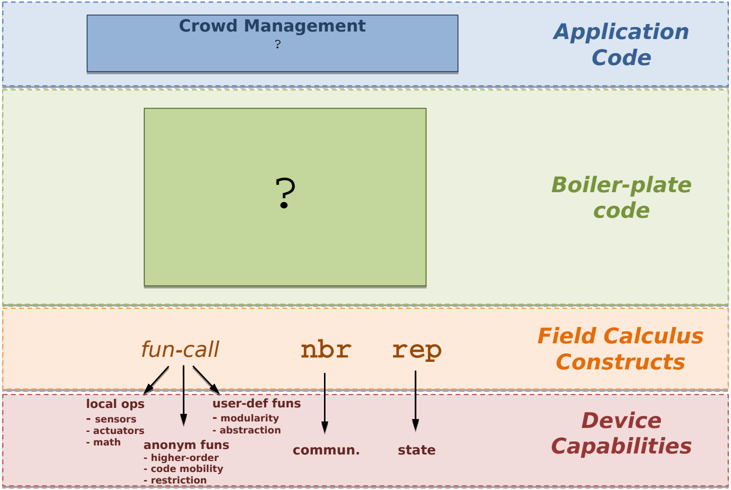
- edge devices are just like any other device
- the programmer takes care of use them for specific tasks
  - typically: leaders/aggregators of distributed sensing/decision making
  - they could be nodes with higher round frequency and connectivity

Implicit approach: edge devices are part of the underlying platform

- using edge devices as sort of vertical optimisation
- when too much computation/communication resources are required, the platform starts delegating to the edges, then to cloud

Outline

How to scale with complexity?



Survey of recent efforts

Attacking a multifaceted problem

- Properties (self-stabilisation, density-independence, universality)
- Tools (languages, simulators, platforms)
- Libraries (reusable components, correctness, raising abstraction)

Properties

Self-stabilisation

- Def: If environment and inputs stop changing, computation reaches a fixpoint
- Identified a rather large subset of the language [SASO-2015]

Density independence

- Def: the denotation of an expression computation converges with the space-time density of events
- Identified a (small) subset of the language [Submitted]

Universality

- Def: for any causal field evolution Φ over arbitrary domain D (even continuous), there exists an expression whose denotation converges to Φ as the domain converges to D
- Field calculus is arguably universal [SCW-2014]

Self-stabilisation for computational fields

Definition of self-stabilising field expression e

- Given an environment: inputs (sensor fields) and network topology

$\Rightarrow$ computing e results in a stable unique field in finite time

Implications

- After fixing a topology, a field computation is an I/O problem

$\Rightarrow$ Transient env. changes do not affect the result of computation

Self-stabilisation is undecidable, but can identify sufficient conditions

Self-stabilisation for computational fields

Definition of self-stabilising field expression *e*

- Given an environment: inputs (sensor fields) and network topology

⇒ computing *e* results in a stable unique field in finite time

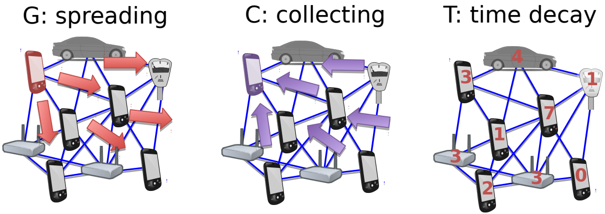
Implications

- After fixing a topology, a field computation is an I/O problem

⇒ Transient env. changes do not affect the result of computation

Self-stabilisation is undecidable, but can identify sufficient conditions

GCT as self-stabilising combinators set



Functions

G: Spreads and en-route computes information outwards a source

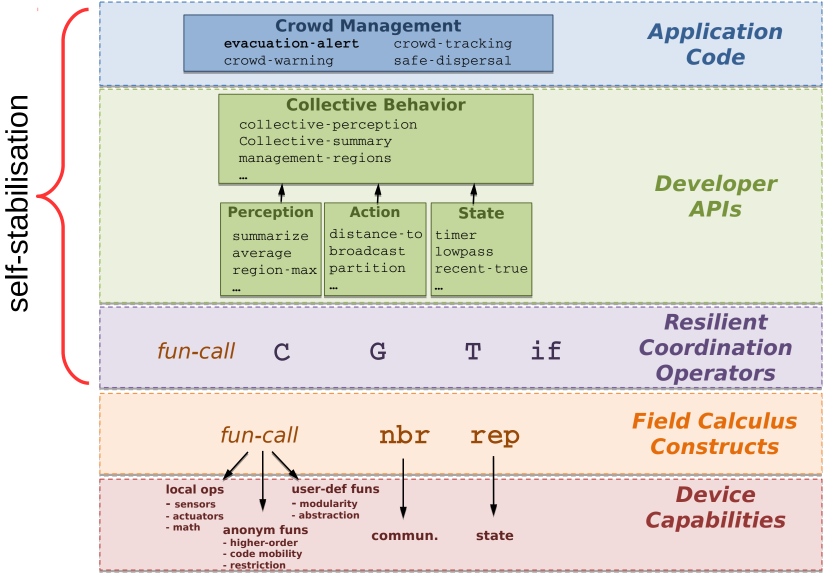
C: Collects and en-route aggregates information inwards a destination

T: Locally iterates computations until a termination

Observations

- The three blocks can pragmatically replace `nbr` and `rep`
- Towards a GCT-based system of libraries

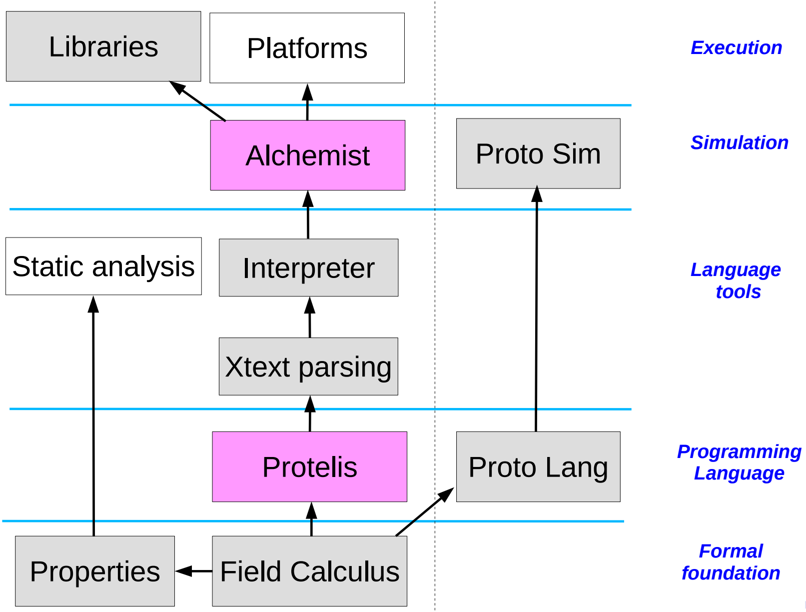
Libraries (each function with a 1-5 lines body)



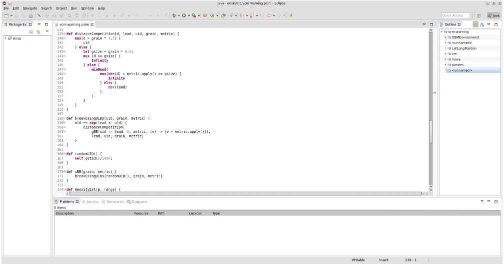
Crowd estimation service, on top of APIs [Fruin, 1971]

```
;; Density Estimation: density of neighbours within a short 3.0mt range
def densityEstimation() {
  countHood(nbrRange < 3.0) / (3.0 * 3.0 * 3.14)
}
;; More then 2.17 density and 'threshold' overcame in a 'partition' region
def dangerousDensity(partition, threshold, range) {
  average(partition, densityEstimation()) > 2.17   ;; Fruin LoS
  &&
  count(partition) > threshold ;; and, many people..
}
;; Crowd levels:
;; Level 1 (low): density greater than 1.08 in last 60 seconds
;; Level 2 (high): in a 30mt-range partition, L1 persons are > 300 with density > 2.18
;; Level 0 (none): others
def crowdTracking(){
  if (recentlyTrue(densityEstimation() > 1.08, 60) { ;; note restriction here..
      dangerousDensity(randomPartition(30), 300) ? high : low
  } else {
      none
  }
}
```

Current tool-chain for aggregate computing



Protelis + Alchemist [SAC-2015]



Protelis language: `http://protelis.org/`

- Field calculus in disguised and full-blown version
- Java-like syntax and Java API integration

Alchemist simulator: `http://alchemist.apice.unibo.it/`

- A general-purpose simulator with pluggable specification language
- XText/Eclipse integration
- Support from working with Maps, Traces, Paths, Movement models

Current/future investigations

Field calculus

- fields as processes, neighbours as ensembles, dealing with streams
- universality, relation with continuous space-time, self-stabilisation
- model checking with abstractions for large-scale systems

Language and programming

- Protelis released, and pluggable into Alchemist simulator
- Scala library support to be released soon

Platform level

- single-server general-purpose coordinator (RESTlets + RedisDB)
- cloud support (experiments with Apache Kafka & Storm)

Conclusions

Aggregate Computing

- a new paradigm for developing large-scale situated systems
- a bunch of results and tools emerged, many to come
- we're always eager to find new collaborations!

Messages for the fog people

- evaluate our toolchain for location-aware applications
- think at a fog support that does not impact programming
- try to think at systems as aggregates, it is worthy!

Acknowledgments

- Jacob Beal (BBN, USA)
- Ferruccio Damiani (UNITO)
- Danilo Pianini (UNIBO)

References I

[IEEE Computer 48(9), 2015] Beal, J., Pianini, D, and Viroli, M. (2015).
Aggregate programming for the Internet of Things.
IEEE Computer, 48(9) 2015.

[Beal et.al., 2013] Beal, J., Dulman, S., Usbeck, K., Viroli, M., and Correll, N. (2013).
Organizing the aggregate: Languages for spatial computing.
In Mernik, M., editor, *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, chapter 16, pages 436–501. IGI Global.
A longer version available at: `http://arxiv.org/abs/1202.5509`.

[SCW-2014] Beal, J., Viroli, M., and Damiani, F. (2014).
Towards a unified model of spatial computing.
In *7th Spatial Computing Workshop (SCW 2014)*, AAMAS 2014, Paris, France.

[Fruin, 1971] Fruin, J. (1971).
Pedestrian Planning and Design.
Metropolitan Association of Urban Designers and Environmental Planners.

[Mamei et.al., 2009] Mamei, M. and Zambonelli, F. (2009).
Programming pervasive and mobile computing applications: The tota approach.
ACM Transactions on Software Engineering and Methodologies, 18(4).

[SAC-2015] Pianini, D., Beal, J., and Viroli, M. (2015).
Practical aggregate programming with PROTELIS.
In *ACM Symposium on Applied Computing (SAC 2015)*.
To appear.

References II

[Damiani & Viroli & Beal & Pianini, FORTE2015] Damiani, F., Viroli, M., Pianini, D., and Beal, J. (2015).
Code mobility meets self-organisation: a higher-order calculus of computational fields.
In *Formal Techniques for Distributed Objects, Components, and Systems*, volume 9039 of *LNCS*, pages 113–128. Springer.

[SASO-2015] Viroli, M., Beal, J., Damiani, F. and Pianini, D. (2015).
Efficient Engineering of Complex Self-Organising Systems by Self-Stabilising Fields
In *IEEE Conference on Self-Adaptive and Self-Organising Systems*.

[Beal, SAC2009] Beal, j. (2009).
Flexible self-healing gradients
In *ACM Symposium on Applied Computing*, pp. 1197–1201.