Aggregate Programming
Foundations and Tools

Mirko Viroli

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

Seminar at Università degli Studi di Urbino "Carlo Bo"
Corso di Laurea in Informatica Applicata
Urbino, 17/5/2016

The IoT is becoming a crowded and complex place..

Future and emerging Internet-of-things are witnessing..

- increasing availability of wearable / mobile / embedded / flying devices
- increasing availability of heterogeneous wireless connectivity
- increasing availability of computational resources (device/edge/cloud)
- increasing production/analysis of data, everywhere, anytime

$\Rightarrow$ business / security / privacy concerns will probably be drivers, too

The IoT is becoming a crowded and complex place..

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 local 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

Implications

Where programming effort ends up?

- programs of clients and servers highly depend on
  - the chosen platform / API / communication technology
  - the number, type, and displocation of involved devices

$\Rightarrow$ IoT systems tend to be very rigid, hard and costly to debug/maintain

$\Rightarrow$ 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 currently 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, type, and displocation of involved devices

$\Rightarrow$ IoT systems tend to be very rigid, hard and costly to debug/maintain

$\Rightarrow$ 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 currently 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

  $\Rightarrow$ 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”
  - whether with have fog++/cloud++ in background

  $\Rightarrow$ 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
  - $\Rightarrow$ 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"
  - whether with have fog++/cloud++ in background
  - $\Rightarrow$ 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"
  - whether with have 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

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

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

Outline

Outline

Let's follow standard "software engineering" process

Requirements and Analysis

- The customer does not mention "servers" or "connectivity"
- Different services to be implemented
- All in need of robustness at different levels
- Several common problems, to be "factored out"

Architectural design

- Depict strategies and abstractions in a platform-independent way
- Using concepts very near to the problem domain
- Identify common patterns

Detailed design and other stages

- Choose technologies, write APIs and component interfaces
- Implementation, Testing, Deployment

Let's follow standard "software engineering" process

Requirements and Analysis

- The customer does not mention "servers" or "connectivity"
- Different services to be implemented
- All in need of robustness at different levels
- Several common problems, to be "factored out"

Architectural design

- Depict strategies and abstractions in a platform-independent way
- Using concepts very near to the problem domain
- Identify common patterns

Detailed design and other stages

- Choose technologies, write APIs and component interfaces
- Implementation, Testing, Deployment

Let's follow standard "software engineering" process

Requirements and Analysis

- The customer does not mention "servers" or "connectivity"
- Different services to be implemented
- All in need of robustness at different levels
- Several common problems, to be "factored out"

Architectural design

- Depict strategies and abstractions in a platform-independent way
- Using concepts very near to the problem domain
- Identify common patterns

Detailed design and other stages

- Choose technologies, write APIs and component interfaces
- Implementation, Testing, Deployment

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

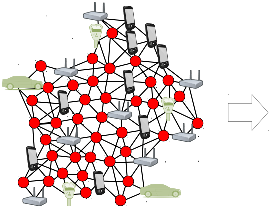
Our approach

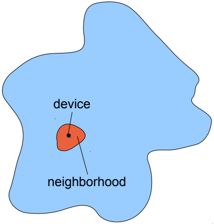
- Combining the above efforts of “macro” programming
- Taking some of those ideas to the extreme consequences

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"



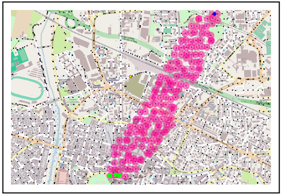


Outline

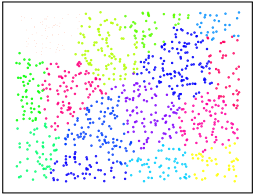
Computational Fields [Mamei et.al., 2009, Beal et.al., 2013]

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

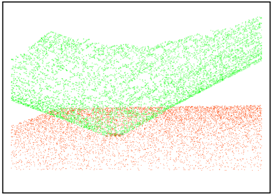
- possibly: evolving over time, dynamically injected, stabilising
- smoothly adapting to very heterogeneous domains
- more easily "understood" on continuous and flat spatial domains
- ranging to: booleans, reals, vectors, functions



boolean channel in 2D



numeric partition in 2D

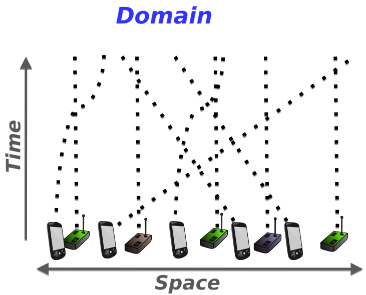


real-valued gradient in 3D

(Computational) Fields revisited

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

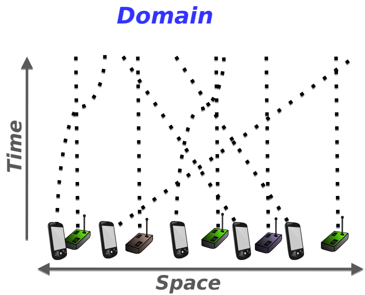
- *event* E: a triple $\langle \delta, t, p \rangle$ – device δ, "firing" at time t in position p
- *events domain* D: a coherent set of events (devices cannot move too fast)
- *field values* V: any data value

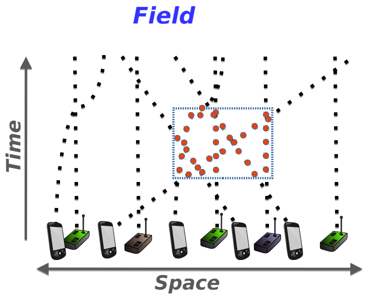


(Computational) Fields revisited

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

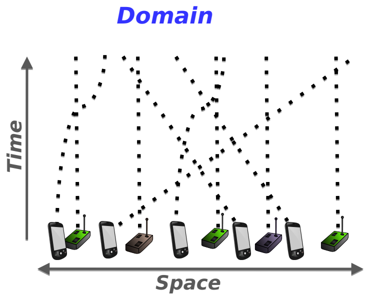


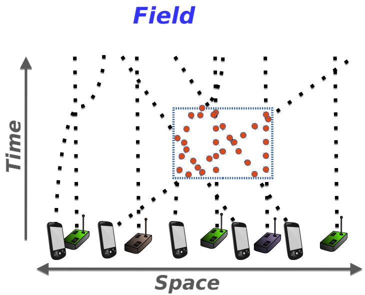


(Computational) Fields revisited

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

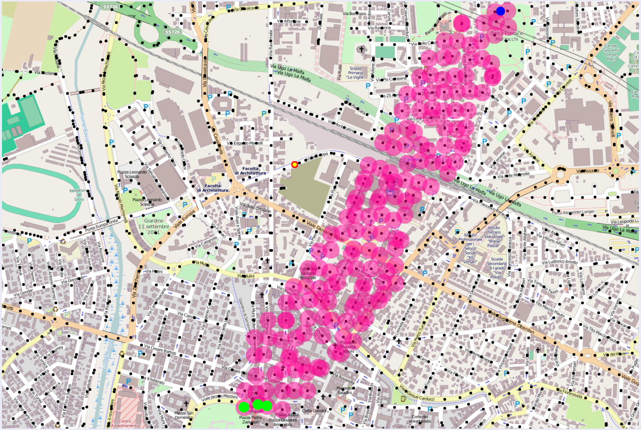




will later show only snaphots of fields in 2D space..

The “channel” example: computing a redundant route

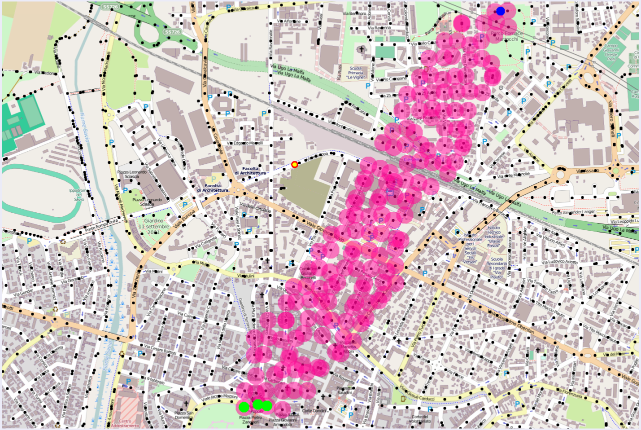
How would you program it?



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?



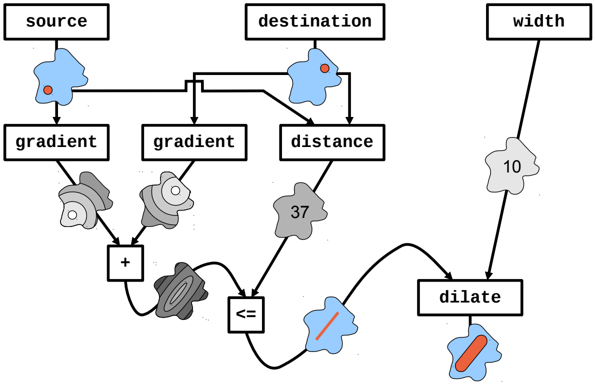
how could a program be platform-independent,
unaware of global map, resilient to changes, faults,..

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

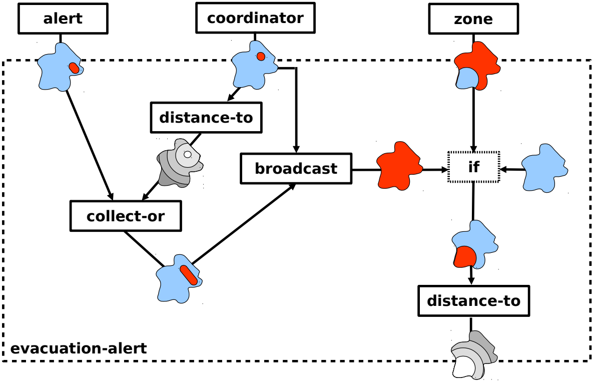


Crowd evacuation as a field computation

Computing by purely functional composition of space-time fields

- Inputs: sensor fields, Output: actuator field
- Computation is a pure function over fields (time embeds state!)

$\Rightarrow$ 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)

$$\texttt{e} ::= \texttt{x} \mid \texttt{v} \mid \texttt{e}(\texttt{e}_1, \ldots, \texttt{e}_n) \mid \texttt{rep}(\texttt{e}_0)\{\texttt{e}\} \mid \texttt{nbr}\{\texttt{e}\} \qquad \text{(expr)}$$

$$\texttt{v} ::= < \text{standard-values} > \mid \lambda \qquad \text{(value)}$$

$$\lambda ::= \texttt{f} \mid \texttt{o} \mid (\overline{\texttt{x}})\texttt{=>e} \qquad \text{(functional value)}$$

$$\texttt{F} ::= \texttt{def f}(\overline{\texttt{x}})\ \{\texttt{e}\} \qquad \text{(function definition)}$$

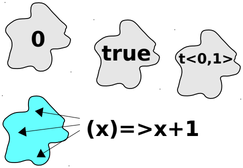
Few explanations

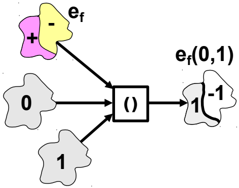
- v includes numbers, booleans, strings,..
  ..tuples/vectors/maps/any-ADT (of expressions)
- f is a user-defined function
- o is a built-in functional operator (mostly pure math or a sensor)

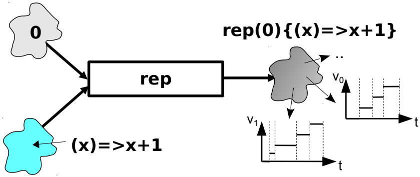
Intuition of global-level semantics

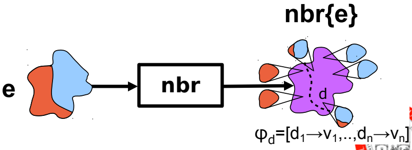
The four main constructs at work
$\Rightarrow$ values, application, evolution, and interaction – in aggregate guise

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









A mini-tutorial: functions, repetitions, neighbouring

```
1
2 + 3
pair(10,20)
random()
sense(1)
sense(1) ? 10 : 20
mid()
minHood(nbrRange)
```

```
rep(0){ (x) => x + 1 }
rep(random()){ (x) => x }
rep(0){ (x) => x + rep(random()){ (y) => y } }
```

```
maxHood( nbr{ sense(1) } )
sumHood( nbr{ 1 } )
```

```
rep(0){ (x) => max( sense(1), maxHood( nbr{ x } ) ) }
rep(Infinity) { (d) => sense(1) ? 0 : minHood( nbr{d} + 1 ) }
rep(Infinity) { (d) => sense(1) ? 0 : minHood( nbr{d} + nbrRange ) }
```

A mini-tutorial: functions, repetitions, neighbouring

```
1                            ;; values become constant fields
2 + 3                        ;; math is done infix
pair(10,20)                  ;; fst, snd to extract
random()                     ;; note iterative execution..
sense(1)                     ;; a boolean sensor
sense(1) ? 10 : 20           ;; muxing
mid()                        ;; unique identifiers
minHood(nbrRange)            ;; distance of closest neighbour
```

```
rep(0){ (x) => x + 1 }   ;; counting the number of rounds
rep(random()){ (x) => x }   ;; stable random
rep(0){ (x) => x + rep(random()){ (y) => y } }   ;; counting at different velocities
```

```
maxHood( nbr{ sense(1) } )   ;; maximum value of sensor in neighbours
sumHood( nbr{ 1 } )   ;; number of neighbours
```

```
rep(0){ (x) => max( sense(1), maxHood( nbr{ x } ) ) } ;; gossiping max of sense(1)
rep(Infinity) { (d) => sense(1) ? 0 : minHood( nbr{d} + 1 ) } ;; hop-count
rep(Infinity) { (d) => sense(1) ? 0 : minHood( nbr{d} + nbrRange ) } ;; gradient
```

Intuition of global-level semantics

Value $\mathtt{v}$

- A field constant in space and time, mapping any event to $\mathtt{v}$

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

- $\mathtt{e}$ 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: $\forall i$, $\lambda_i(\mathtt{e}_1, \ldots, \mathtt{e}_n)$ restricted in D_i

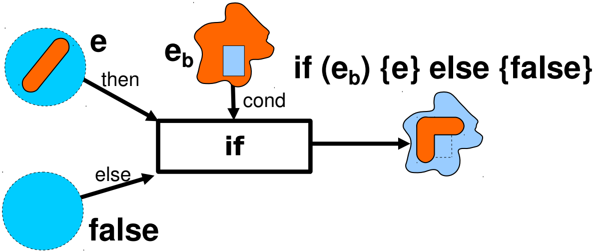
Repetition $\mathtt{rep}(\mathtt{e}_0)\{\mathtt{e}_\lambda\}$

- the value of $\mathtt{e}_0$ where the restricted domain "begins"
- elsewhere, unary function $\mathtt{e}_\lambda$ is applied to previous value at each device

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

- at each event gathers most recent value of $\mathtt{e}$ 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}

$\approx$

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

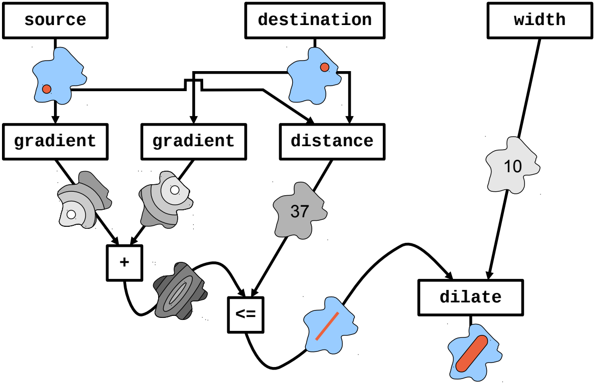
More advanced patterns

- 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

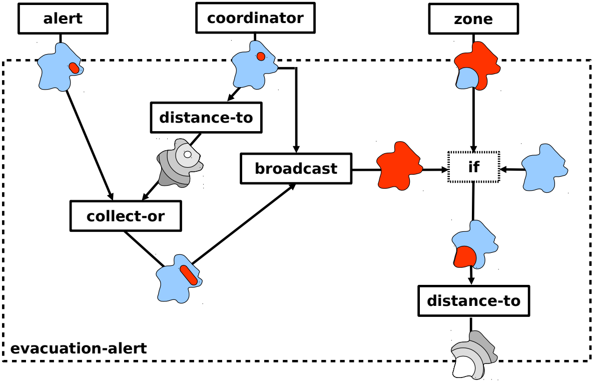
- ?: — Java-like (though, call-by-value) ternary operator
- nbrRange — maps each device to a neighbour field of estimated distances
- minHood — in each device, collapse a neighbour field into its minimum value
- sumHood — in each device, collapse a neighbour field into sum of values
- *,-,*,/,>,. . .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

Crowd evacuation as a field computation

Computing by purely functional composition of space-time fields

- Inputs: sensor fields, Output: actuator field
- Computation is a pure function over fields (time embeds state!)

$\Rightarrow$ for this to be practical/expressive we need a good programming language



Evacuation example

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

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

def collect-or (potential, value){;; Collects 'value' by descending 'potential', by 'or'
  rep(value){
     (v) => anyHood( nbr{find-parent potential()} = uid ? nbr{v} : false )
              or value
} }

def evacuation-alert (zone, coordinator, alert){
  distance-to(
     if(zone){false} else {
         broadcast(coordinator,collect-or(distance-to(coordinator),alert))})
}
```

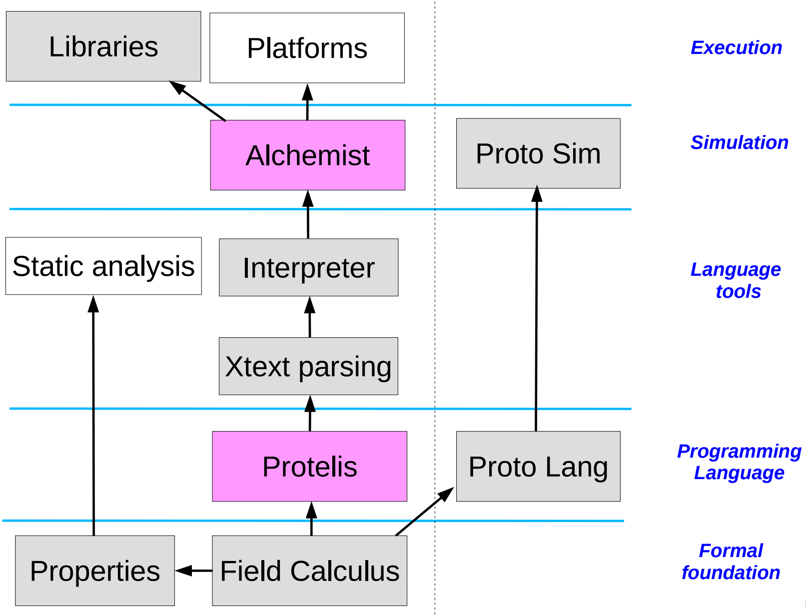
On expressiveness of the field calculus

Practically, we can express:

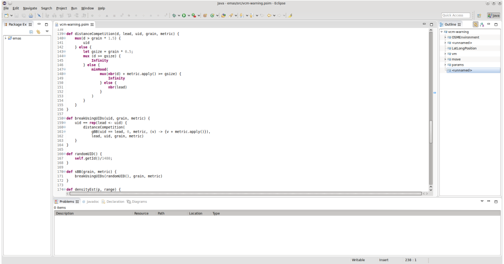
- complex spreading / aggregation / decay functions
- spatial leader election, partitioning, consensus
- distributed spatio-temporal sensing and situation recognition
- dynamic deployment/spreading of code (via lambda)
- implicit/explicit device selection of what code execute
- "collective teams" forming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

Outline

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

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!

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.