Programmazione Aggregata: un paradigma innovativo per l’Internet delle Cose

Giorgio Audrito, Ferruccio Damiani*
http://giorgio.audrito.info  http://www.unito.it/damiani
http://di.unito.it/movere

DipInfo si racconta
Workshop del Dipartimento di Informatica
Torino, Italy, 5 February 2019

* Laurea Triennale: Laboratorio di Programmazione II
Laurea Magistrale: Programmazione per Dispositivi Mobili
Programming for the IoT

Aiming for a new “computing system”

- 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
Programming for the IoT poses several non-trivial challenges:

- diverse heterogeneous entities → device abstraction?
- collaboration vs selfishness → centralization? aggregation?
- dynamic goals and environment → adaptive algorithms?
- data security and privacy → cryptography? localised aggregation?

Classical paradigms, algorithms and languages fail to match these expectations.
Programming for the IoT poses several non-trivial challenges:

- diverse heterogeneous entities → device abstraction?
- collaboration vs selfishness → centralization? aggregation?
- dynamic goals and environment → adaptive algorithms?
- data security and privacy → cryptography? localised aggregation?

Classical paradigms, algorithms and languages fail to match these expectations.
Programming for the IoT poses several non-trivial challenges:

- diverse heterogeneous entities $\rightarrow$ device abstraction?
- collaboration vs selfishness $\rightarrow$ centralization? aggregation?
- dynamic goals and environment $\rightarrow$ adaptive algorithms?
- data security and privacy $\rightarrow$ cryptography? localised aggregation?

Classical paradigms, algorithms and languages fail to match these expectations.
Programming for the IoT poses several non-trivial challenges:

- diverse heterogeneous entities → device abstraction?
- collaboration vs selfishness → centralization? aggregation?
- dynamic goals and environment → adaptive algorithms?
- data security and privacy → cryptography? localised aggregation?

Classical paradigms, algorithms and languages fail to match these expectations.
Programming for the IoT poses several non-trivial challenges:

- diverse heterogeneous entities → device abstraction?
- collaboration vs selfishness → centralization? aggregation?
- dynamic goals and environment → adaptive algorithms?
- data security and privacy → cryptography? localised aggregation?

Classical paradigms, algorithms and languages fail to match these expectations.
Aggregate Computing


Practical Toolchain and Programming Languages

Adaptivity, resiliency, robustness, simplicity

Shifting the viewpoint

- From standard single-device focus on system programming
- To aggregate viewpoint:
  - overall set of devices spread in a pervasive computing environment as a single aggregate machine
  - overall dispersed localised data as a single object: a computational field
Aggregate Computing


Practical Toolchain and Programming Languages
Adaptivity, resiliency, robustness, simplicity

Shifting the viewpoint

- From standard single-device focus on system programming
- To aggregate viewpoint:
  - overall set of devices spread in a pervasive computing environment as a single aggregate machine
  - overall dispersed localised data as a single object: a computational field
Aggregate Computing


Practical Toolchain and Programming Languages

Adaptivity, resiliency, robustness, *simplicity*

Shifting the viewpoint

- From standard single-device focus on system programming
- To **aggregate viewpoint:**
  - *overall set of devices spread in a pervasive computing environment as a single aggregate machine*
  - *overall dispersed localised data as a single object: a computational field*
Aggregate Computing


Practical Toolchain and Programming Languages
Adaptivity, resiliency, robustness, *simplicity*

Shifting the viewpoint

- From standard single-device focus on system programming
- To aggregate viewpoint:
  - *overall set of devices spread in a pervasive computing environment as a single aggregate machine*
  - *overall dispersed localised data as a single object: a computational field*
Computational Field

- Map from spatial devices to computational values
  \((\text{booleans, integers, functions...})\)
- Possibly evolving over time
- Can be atomically created and combined with aggregate-level operators
- Functions on them have a natural functional composition
Computational Field

- Map from spatial devices to computational values
  (*booleans, integers, functions...*)
- Possibly evolving over time
- Can be atomically created and combined with aggregate-level operators
- Functions on them have a natural functional composition
Computational Field

- Map from spatial devices to computational values
  
  \(\textit{booleans, integers, functions...}\)

- Possibly evolving over time

- Can be atomically created and combined with aggregate-level operators

- Functions on them have a natural functional composition
Computational Field

- Map from spatial devices to computational values
  (*booleans, integers, functions...*)
- Possibly evolving over time
- Can be atomically created and combined with aggregate-level operators
- Functions on them have a natural **fuctional composition**
Spatial Computing

A natural scenario for Aggregate Computing

- (a) continuous mathematical input transformed by continuous functions
- (b) a discrete network samples the input and the transformation, producing
- (c) an approximation of the modelled behaviour
Spatial Computing

A natural scenario for Aggregate Computing

- (a) continuous mathematical input transformed by continuous functions
- (b) a discrete network samples the input and the transformation, producing
- (c) an approximation of the modelled behaviour
Spatial Computing

A natural scenario for Aggregate Computing

- (a) continuous mathematical input transformed by continuous functions
- (b) a discrete network samples the input and the transformation, producing
- (c) an approximation of the modelled behaviour
def maxDistance(source) {
    collectMax(source, distanceTo(source))
}
maxDistance(temperature() > 20)
Abstraction Layers

Functional composition allows:

- Build **application code** from libraries of **coordination operators**
- Build **coordination operators** from a basic set of constructs in a DSL
- ...which are in turn abstract from **device capabilities**
Abstraction Layers

Functional composition allows:

- Build **application code** from libraries of **coordination operators**
- Build **coordination operators** from a basic set of **constructs** in a DSL
- ...which are in turn abstract from **device capabilities**
Abstraction Layers

Functional composition allows:

- Build **application code** from libraries of **coordination operators**
- Build **coordination operators** from a basic set of **constructs** in a DSL
- ...which are in turn abstract from **device capabilities**
Programming Model

- Programs executed by a network of devices (a dynamic neighbouring relation represents physical or logical proximity)
- Every device is given the same (terminating) program, iterated in asynchronous computation rounds

Computational rounds
- Collect data from:
  - sensors
  - local memory
  - messages received during sleep
- Perform the computation
- Send messages to neighbour devices
- Sleep until the next activation
Programming Model

- Programs executed by a network of devices (a dynamic neighbouring relation represents physical or logical proximity)
- Every device is given the same (terminating) program, iterated in asynchronous computation rounds

Computational rounds
- Collect data from:
  - sensors
  - local memory
  - messages received during sleep
- Perform the computation
- Send messages to neighbour devices
- Sleep until the next activation
Programming Model

- Programs executed by a network of devices (a dynamic neighbouring relation represents physical or logical proximity)
- Every device is given the same (terminating) program, iterated in asynchronous computation rounds

Computational rounds
- Collect data from:
  - sensors
  - local memory
  - messages received during sleep
- Perform the computation
- Send messages to neighbour devices
- Sleep until the next activation
Programming Model

- Programs executed by a network of devices (a dynamic neighbouring relation represents physical or logical proximity)
- Every device is given the same (terminating) program, iterated in asynchronous computation rounds

Computational rounds
- Collect data from:
  - sensors
  - local memory
  - messages received during sleep
- Perform the computation
- Send messages to neighbour devices
- Sleep until the next activation
Programming Model

- Programs executed by a network of devices (a dynamic neighbouring relation represents physical or logical proximity)
- Every device is given the same (terminating) program, iterated in asynchronous computation rounds

Computational rounds

- Collect data from:
  - sensors
  - local memory
  - messages received during sleep
- Perform the computation
- Send messages to neighbour devices
- Sleep until the next activation
Programming Model

- Programs executed by a network of devices (a dynamic neighbouring relation represents physical or logical proximity)
- Every device is given the same (terminating) program, iterated in asynchronous computation rounds

Computational rounds

- Collect data from:
  - sensors
  - local memory
  - messages received during sleep
- Perform the computation
- Send messages to neighbour devices
- Sleep until the next activation
Programming Language: Field Calculus


- High level of abstraction: implicit message exchange and asynchrony
- Automatic translation of behaviour from global to local
- Eased application design by composition of “building blocks”
- Universal language for distributed computations

```java
def distanceTo(source) {
    rep (infinity) { (d) =>
        mux( source, 0, minHood(nbr{d} + nbrRange()) )
    }
}
def collectMax(source, value) {
    let dist = distanceTo(source) in
    rep (-infinity) { (maxv) =>
        maxHoodPlusSelf(mux( nbr{dist} > dist, nbr{maxv}, value ))
    }
}
```
Programming Language: Field Calculus


- High level of abstraction: implicit message exchange and asynchrony
- Automatic translation of behaviour from global to local
- Eased application design by composition of “building blocks”
- Universal language for distributed computations

```python
def distanceTo(source) {
    rep (infinity) { (d) =>
        mux( source, 0, minHood(nbr{d} + nbrRange()) )
    } }
def collectMax(source, value) {
    let dist = distanceTo(source) in
    rep (-infinity) { (maxv) =>
        maxHoodPlusSelf(mux( nbr{dist} > dist, nbr{maxv}, value ))
    } }
```
Programming Language: Field Calculus


- High level of abstraction: implicit message exchange and asynchrony
- Automatic translation of behaviour from global to local
- Eased application design by composition of “building blocks”
- Universal language for distributed computations

```python
def distanceTo(source) {
    rep (infinity) {
        (d) =>
        mux( source, 0, minHood(nbr{d} + nbrRange()) )
    }
}
def collectMax(source, value) {
    let dist = distanceTo(source) in
    rep (-infinity) {
        (maxv) =>
        maxHoodPlusSelf(mux( nbr{dist} > dist, nbr{maxv}, value ))
    }
}
```
Programming Language: Field Calculus

- High level of abstraction: implicit message exchange and asynchrony
- Automatic translation of behaviour from global to local
- Eased application design by composition of “building blocks”
- Universal language for distributed computations

```python
def distanceTo(source) {
    rep (infinity) { (d) =>
        mux( source, 0, minHood(nbr{d} + nbrRange()) )
    }
}
def collectMax(source, value) {
    let dist = distanceTo(source) in
    rep (-infinity) { (maxv) =>
        maxHoodPlusSelf(mux( nbr{dist} > dist, nbr{maxv}, value ))
    }
}
```
Aggregate Computing Toolchain

1. Formal language, semantics and computational models

2. Algorithms of provable behavioural properties
Aggregate Computing Toolchain

1. Formal language, semantics and computational models

2. Algorithms of provable behavioural properties
Library of Building Blocks

Basic
- Temporal evolution (low-pass filter…)
- Distance estimation from metric
- Spreading (broadcast…)
- Collection (map-reduce)


- Voronoi partitioning
- Routing, database sharding...

Derived
- Graph statistics (centrality…)
- Channel establishment
- ...

12 / 20
Example: Distance Estimation

```python
def distanceTo(source) {
    rep (infinity) { (d) =>
        mux ( source, 0, minHood(nbr{d} + nbrRange()) )
    }
}
```

- Distance varies from **red** (zero) to **violet** (high)
Example: Voronoi Partitioning

- Devices partitioned into contiguous regions (represented with different colours)
Example: Channel with Obstacles

- Channels (blue) route between source (orange) and destination (purple) around obstacles (pink)
- Low-density network with topology (green) in evidence (left)
- High-density environment of 10,000 nodes (right)
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- Temporal limit: stabilisation properties \((\textit{independent from scheduling})\)
  

- Spatial limit: approximation properties \((\textit{independent from positioning})\)

Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- **Temporal limit:** stabilisation properties (*independent from scheduling*)

- **Spatial limit:** approximation properties (*independent from positioning*)
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- **Temporal limit:** stabilisation properties \((independent \ from \ scheduling)\)

- **Spatial limit:** approximation properties \((independent \ from \ positioning)\)
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- Temporal limit: stabilisation properties (independent from scheduling)

- Spatial limit: approximation properties (independent from positioning)
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- Temporal limit: stabilisation properties \((independent \ from \ scheduling)\)
  

- Spatial limit: approximation properties \((independent \ from \ positioning)\)
  
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- Temporal limit: **stabilisation** properties (*independent from scheduling*)

- Spatial limit: **approximation** properties (*independent from positioning*)
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- Temporal limit: stabilisation properties \((\text{independent from scheduling})\)

- Spatial limit: approximation properties \((\text{independent from positioning})\)
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- Temporal limit: stabilisation properties \((independent\ from\ scheduling)\)
  

- Spatial limit: approximation properties \((independent\ from\ positioning)\)
  
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- **Temporal limit:** stabilisation properties \((\text{independent from scheduling})\)
  

- **Spatial limit:** approximation properties \((\text{independent from positioning})\)
  
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- Temporal limit: stabilisation properties \( (\text{independent from scheduling}) \)


- Spatial limit: approximation properties \( (\text{independent from positioning}) \)

Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- **Temporal limit: stabilisation properties** *(independent from scheduling)*
  

- **Spatial limit: approximation properties** *(independent from positioning)*
  
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- Temporal limit: stabilisation properties (independent from scheduling)

- Spatial limit: approximation properties (independent from positioning)
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- Temporal limit: stabilisation properties (*independent from scheduling*)

- Spatial limit: approximation properties (*independent from positioning*)
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- Temporal limit: stabilisation properties \((independent \ from \ scheduling)\)
- Spatial limit: approximation properties \((independent \ from \ positioning)\)

Transient behaviour

Bound the speed of temporal convergence towards a limit

- Average case: statistical expected convergence speed
  

- Worst case: real-time guarantees
  
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- Temporal limit: stabilisation properties \((\text{independent from scheduling})\)
- Spatial limit: approximation properties \((\text{independent from positioning})\)

Transient behaviour

Bound the speed of temporal convergence towards a limit

- Average case: statistical expected convergence speed
  

- Worst case: real-time guarantees
  
Properties of Aggregate Programs

Limit behaviour

Ensure that execution is independent from network details

- Temporal limit: stabilisation properties (*independent from scheduling*)
- Spatial limit: approximation properties (*independent from positioning*)

Transient behaviour

Bound the speed of temporal convergence towards a limit

- Average case: statistical expected convergence speed
  

- Worst case: real-time guarantees
  
1. Formal language, semantics and computational models

2. Algorithms of provable behavioural properties

3. Language implementations
   - (2008) Proto built on Scheme
   - (2015) Protelis built on Java
     - https://protelis.github.io
   - (2016) ScaFi built on Scala
     - https://bitbucket.org/scafiteam/scafi
   - upcoming C++ implementation

4. Deployment engine
   - Network simulator (Alchemist)
     - https://alchemistsimulator.github.io
   - Real deployment (BBN Technologies)
Aggregate Computing Toolchain

1. Formal language, semantics and computational models

2. Algorithms of provable behavioural properties

3. Language implementations
   - (2008) Proto built on Scheme
   - (2015) Protelis built on Java
   - (2016) ScaFi built on Scala
   - upcoming C++ implementation

4. Deployment engine
   - Network simulator (Alchemist)
   - Real deployment (BBN Technologies)
Aggregate Computing Toolchain

1. Formal language, semantics and computational models

2. Algorithms of provable behavioural properties

3. Language implementations
   - (2008) Proto built on Scheme
   - (2015) Protelis built on Java
   - (2016) ScaFi built on Scala
     - upcoming C++ implementation

4. Deployment engine
   - Network simulator (Alchemist)
   - Real deployment (BBN Technologies)
Conclusions

- Framework for convenient programming of large distributed networks
- Common algorithms with resilience guarantees
- Practically usable toolchain

Future Work

- Fine-tune the language with optimized constructs
- Expand the library of algorithms
- ...while providing additional composable guarantees
- Improve practical deployments (micro-controllers)
- Develop an execution strategy where computations are carried out partially in the Cloud and dynamically flow up and down
Conclusions

- Framework for convenient programming of large distributed networks
- Common algorithms with resilience guarantees
- Practically usable toolchain

Future Work

- Fine-tune the language with optimized constructs
- Expand the library of algorithms
- ...while providing additional composable guarantees
- Improve practical deployments (micro-controllers)
- Develop an execution strategy where computations are carried out partially in the Cloud and dynamically flow up and down
Thanks!

People currently working on this topic at di.unito.it:

- Ferruccio Damiani (contact) [http://www.di.unito.it/~damiani](http://www.di.unito.it/~damiani)
- Giorgio Audrito [http://www.di.unito.it/~audrito](http://www.di.unito.it/~audrito)
- Enrico Bini [http://www.di.unito.it/~bini](http://www.di.unito.it/~bini)
- Gianluca Torta [http://www.di.unito.it/~torta](http://www.di.unito.it/~torta)

Research group:

- *System Modelling, Verification and Reuse* [http://di.unito.it/movere](http://di.unito.it/movere)