ScaFi: a Scala DSL and Toolkit for Aggregate Programming

Roberto Casadei[a], Mirko Viroli[a], Gianluca Aguzzi[a], Danilo Pianini[a]

[a]*Alma Mater Studiorum—Università di Bologna, Italy*
{*roby.casadei, mirko.viroli, gianluca.aguzzi, danilo.pianini*}*@unibo.it*

Abstract

Supported by current socio-scientific trends, programming the global behaviour of whole computational collectives makes for great opportunities, but also significant challenges. Recently, aggregate computing has emerged as a prominent paradigm for so-called collective adaptive systems programming. To shorten the gap between such research endeavours and mainstream software development and engineering, we present ScaFi, a Scala toolkit providing an internal domain-specific language, libraries, a simulation environment, and runtime support for practical aggregate computing systems development.

Keywords: aggregate programming, computational fields, macro-level programming, distributed computing, Scala toolkit

Nr.	**Code metadata description**	
C1	Current code version	1.1.5
C2	Permanent link to code/repository used for this code version	`https://github.com/scafi/scafi/releases/tag/v1.1.5`
C3	Code Ocean compute capsule	
C4	Legal Code License	Apache 2.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Scala; Scala.js
C7	Compilation requirements, operating environments & dependencies	JDK 1.8+, SBT
C8	Link to developer documentation/-manual	`http://scafi.github.io/docs/`
C9	Support email for questions	roby.casadei@unibo.it

Table 1: Code metadata (mandatory)

Nr.	(Executable) software metadata description	
S1	Current software version	1.1.5
S2	Permanent link to executables of this version	`https://index.scala-lang.org/scafi/scafi/artifacts/`
S3	Legal Software License	Apache 2.0
S4	Computing platforms/Operating Systems	JVM; web
S5	Installation requirements & dependencies	JDK 1.8+
S6	Link to user manual	`http://scafi.github.io/docs/`
S7	Support email for questions	roby.casadei@unibo.it

Table 2: Software metadata (optional)

1. Motivation and significance

Current trends like the Internet of Things and edge computing let us imagine a future of large-scale cyber-physical ecosystems [1]. According to the pervasive computing vision [2], an increasing number of devices capable of computation and communication are expected to be seemingly deployed into the physical world in the near future. This leads to opportunities based on exploiting a large body of computational resources, sensing/actuation capabilities, and data, but also leads to a number of challenges, including coordination, scalability, and maintenance. Multiple research fields try to exploit these opportunities and address the related challenges, including multi-agent systems [3], self-* computing [4], and collective intelligence [5].

Specifically, a fundamental problem is how to practically engineer and even *program* the *collective adaptive* (also called *self-organising*) behaviour of a group of devices or agents [6]. A recent, prominent approach is *aggregate computing* [7, 8]. This approach consists of two main elements:

1. *aggregate execution model* [9] — a "self-organisation-like" distributed execution model based on "continuous" sensing, computation, communication, and actuation, to be performed by all the devices of the system;
2. *field calculus* [10, 8] — a functional language based on a collective data structure abstraction, the *computational field*, supporting the definition of a single *aggregate program* expressing the overall behaviour of the entire aggregate of devices from a global perspective.

By letting every device in the system work according to the aggregate execution model and repeatedly evaluate the aggregate program against its up-to-date local context, it is possible to promote the emergence of robust, collective behaviour [11, 12].

With respect to other approaches for collective adaptive systems programming [6, 13, 14] and more classical approaches for multi-agent (e.g., JaCaMo [15]) and distributed systems programming (e.g. actors [16]), aggregate computing arguably provides benefits to development productivity as a result of the following: (i) *macro-level stance* [17], promoting the ability to address system-level behaviour globally; (ii) *compositionality*, promoting construction of complex behaviour out of simpler behaviours; (iii) *formality*, enabling theoretical investigations and analyses; and (iv) *practicality*, with tools supporting actual *programming* and simulation of resulting collective adaptive systems. A comparison with metrics against traditional approaches can be found in [18].

So, engineering *aggregate systems* involves devising an aggregate program and setting up the *aggregate computing distributed protocol* for its collective execution according to the aggregate execution model. In practice, the aggregate program could be written in any programming framework featuring library-level or programming-level aggregate computing mechanisms (e.g., [19, 20, 21, 22]). Then, the system should be evaluated and tested by simulation before getting deployed on the execution platform of choice. Proper software tooling is essential to support these phases and hence the investigation of new self-organising algorithms and variants or extensions of the programming model, promoting scientific and technological progress.

In the following, we present the ScaFi *(Scala-Fields)* software[1]: an aggregate programming toolkit that comprises an internal DSL (language and virtual machine) as well as supporting components for the simulation and execution of aggregate systems.

2. Software description

ScaFi is a multi-module Scala project hosted on GitHub[2]. It provides a DSL and API modules for writing, testing, and running aggregate programs, namely programs expressed according to the aggregate programming paradigm [7, 8]. Stable versions of ScaFi are delivered through the *Maven Central Repository*. All the artifacts are collected under group

[1] `https://scafi.github.io`

[2] `https://github.com/scafi/scafi`

`it.unibo.scafi`. ScaFi's build process and dependency management leverages the *Simple Build Tool (SBT)*, and a continuous integration/delivery pipeline on *GitHub Actions (GHA)* is in place to ensure that changes do not break existing functionality. ScaFi cross-compiles for Scala 2.11, 2.12, 2.13 and targets both the JVM and the JavaScript platform (through *Scala.js*). Besides functional testing, the quality assurance pipeline includes tools that enforce a consistent programming style (*ScalaStyle*), perform static analysis for early intercepting code smells (*codiga.io*), track and report code coverage (*codecov.io*), and enforce git commit messages consistency (*commitlint*).

2.1. Software Architecture

The high-level architecture of ScaFi is depicted in Figure 1. It consists of the following main components (where each component is an SBT module and deployable artifact):

- `scafi-commons` — provides basic abstractions and utilities (e.g., spatial and temporal abstractions);
- `scafi-core` — provides an aggregate programming DSL (syntax, semantics, and a virtual machine for evaluation of programs), together with a "standard library" of reusable functions;
- `scafi-stdlib-ext` — provides extra library functionality that requires external dependencies and is hence kept separated from the minimalist `scafi-core`;
- `scafi-simulator`: provides basic support for simulating aggregate systems;
- `scafi-simulator-gui` — provides a GUI for visualising and interacting with simulations of aggregate systems;
- `spala` ("spatial Scala"—i.e., a general Aggregate Computing platform[3]) — provides an actor-based aggregate computing middleware (independent of the ScaFi DSL and potentially applicable to other aggregate programming languages as well) based on the Akka toolkit [24];
- `scafi-distributed` — ScaFi integration-layer for `spala`, which can be leveraged to set up actor-based deployments of ScaFi-programmed systems.

[3]Aggregate computing is rooted in spatial computing [23].

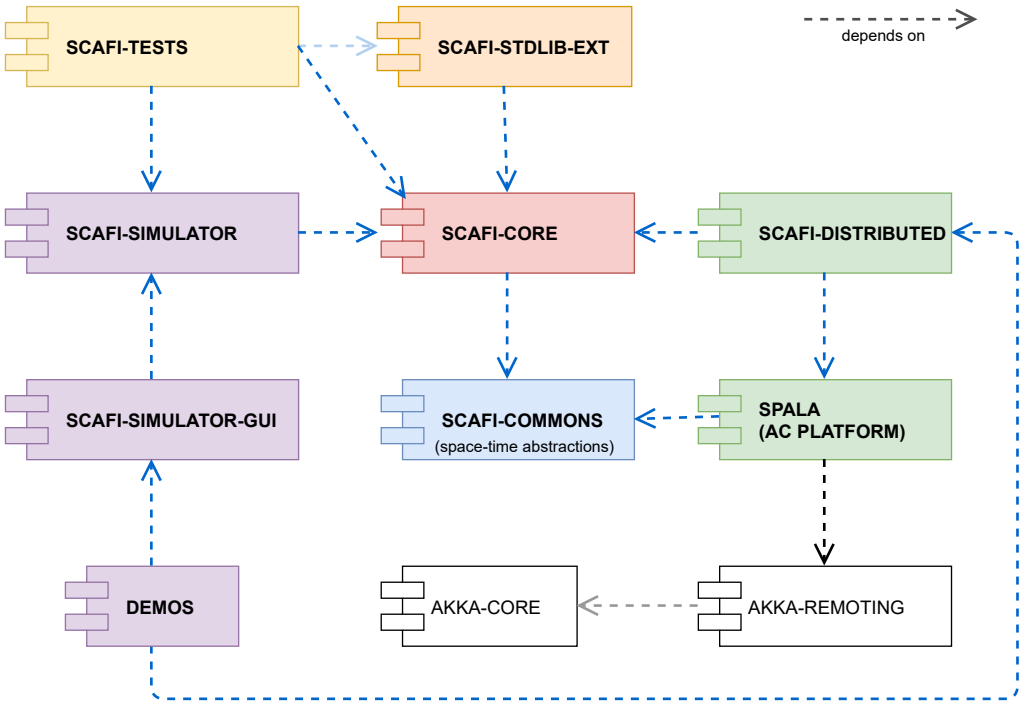


Figure 1: High-level architecture of the ScaFi toolkit.

ScaFi leverages the concept of an *incarnation*, namely a concrete "family of types" [25] that is progressively refined through inheritance, composed, and finally instantiated into an object (cf. the Scala *cake pattern* [26, 25]) which ultimately provides access to a type-coherent set of features.

Figure 2 provides an excerpt of the main Scala traits with some of the types and objects they define. Trait `Core` provides the abstract fundamental types: `CNAME` for capability names, `ID` for device identifiers, `Context` for the input environment of computation rounds, and `Export` for the outcomes of computation rounds. Trait `Language` provides the syntax of the DSL in terms of methods, through interface `Constructs`. Trait `Semantics` and `Engine` implement the DSL construct semantics, providing a template for `AggregateProgram` base class defined in the `Incarnation` trait. The incarnation also exposes `StandardSensors` in terms of, e.g., `SpatialAbstraction`'s and `TimeAbstraction`'s types for positions (`P`), distances (`P`), and time. The `StandardLibrary` is provided by leveraging what an incarnation provides, providing traits of functionality to be mixed into `AggregatePrograms`.

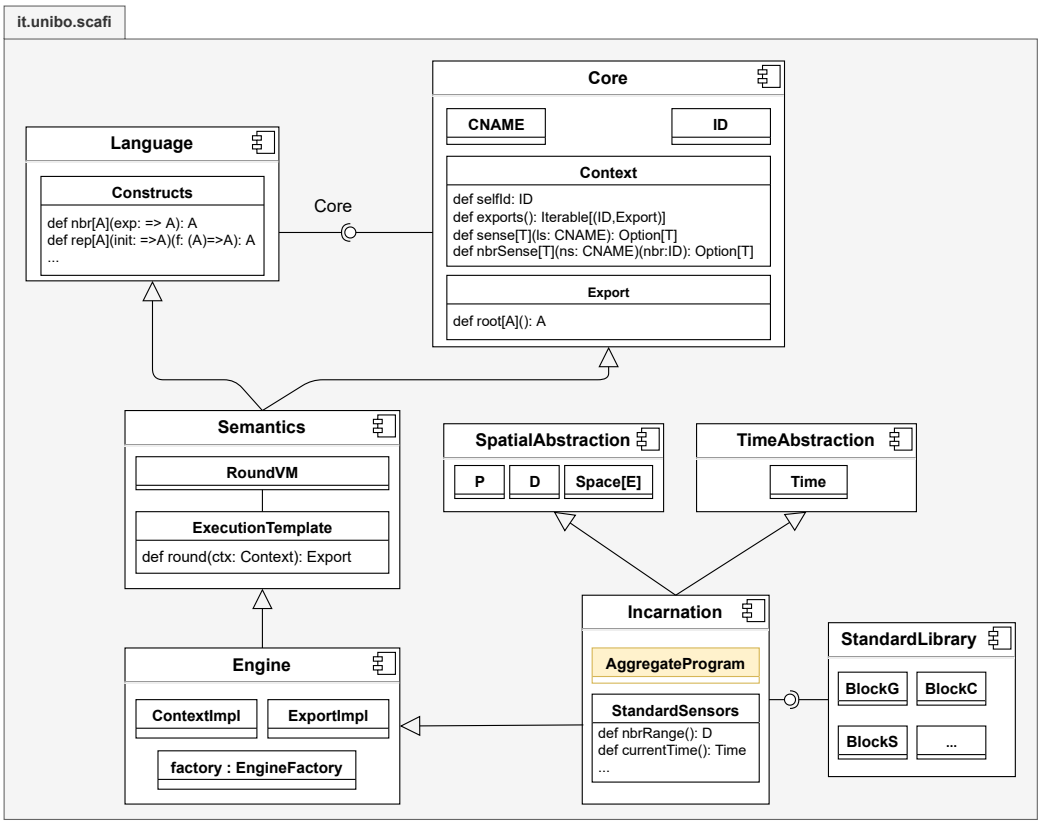


Figure 2: Design of the core of ScaFi (DSL).

2.2. Software Functionalities

Expressing aggregate programs through a Scala DSL

Module `scafi-core` exposes, through incarnations, an `AggregateProgram` trait that provides access to aggregate programming constructs—following a variant of the field calculus [10, 8] formalised in [22, 27]. This single program defines – from a global perspective – the collective adaptive behaviour of an entire ensemble of computational devices. Besides the core constructs, this module also provides "standard library" traits providing access to reusable functions of aggregate functionality. For instance, by mixing trait `Gradients` into an `AggregateProgram` subclass, a developer gets access to *gradient functions* [28, 11], used to continuously compute (over space and time) the self-healing field of minimum distances of each node from a set of source nodes. Several such traits are available to provide other key building blocks for self-organising applications [29, 11] (e.g., `BlockG` for gradient-wise information propagation, `BlockC` for gradient-wise information collection, `BlockS` for sparse choice or leader election) or experimental language features (e.g., the `spawn` function for concurrent aggregate processes [12, 30], for modelling independent and overlapping aggregate computations). Even more functionality is available in module `scafi-stdlib-ext`, which currently provides Shapeless-leveraging [31] typeclasses to extend `Boundedness` constraints (required by some library functions) to arbitrary product types.

Virtual machine for the local execution of aggregate programs

An `AggregateProgram` instance is a function mapping a `Context` (the set of inputs needed by an individual device to properly evaluate the program locally) to an `Export` (the tree of values that has to be shared with neighbours to effectively coordinate and promote emergence of collective behaviours). Using this API, a developer can integrate "aggregate functionality" into its system—what remains to be specified are the details of the aggregate execution model and the communication among devices, that may change in different applications. Devices must continuously run the aggregate program, but the scheduling of these computation rounds can be tuned as the application needs [32]. `Export`s must be shared with neighbouring devices to allow them to properly set up their `Context`s, but the network protocol to be used to do so can be selected independently of the program.

Simulation support

In order to simulate an "aggregate system", it is necessary to (i) define the set of computational devices that make up the aggregate, including their sensors and actuators; (ii) define the aggregate topology, i.e., some application-

specific *neighbouring relationship* from which the set of *neighbours* of each device can be determined; (iii) define the aggregate program to be executed; (iv) define a certain dynamics of the system by proper scheduling of computation rounds, and the environment by proper scheduling of changes in sensor values. Module `scafi-simulator` provides this basic support. It exposes some factory methods to configure simulations properly (e.g., it supports ad-hoc and spatial distance-based connectivity rules) and an API to run and interact with simulations. Then, module `scafi-simulator-gui` provides a convenient graphical user interface to launch and visually show simulations in execution. We remark that these modules currently support basic simulation scenarios and are mainly meant for quick experiments or as a starting basis for ad-hoc simulation frameworks; a further option for sophisticated simulations and data analysis is to use ScaFi within the Alchemist simulator for pervasive computing systems [33, 34].

Experimental or work-in-progress features: 3D simulation frontend and actor-based middleware

ScaFi also includes a front-end for 3D simulations (`renderer-3d`), which are already supported by an execution perspective.

Regarding the construction of actual systems, ScaFi provides an actor-based implementation of the aggregate execution model [35], in the `spala` (`Spa`tial Sca`la`) module, which is instrumental for integrating aggregate computing into existing systems and distributed architectures [35]. Indeed, aggregate computing systems can be designed, deployed, and executed according to different architectural styles and concrete architectures [9]. So, ScaFi provides *two* main implementations of the middleware, in package `it.unibo.scafi.distrib.actor`, for purely peer-to-peer (sub-package `p2p`) and server-based designs (sub-package `server`). The main abstraction is the `DeviceActor`, which exposes a message-based interface for controlling and interacting with an individual logical node of the aggregate system. Then, an object-oriented façade API is provided to set up a system of middleware-level actors.

3. Illustrative Examples

3.1. Hello ScaFi*: building an aggregate system that computes a gradient, from scratch*

This complete example, shown in Figure 3 and available online[4], illustrates how ScaFi can be used to program a (simulated) aggregate system

[4] `https://github.com/scafi/hello-scafi`

for computing a self-stabilising *gradient* field [28, 11] where the output of each device self-stabilises to its minimum distance from an appointed *source* device. Development comes into two parts: (i) definition of the aggregate program, namely the logic of collective behaviour (Figure 3a)[5] [6]; and (ii) definition of an "aggregate execution protocol" determining how devices communicate and act upon their environment (Figure 3b).

3.2. Self-organising Coordination Regions in Simulation

As a more complex example, consider a ScaFi implementation of the Self-Organising Coordination Regions (SCR) pattern [36]. The idea of SCR is to organise a distributed activity into multiple spatial *regions* (inducing a partition of the system), each one controlled by a *leader* device, which collects data from the area members and spreads decisions to enact some area-wide policy. This pattern can be easily implemented in ScaFi using its standard library functions, and simulated through the feature provided by `scafi-simulator`.

For instance, consider the following scenario: temperature monitoring and control in a large environment. For distributed summarisation, we could create areas of uniform sizes and let the devices collectively compute the area's average temperature. Then, we could create an alarm based on collective information, for more coarse-grained analysis and intervention. We implemented this scenario in the repository[7]: Figure 4 shows a simple ScaFi implementation of SCR and a snapshot taken from the ScaFi simulator.

4. Impact

ScaFi has been used in aggregate computing-related research [12, 18, 37, 38, 39, 40, 41, 42, 43, 27], touching themes such as software engineering, computational models, and distributed systems/algorithms. This thread has also several intersections with fields like multi-agent systems, self-organisation, collective intelligence, and scenarios like the Internet of Things, cyber-physical systems, and edge computing. Artifacts published on permanent repositories (like Zenodo) using ScaFi include [44, 45, 46]. Aggregate programming languages have been used in industry [47, 48]. The impact of ScaFi can be understood in terms of existing and prospective contributions, discussed in the following.

[5] For a detailed explanation of this gradient implementation, please refer to e.g. [12].

[6] Concerning source code listings, we highlight symbols as follows: we use blue for Scala keywords, red for ScaFi DSL constructs, purple for ScaFi library functions, and brown for other ScaFi API symbols (e.g., types, objects, constants, and methods).

[7] `https://github.com/scafi/scafi-softwarex-scr-example`

```
// 1. Define/import an incarnation, which provides ScaFi types and classes
object MyIncarnation extends
  it.unibo.scafi.incarnations.BasicAbstractIncarnation
// 2. Bring into scope the stuff from the chosen incarnation
import MyIncarnation._
// 3. Define an "aggregate program" using the ScaFi DSL
// by extending AggregateProgram and specifying a "main" expression
class GradientProgram extends AggregateProgram {
  def isSource: Boolean = sense("source")
  override def main(): Any = rep(Double.PositiveInfinity)(d => {
    mux(isSource){ 0.0 } {
      foldhoodPlus(Double.PositiveInfinity)(Math.min){ nbr(d) + 1.0 }
}})}
```

(a) Program definition

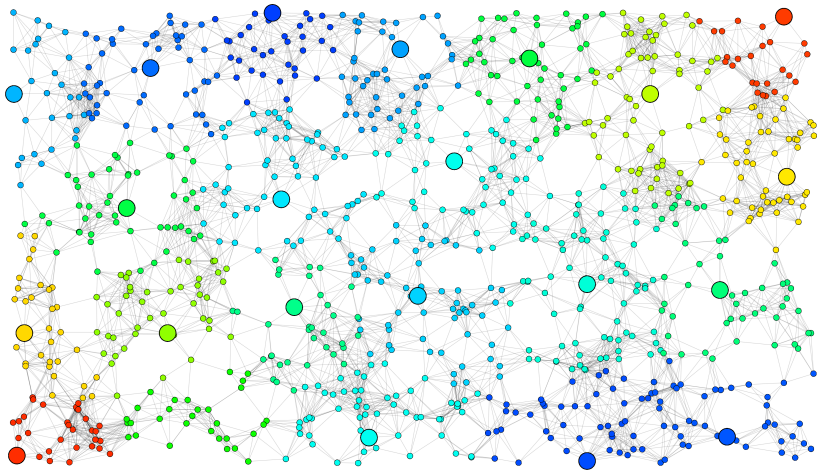
```
// 4. In your program, implement an "execution loop" whereby
// your device or system executes the aggregate program
object HelloScafi extends App {
  case class DeviceState(self: ID, exports: Map[ID, EXPORT],
      localSensors: Map[CNAME, Any], nbrSensors: Map[CNAME, Map[ID, Any]]) {
    def updateExport(dev: ID, export: EXPORT): DeviceState =
        this.copy(exports = exports + (dev -> export))
  }
  val devices = 1 to 5  // (1,2,3,4,5), i.e., 5 devices
  val sourceId = 2      // device 2 is the source of the gradient
  val scheduling = devices ++ devices ++ devices ++ devices ++ devices
  val program = new GradientProgram()
  def neighboursFrom(id: ID): Seq[Int] = // topology: [1]-[2]-[3]-[4]-[5]
    Seq(id - 1, id, id + 1).filter(n => n > 0 && n < 6)
  // Now let's build a simplified system to illustrate the execution model
  var state: Map[ID, DeviceState] = (for {
    d <- devices
  } yield d -> DeviceState(d, Map.empty, Map("source" -> false),
      Map(NBR_RANGE -> (neighboursFrom(d).toSet[ID]
        .map(nbr => nbr -> Math.abs(d - nbr).toDouble)).toMap))).toMap
  state = state + (sourceId -> state(sourceId).copy(localSensors =
      state(sourceId).localSensors + ("source" -> true))) // set source
  // The cycle simulates scheduling&communication by read/write on `state`
  for(d <- scheduling){ // run 5 rounds for each device `d`, round-robin
    // build the local context for device d
    val ctx = factory.context(selfId = d, exports = state(d).exports,
      lsens = state(d).localSensors,nbsens = state(d).nbrSensors)
    println(s"RUN: DEVICE ${d}\n\tCONTEXT: ${state(d)}")
    // run the program against the local context
    val export = program.round(ctx)
    // update d's state
    state += d -> state(d).updateExport(d, export)
    // Simulate sending of messages to neighbours
    state(d).nbrSensors(NBR_RANGE).keySet.foreach(
      nbr => state += nbr -> state(nbr).updateExport(d, export))
    println(s"\tEXPORT: ${export}\n\tOUTPUT: ${export.root()}\n----------")
  }
}
```

(b) System and execution definition

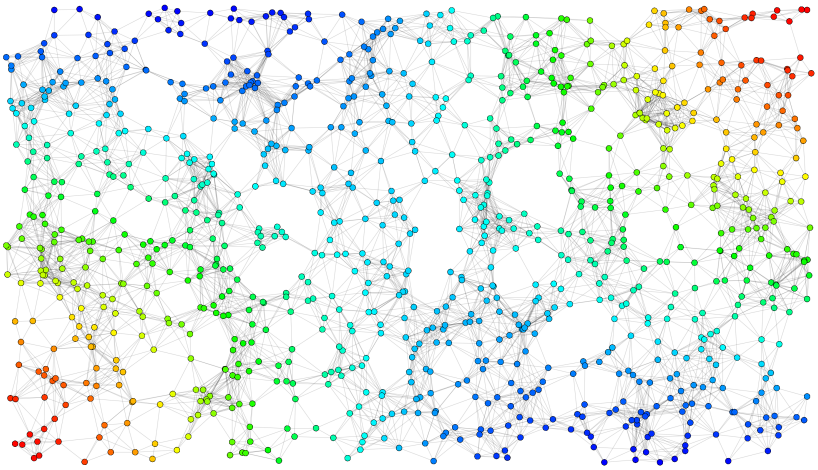
Figure 3: Complete example: an aggregate system computing a gradient.

```
class SCR extends AggregateProgram with BuildingBlocks with StandardSensors {
  val radius = 300 // average area of interest
  val leader = S(radius, nbrRange)
  val potential = distanceTo(leader)
  val averageTemperature = collectMean(potential, temperature)
  val zoneTemperature = broadcast(leader, averageTemperature)
  (leader, zoneTemperature)
  // Coloring following leader information
}
```

(a) ScaFi program



(b) Snapshot from the ScaFi simulator (Big circles mark node as leader).



(c) Simulated temperature distribution.

Figure 4: SCR pattern in ScaFi. Colours denote the temperature perceived by the devices (the redder the higher the temperature).

Interplay between programming language design and foundational research

The implementation of the ScaFi DSL has inspired a variant of the field calculus which arguably supports easier embeddability into mainstream programming languages [22, 27].

Platform for experimenting new aggregate programming language features

ScaFi includes extensions to the basic field calculus. In particular, it supports the *aggregate process* abstraction [12], enabled by the `spawn` construct [49], which provides a way to specify a dynamic number of collective computations running on dynamic ensembles of devices. Another extension is the `exchange` primitive [18, 44], which subsumes previous communication primitives (like `nbr` [10]) and enables differentiated messages for neighbours. In general, as the aggregate programming DSL is exposed as a "plain-old library", it is arguably easier to implement new features, as the developer does not need to deal with parser, compilers, type systems, or language workbenches—of course, at the expense of (syntactic and analytic) constraints exerted by the host language. Moreover, the research orientation of Scala [50] makes it a powerful environment for experimenting new language features and mechanisms.

High-level programming models

The previous discussion makes the case for "DSL stacking" [51]. Indeed, by leveraging the aforementioned aggregate process extension, it is possible to reduce the abstraction gap needed to implement *situated tuples* [42], which is a Linda-like model [52] for coordinating processes where tuples and tuple operations are situated in space. By mapping high-level specifications into aggregate programs, it is sometimes straightforward to develop resilient distributed implementations—as in [53], where translation rules from spatial logic formulas to field calculus expressions enable seamless construction of decentralised monitors for such formulas.

Web-friendliness

By leveraging Scala.js [54], ScaFi can be easily accessed through JavaScript, which promotes cross-platform language design and reuse of functionality in the browser (to support web applications without the need of server-side components). This paved the path to ScaFi-Web [55], a web playground for aggregate programming.

Developer-friendliness

With respect to other programming frameworks for aggregate computing like Proto [19], Protelis [20], and FCPP [21], the ScaFi toolkit provides a

privileged environment for developers. Proto has been discontinued. Its successor, Protelis, is a standalone DSL with duck typing and no support for the definition of new data structures, and whose support for syntax highlighting and code completion is only available for the Eclipse IDE (being based on the Xtext framework [56]). Relatively to FCPP, which is based on C++, ScaFi benefits from the higher level of abstraction provided by Scala and the integration with the Java ecosystem. A more detailed account of this comparison between aggregate programming languages can be found in [8, 27].

Engineering of complex systems and collective intelligence (and related research)

The paradigm embodied by ScaFi provides a means to explore *complex systems* themes [57] (including collective intelligence [5], self-organisation [58], socio-technical collectives [59], emergence [60], etc.), and to do so by an *engineering* and *programming perspective*. For instance, in [12] the ability to self-organise into dynamic groups is exploited to provide forms of intelligent behaviour at the edge; in [36], a self-organisation pattern has been discovered that enables dynamic adjustment of the diameter of feedback-regulated networks and hence of the level of decentralisation in a system, for intelligent use of resources. In [37], reinforcement learning is used to learn policies for determining what actions to execute, in "holes" of ScaFi programs, to improve the dynamics of collective algorithms. We foresee that accessible software toolkits such as ScaFi aimed at programming collective adaptive systems could have an important role in these research threads.

5. Conclusion

This paper presents ScaFi, an open-source Scala-based toolkit for aggregate computing, enabling the development of collective adaptive systems. It provides an internal DSL for the field calculus, a library of reusable aggregate behaviour functions, as well as support components for simulating and executing aggregate systems. Compared to other aggregate programming languages such as Protelis and FCPP, it provides a more high-level platform that might support agile prototyping for research and easier integration with other tools and environments for distributed systems (cf. the Web and Android). We believe it represents a valuable tool for potential scientific and technological developments related to intelligent collective systems.

Conflict of Interest

No conflict of interest exists.

Acknowledgements

This work has been partially supported by the MUR PRIN 2020 Project "COMMON-WEARS" (2020HCWWLP) and the EU/MUR FSE REACT-EU PON R&I 2014-2020.

We also would like to thank Prof. Ferruccio Damiani and Dr. Giorgio Audrito for their contribution to the formal underpinnings of the ScaFi DSL, and the students at the University of Bologna that contributed to the tool.

References

[1] G. D. Abowd, Beyond weiser: From ubiquitous to collective computing, Computer 49 (1) (2016) 17–23. doi:10.1109/MC.2016.22.
URL https://doi.org/10.1109/MC.2016.22

[2] M. Satyanarayanan, Pervasive computing: vision and challenges, IEEE Wirel. Commun. 8 (4) (2001) 10–17. doi:10.1109/98.943998.
URL https://doi.org/10.1109/98.943998

[3] J. Ferber, Multi-agent systems - an introduction to distributed artificial intelligence, Addison-Wesley-Longman, 1999.

[4] J. O. Kephart, D. M. Chess, The vision of autonomic computing, Computer 36 (1) (2003) 41–50. doi:10.1109/MC.2003.1160055.
URL https://doi.org/10.1109/MC.2003.1160055

[5] F. He, Y. Pan, Q. Lin, X. Miao, Z. Chen, Collective intelligence: A taxonomy and survey, IEEE Access 7 (2019) 170213–170225. doi:10.1109/ACCESS.2019.2955677.
URL https://doi.org/10.1109/ACCESS.2019.2955677

[6] R. D. Nicola, S. Jähnichen, M. Wirsing, Rigorous engineering of collective adaptive systems: special section, Int. J. Softw. Tools Technol. Transf. 22 (4) (2020) 389–397. doi:10.1007/s10009-020-00565-0.
URL https://doi.org/10.1007/s10009-020-00565-0

[7] J. Beal, D. Pianini, M. Viroli, Aggregate programming for the internet of things, Computer 48 (9) (2015) 22–30. doi:10.1109/MC.2015.261.
URL https://doi.org/10.1109/MC.2015.261

[8] M. Viroli, J. Beal, F. Damiani, G. Audrito, R. Casadei, D. Pianini, From distributed coordination to field calculus and aggregate computing, J. Log. Algebraic Methods Program. 109. doi:10.1016/j.jlamp.2019.

100486.
URL https://doi.org/10.1016/j.jlamp.2019.100486

[9] R. Casadei, D. Pianini, A. Placuzzi, M. Viroli, D. Weyns, Pulverization in cyber-physical systems: Engineering the self-organizing logic separated from deployment, Future Internet 12 (11) (2020) 203. doi:10.3390/fi12110203.
URL https://doi.org/10.3390/fi12110203

[10] G. Audrito, M. Viroli, F. Damiani, D. Pianini, J. Beal, A higher-order calculus of computational fields, ACM Trans. Comput. Log. 20 (1) (2019) 5:1–5:55. doi:10.1145/3285956.
URL https://doi.org/10.1145/3285956

[11] M. Viroli, G. Audrito, J. Beal, F. Damiani, D. Pianini, Engineering resilient collective adaptive systems by self-stabilisation, ACM Trans. Model. Comput. Simul. 28 (2) (2018) 16:1–16:28. doi:10.1145/3177774.
URL https://doi.org/10.1145/3177774

[12] R. Casadei, M. Viroli, G. Audrito, D. Pianini, F. Damiani, Engineering collective intelligence at the edge with aggregate processes, Eng. Appl. Artif. Intell. 97 (2021) 104081. doi:10.1016/j.engappai.2020.104081.
URL https://doi.org/10.1016/j.engappai.2020.104081

[13] C. Pinciroli, G. Beltrame, Buzz: A programming language for robot swarms, IEEE Softw. 33 (4) (2016) 97–100. doi:10.1109/MS.2016.95.
URL https://doi.org/10.1109/MS.2016.95

[14] Y. A. Alrahman, R. D. Nicola, M. Loreti, Programming interactions in collective adaptive systems by relying on attribute-based communication, Sci. Comput. Program. 192 (2020) 102428. doi:10.1016/j.scico.2020.102428.
URL https://doi.org/10.1016/j.scico.2020.102428

[15] O. Boissier, R. H. Bordini, J. Hubner, A. Ricci, Multi-agent oriented programming: programming multi-agent systems using JaCaMo, MIT Press, 2020.

[16] A. Ricci, P. Haller (Eds.), Programming with Actors - State-of-the-Art and Research Perspectives, Vol. 10789 of Lecture Notes in Computer Science, Springer, 2018. doi:10.1007/978-3-030-00302-9.
URL https://doi.org/10.1007/978-3-030-00302-9

[17] R. Newton, G. Morrisett, M. Welsh, The regiment macroprogramming system, in: T. F. Abdelzaher, L. J. Guibas, M. Welsh (Eds.), Proceedings of the 6th International Conference on Information Processing in Sensor Networks, IPSN 2007, Cambridge, Massachusetts, USA, April 25-27, 2007, ACM, 2007, pp. 489–498. doi:10.1145/1236360.1236422.
URL https://doi.org/10.1145/1236360.1236422

[18] G. Audrito, R. Casadei, F. Damiani, G. Salvaneschi, M. Viroli, Functional programming for distributed systems with XC, in: K. Ali, J. Vitek (Eds.), 36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany, Vol. 222 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 20:1–20:28. doi:10.4230/LIPIcs.ECOOP.2022.20.
URL https://doi.org/10.4230/LIPIcs.ECOOP.2022.20

[19] J. Beal, J. Bachrach, Infrastructure for engineered emergence on sensor/actuator networks, IEEE Intell. Syst. 21 (2) (2006) 10–19. doi:10.1109/MIS.2006.29.
URL https://doi.org/10.1109/MIS.2006.29

[20] D. Pianini, M. Viroli, J. Beal, Protelis: practical aggregate programming, in: R. L. Wainwright, J. M. Corchado, A. Bechini, J. Hong (Eds.), Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015, ACM, 2015, pp. 1846–1853. doi:10.1145/2695664.2695913.
URL https://doi.org/10.1145/2695664.2695913

[21] G. Audrito, FCPP: an efficient and extensible field calculus framework, in: IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2020, Washington, DC, USA, August 17-21, 2020, IEEE, 2020, pp. 153–159. doi:10.1109/ACSOS49614.2020.00037.
URL https://doi.org/10.1109/ACSOS49614.2020.00037

[22] R. Casadei, M. Viroli, G. Audrito, F. Damiani, Fscafi : A core calculus for collective adaptive systems programming, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part II, Vol. 12477 of Lecture Notes in Computer Science, Springer, 2020, pp. 344–360. doi:10.1007/978-3-030-61470-6_21.
URL https://doi.org/10.1007/978-3-030-61470-6_21

[23] J. Beal, S. Dulman, K. Usbeck, M. Viroli, N. Correll, Organizing the aggregate: Languages for spatial computing, CoRR abs/1202.5509. arXiv:1202.5509.
URL http://arxiv.org/abs/1202.5509

[24] R. Roestenburg, R. Bakker, R. Williams, Akka in Action, 1st Edition, Manning Publications Co., USA, 2015.

[25] M. Odersky, M. Zenger, Scalable component abstractions, in: R. E. Johnson, R. P. Gabriel (Eds.), Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2005, October 16-20, 2005, San Diego, CA, USA, ACM, 2005, pp. 41–57. doi:10.1145/1094811.1094815.
URL https://doi.org/10.1145/1094811.1094815

[26] J. Hunt, Cake Pattern, Springer International Publishing, Cham, 2013, pp. 115–119. doi:10.1007/978-3-319-02192-8_13.
URL https://doi.org/10.1007/978-3-319-02192-8_13

[27] G. Audrito, R. Casadei, F. Damiani, M. Viroli, Computation against a neighbour: Addressing large-scale distribution and adaptivity with functional programming and scala (2020). doi:10.48550/ARXIV.2012.08626.
URL https://arxiv.org/abs/2012.08626

[28] J. Beal, J. Bachrach, D. Vickery, M. M. Tobenkin, Fast self-healing gradients, in: R. L. Wainwright, H. Haddad (Eds.), Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008, ACM, 2008, pp. 1969–1975. doi:10.1145/1363686.1364163.
URL https://doi.org/10.1145/1363686.1364163

[29] T. D. Wolf, T. Holvoet, Designing self-organising emergent systems based on information flows and feedback-loops, in: Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2007, Boston, MA, USA, July 9-11, 2007, IEEE Computer Society, 2007, pp. 295–298. doi:10.1109/SASO.2007.16.
URL https://doi.org/10.1109/SASO.2007.16

[30] L. Testa, G. Audrito, F. Damiani, G. Torta, Aggregate processes as distributed adaptive services for the industrial internet of things, Pervasive and Mobile Computing 85 (2022) 101658.

doi:https://doi.org/10.1016/j.pmcj.2022.101658.
URL https://www.sciencedirect.com/science/article/pii/S1574119222000797

[31] D. Gurnell, The Type Astronaut's Guide to Shapeless, Lulu.com, 2017.
URL https://books.google.it/books?id=c9evDgAAQBAJ

[32] D. Pianini, R. Casadei, M. Viroli, S. Mariani, F. Zambonelli, Time-fluid field-based coordination through programmable distributed schedulers, Log. Methods Comput. Sci. 17 (4). doi:10.46298/lmcs-17(4:13)2021.
URL https://doi.org/10.46298/lmcs-17(4:13)2021

[33] D. Pianini, S. Montagna, M. Viroli, Chemical-oriented simulation of computational systems with ALCHEMIST, J. Simulation 7 (3) (2013) 202–215. doi:10.1057/jos.2012.27.
URL https://doi.org/10.1057/jos.2012.27

[34] M. Viroli, R. Casadei, D. Pianini, Simulating large-scale aggregate MASs with Alchemist and Scala, in: M. Ganzha, L. A. Maciaszek, M. Paprzycki (Eds.), Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016, Vol. 8 of Annals of Computer Science and Information Systems, IEEE, 2016, pp. 1495–1504. doi:10.15439/2016F407.
URL https://doi.org/10.15439/2016F407

[35] R. Casadei, M. Viroli, Programming actor-based collective adaptive systems, in: A. Ricci, P. Haller (Eds.), Programming with Actors - State-of-the-Art and Research Perspectives, Vol. 10789 of Lecture Notes in Computer Science, Springer, 2018, pp. 94–122. doi:10.1007/978-3-030-00302-9_4.
URL https://doi.org/10.1007/978-3-030-00302-9_4

[36] D. Pianini, R. Casadei, M. Viroli, A. Natali, Partitioned integration and coordination via the self-organising coordination regions pattern, Future Gener. Comput. Syst. 114 (2021) 44–68. doi:10.1016/j.future.2020.07.032.
URL https://doi.org/10.1016/j.future.2020.07.032

[37] G. Aguzzi, R. Casadei, M. Viroli, Towards reinforcement learning-based aggregate computing, in: M. H. ter Beek, M. Sirjani (Eds.), Coordination Models and Languages - 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings, Vol. 13271

of Lecture Notes in Computer Science, Springer, 2022, pp. 72–91. doi:10.1007/978-3-031-08143-9_5.
URL https://doi.org/10.1007/978-3-031-08143-9_5

[38] R. Casadei, M. Viroli, Coordinating computation at the edge: a decentralized, self-organizing, spatial approach, in: Fourth International Conference on Fog and Mobile Edge Computing, FMEC 2019, Rome, Italy, June 10-13, 2019, IEEE, 2019, pp. 60–67. doi:10.1109/FMEC.2019.8795355.
URL https://doi.org/10.1109/FMEC.2019.8795355

[39] R. Casadei, C. Tsigkanos, M. Viroli, S. Dustdar, Engineering resilient collaborative edge-enabled iot, in: E. Bertino, C. K. Chang, P. Chen, E. Damiani, M. Goul, K. Oyama (Eds.), 2019 IEEE International Conference on Services Computing, SCC 2019, Milan, Italy, July 8-13, 2019, IEEE, 2019, pp. 36–45. doi:10.1109/SCC.2019.00019.
URL https://doi.org/10.1109/SCC.2019.00019

[40] R. Casadei, A. Aldini, M. Viroli, Towards attack-resistant aggregate computing using trust mechanisms, Sci. Comput. Program. 167 (2018) 114–137. doi:10.1016/j.scico.2018.07.006.
URL https://doi.org/10.1016/j.scico.2018.07.006

[41] R. Casadei, G. Aguzzi, M. Viroli, A programming approach to collective autonomy, J. Sens. Actuator Networks 10 (2) (2021) 27. doi:10.3390/jsan10020027.
URL https://doi.org/10.3390/jsan10020027

[42] R. Casadei, M. Viroli, A. Ricci, G. Audrito, Tuple-based coordination in large-scale situated systems, in: F. Damiani, O. Dardha (Eds.), Coordination Models and Languages - 23rd IFIP WG 6.1 International Conference, COORDINATION 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings, Vol. 12717 of Lecture Notes in Computer Science, Springer, 2021, pp. 149–167. doi:10.1007/978-3-030-78142-2_10.
URL https://doi.org/10.1007/978-3-030-78142-2_10

[43] R. Casadei, D. Pianini, M. Viroli, D. Weyns, Digital twins, virtual devices, and augmentations for self-organising cyber-physical collectives, Applied Sciences 12 (1). doi:10.3390/app12010349.
URL https://www.mdpi.com/2076-3417/12/1/349

[44] R. Casadei, scafi/artifact-2021-ecoop-xc: v1.2 (2022). doi:10.5281/ZENODO.6538810.
URL https://zenodo.org/record/6538810

[45] R. Casadei, scafi/artifact-2021-ecoop-smartc: v1.2 (2022). doi:10.5281/ZENODO.6538822.
URL https://zenodo.org/record/6538822

[46] G. Aguzzi, D. Pianini, cric96/experiment-2022-ieee-decentralised-system: 1.0.1 (2022). doi:10.5281/ZENODO.6477039.
URL https://zenodo.org/record/6477039

[47] A. Paulos, S. Dasgupta, J. Beal, Y. Mo, K. D. Hoang, L. J. Bryan, P. P. Pal, R. E. Schantz, J. Schewe, R. K. Sitaraman, A. Wald, C. Wayllace, W. Yeoh, A framework for self-adaptive dispersal of computing services, in: IEEE 4th International Workshops on Foundations and Applications of Self* Systems, FAS*W@SASO/ICCAC 2019, Umea, Sweden, June 16-20, 2019, IEEE, 2019, pp. 98–103. doi:10.1109/FAS-W.2019.00036.
URL https://doi.org/10.1109/FAS-W.2019.00036

[48] J. Beal, K. Usbeck, J. P. Loyall, M. Rowe, J. M. Metzler, Adaptive opportunistic airborne sensor sharing, ACM Trans. Auton. Adapt. Syst. 13 (1) (2018) 6:1–6:29. doi:10.1145/3179994.
URL https://doi.org/10.1145/3179994

[49] R. Casadei, M. Viroli, G. Audrito, D. Pianini, F. Damiani, Aggregate processes in field calculus, in: H. R. Nielson, E. Tuosto (Eds.), Coordination Models and Languages - 21st IFIP WG 6.1 International Conference, COORDINATION 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17-21, 2019, Proceedings, Vol. 11533 of Lecture Notes in Computer Science, Springer, 2019, pp. 200–217. doi:10.1007/978-3-030-22397-7_12.
URL https://doi.org/10.1007/978-3-030-22397-7_12

[50] M. Odersky, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, M. Zenger, et al., An overview of the scala programming language, Tech. rep. (2004).

[51] B. G. Humm, R. S. Engelschall, Language-oriented programming via DSL stacking, in: J. A. M. Cordeiro, M. Virvou, B. Shishkov (Eds.), ICSOFT 2010 - Proceedings of the Fifth International Conference on

Software and Data Technologies, Volume 2, Athens, Greece, July 22-24, 2010, SciTePress, 2010, pp. 279–287.

[52] D. Gelernter, Generative communication in linda, ACM Trans. Program. Lang. Syst. 7 (1) (1985) 80–112. doi:10.1145/2363.2433.
URL https://doi.org/10.1145/2363.2433

[53] G. Audrito, R. Casadei, F. Damiani, V. Stolz, M. Viroli, Adaptive distributed monitors of spatial properties for cyber-physical systems, J. Syst. Softw. 175 (2021) 110908. doi:10.1016/j.jss.2021.110908.
URL https://doi.org/10.1016/j.jss.2021.110908

[54] S. Doeraene, Cross-platform language design in scala.js (keynote), in: S. Erdweg, B. C. d. S. Oliveira (Eds.), Proceedings of the 9th ACM SIGPLAN International Symposium on Scala, SCALA@ICFP 2018, St. Louis, MO, USA, September 28, 2018, ACM, 2018, p. 1. doi:10.1145/3241653.3266230.
URL https://doi.org/10.1145/3241653.3266230

[55] G. Aguzzi, R. Casadei, N. Maltoni, D. Pianini, M. Viroli, Scafi-web: A web-based application for field-based coordination programming, in: F. Damiani, O. Dardha (Eds.), Coordination Models and Languages - 23rd IFIP WG 6.1 International Conference, COORDINATION 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings, Vol. 12717 of Lecture Notes in Computer Science, Springer, 2021, pp. 285–299. doi:10.1007/978-3-030-78142-2_18.
URL https://doi.org/10.1007/978-3-030-78142-2_18

[56] L. Bettini, Implementing Domain-Specific Languages with Xtext and Xtend, Birmingham, ISBN: 9781786464965, Packt Publishing Ltd., UK, 2016.

[57] G. E. Mobus, M. C. Kalton, Principles of Systems Science, Springer Publishing Company, Incorporated, 2014.

[58] F. Yates, Self-Organizing Systems: The Emergence of Order, Life Science Monographs, Springer US, 2012.
URL https://books.google.it/books?id=IiTvBwAAQBAJ

[59] D. Miorandi, V. Maltese, M. Rovatsos, A. Nijholt, J. Stewart, Social Collective Intelligence: Combining the Powers of Humans and Machines

574 to Build a Smarter Society, Springer Publishing Company, Incorporated,
575 2014.

576 [60] S. Kalantari, E. Nazemi, B. Masoumi, Emergence phenomena in self-
577 organizing systems: a systematic literature review of concepts, re-
578 searches, and future prospects, J. Organ. Comput. Electron. Commer.
579 30 (3) (2020) 224–265. doi:10.1080/10919392.2020.1748977.
580 URL https://doi.org/10.1080/10919392.2020.1748977