Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

ScaFi: A Scala DSL and Toolkit for Aggregate Programming

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

Alma Mater Studiorum—Università di Bologna, Italy

A R T I C L E   I N F O

Article history:
Received 23 June 2022
Received in revised form 23 September 2022
Accepted 24 October 2022

Keywords:
Aggregate programming
Computational fields
Macro-level programming
Distributed computing
Scala toolkit

A B S T R A C T

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.

Code metadata

Current code version	1.1.5
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00177
Code Ocean compute capsule	
Legal Code License	Apache 2.0
Code versioning system used	git
Software code languages, tools, and services used	Scala; Scala.js
Compilation requirements, operating environments & dependencies	JDK 1.8+, SBT
Link to developer documentation/manual	http://scafi.github.io/docs/
Support email for questions	roby.casadei@unibo.it

Software metadata

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

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:

[*] Corresponding author.
E-mail addresses: roby.casadei@unibo.it (Roberto Casadei), mirko.viroli@unibo.it (Mirko Viroli), gianluca.aguzzi@unibo.it (Gianluca Aguzzi), danilo.pianini@unibo.it (D. Pianini).

https://doi.org/10.1016/j.softx.2022.101248

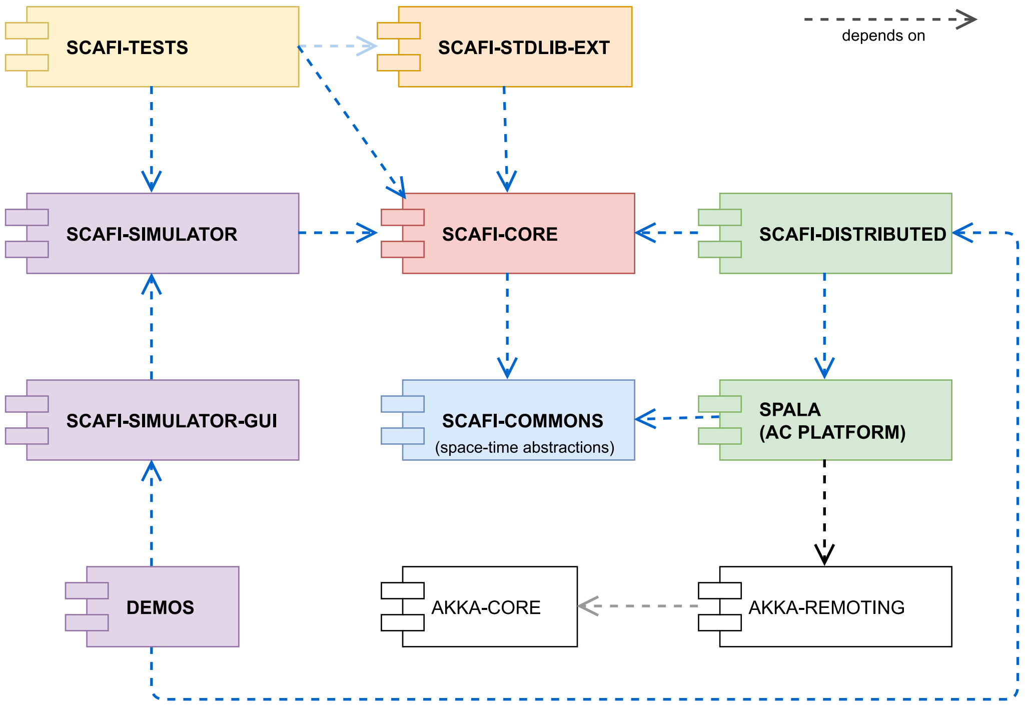


Fig. 1. High-level architecture of the ScaFi toolkit.

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* [8,10] — 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., [19–22]). 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 `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 Fig. 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);

1 https://scafi.github.io
2 https://github.com/scafi/scafi

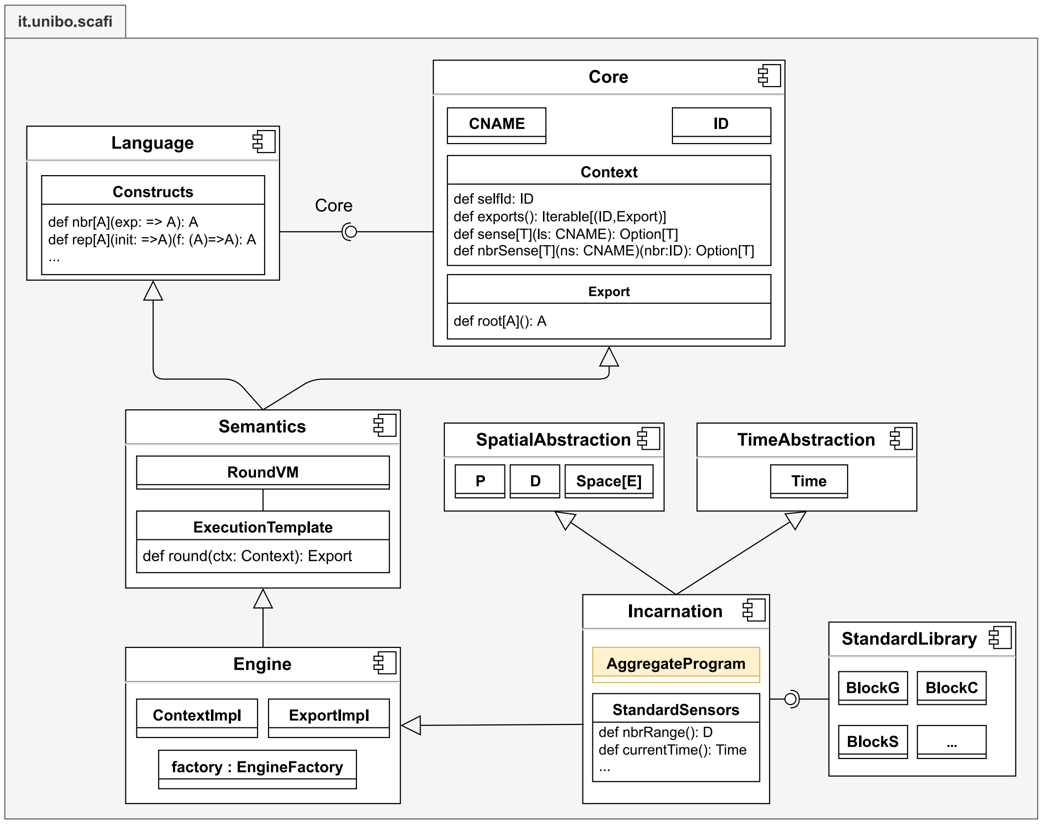


Fig. 2. Design of the core of ScaFi (DSL).

- `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.

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* [25,26]) which ultimately provides access to a type-coherent set of features.

Fig. 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`.

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 [8, 10] 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* [11,28], 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 [11,29] (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

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

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]. Exports must be shared with neighbouring devices to allow them to properly set up their Contexts, 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 (Spatial Scala) 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 Fig. 3 and available online,[4] illustrates how ScaFi can be used to program a (simulated) aggregate system for computing a self-stabilising *gradient* field [11,28] 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 (Fig. 3(a))[5][6]; and (ii) definition of an "aggregate execution protocol" determining how devices communicate and act upon their environment (Fig. 3(b)).

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]: Fig. 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 [12,18,27,37–43], 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 [44–46]. 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.

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].

4 https://github.com/scafi/hello-scafi

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

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

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

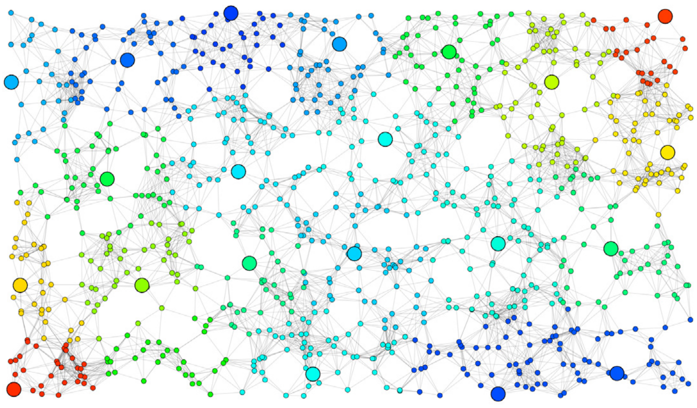
The previous discussion makes the case for "DSL stacking" [51]. Indeed, by leveraging the aforementioned aggregate process extension, it is possible to reduce the abstraction gap needed to implement *situated tuples* [42], which is a Linda-like model [52] for coordinating processes where tuples and tuple 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

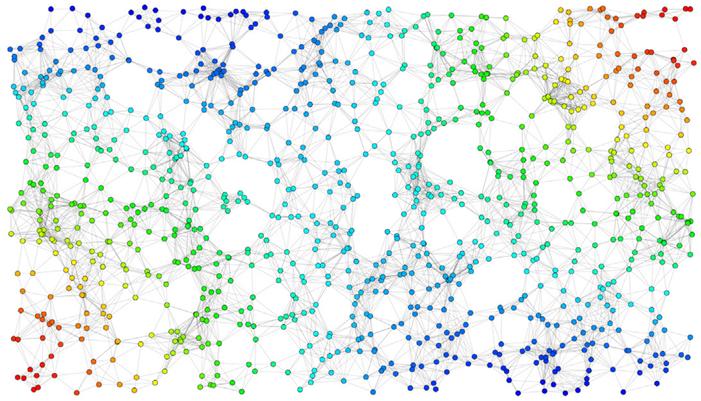
By leveraging Scala.js [54], ScaFi can be easily accessed through JavaScript, which promotes cross-platform language design and reuse of functionality in the browser (to support web

```
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.

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

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.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

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] Abowd GD. Beyond weiser: From ubiquitous to collective computing. Computer 2016;49(1):17–23. http://dx.doi.org/10.1109/MC.2016.22.

[2] Satyanarayanan M. Pervasive computing: Vision and challenges. IEEE Wirel Commun 2001;8(4):10–7. http://dx.doi.org/10.1109/98.943998.

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

[4] Kephart JO, Chess DM. The vision of autonomic computing. Computer 2003;36(1):41–50. http://dx.doi.org/10.1109/MC.2003.1160055.

[5] He F, Pan Y, Lin Q, Miao X, Chen Z. Collective intelligence: A taxonomy and survey. IEEE Access 2019;7:170213–25. http://dx.doi.org/10.1109/ACCESS.2019.2955677.

[6] Nicola RD, Jähnichen S, Wirsing M. Rigorous engineering of collective adaptive systems: Special section. Int J Softw Tools Technol Transf 2020;22(4):389–97. http://dx.doi.org/10.1007/s10009-020-00565-0.

[7] Beal J, Pianini D, Viroli M. Aggregate programming for the Internet of Things. Computer 2015;48(9):22–30. http://dx.doi.org/10.1109/MC.2015.261.

[8] Viroli M, Beal J, Damiani F, Audrito G, Casadei R, Pianini D. From distributed coordination to field calculus and aggregate computing. J Log Algebraic Methods Program 2019;109. http://dx.doi.org/10.1016/j.jlamp.2019.100486.

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

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

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

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

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

[14] Alrahman YA, Nicola RD, Loreti M. Programming interactions in collective adaptive systems by relying on attribute-based communication. Sci Comput Program 2020;192:102428. http://dx.doi.org/10.1016/j.scico.2020.102428.

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

[16] Ricci A, Haller P, editors. Programming with actors - state-of-the-art and research perspectives. In: Lecture notes in computer science, vol. 10789, Springer; 2018, http://dx.doi.org/10.1007/978-3-030-00302-9.

[17] Newton R, Morrisett G, Welsh M. The regiment macroprogramming system. In: Abdelzaher TF, Guibas LJ, Welsh M, editors. Proceedings of the 6th international conference on information processing in sensor networks. ACM; 2007, p. 489–98. http://dx.doi.org/10.1145/1236360.1236422.

[18] Audrito G, Casadei R, Damiani F, Salvaneschi G, Viroli M. Functional programming for distributed systems with XC. In: Ali K, Vitek J, editors. 36th European conference on object-oriented programming. LIPIcs, vol. 222, Schloss Dagstuhl - Leibniz-Zentrum für Informatik; 2022, p. 20:1–28. http://dx.doi.org/10.4230/LIPIcs.ECOOP.2022.20.

[19] Beal J, Bachrach J. Infrastructure for engineered emergence on sensor/actuator networks. IEEE Intell Syst 2006;21(2):10–9. http://dx.doi.org/10.1109/MIS.2006.29.

[20] Pianini D, Viroli M, Beal J. Protelis: practical aggregate programming. In: Wainwright RL, Corchado JM, Bechini A, Hong J, editors. Proceedings of the 30th annual ACM symposium on applied computing. ACM; 2015, p. 1846–53. http://dx.doi.org/10.1145/2695664.2695913.

[21] Audrito G. FCPP: An efficient and extensible field calculus framework. In: IEEE international conference on autonomic computing and self-organizing systems. IEEE; 2020, p. 153–9. http://dx.doi.org/10.1109/ACSOS49614.2020.00037.

[22] Casadei R, Viroli M, Audrito G, Damiani F. FScaFi : A core calculus for collective adaptive systems programming. In: Margaria T, Steffen B, editors. 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. Lecture notes in computer science, vol. 12477, Springer; 2020, p. 344–60. http://dx.doi.org/10.1007/978-3-030-61470-6_21.

[23] Beal J, Dulman S, Usbeck K, Viroli M, Correll N. Organizing the aggregate: Languages for spatial computing. 2012, CoRR abs/1202.5509, arXiv:1202.5509.

[24] Roestenburg R, Bakker R, Williams R. Akka in action. first ed.. USA: Manning Publications Co.; 2015.

[25] Odersky M, Zenger M. Scalable component abstractions. In: Johnson RE, Gabriel RP, editors. Proceedings of the 20th annual ACM SIGPLAN conference on object-oriented programming, systems, languages, and applications. ACM; 2005, p. 41–57. http://dx.doi.org/10.1145/1094811.1094815.

[26] Hunt J. Cake pattern. In: Scala design patterns: patterns for practical reuse and design. Cham: Springer International Publishing; 2013, p. 115–9. http://dx.doi.org/10.1007/978-3-319-02192-8_13.

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

[28] Beal J, Bachrach J, Vickery D, Tobenkin MM. Fast self-healing gradients. In: Wainwright RL, Haddad H, editors. Proceedings of the 2008 ACM symposium on applied computing. ACM; 2008, p. 1969–75. http://dx.doi.org/10.1145/1363686.1364163.

[29] Wolf TD, Holvoet T. 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. IEEE Computer Society; 2007, p. 295–8. http://dx.doi.org/10.1109/SASO.2007.16.

[30] Testa L, Audrito G, Damiani F, Torta G. Aggregate processes as distributed adaptive services for the Industrial Internet of Things. Pervasive Mob Comput 2022;85:101658. http://dx.doi.org/10.1016/j.pmcj.2022.101658, URL https://www.sciencedirect.com/science/article/pii/S1574119222000797.

[31] Gurnell D. The type Astronaut's guide to shapeless. Lulu.com; 2017, URL https://books.google.it/books?id=c9evDgAAQBAJ.

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

[33] Pianini D, Montagna S, Viroli M. Chemical-oriented simulation of computational systems with ALCHEMIST. J Simulation 2013;7(3):202–15. http://dx.doi.org/10.1057/jos.2012.27.

[34] Viroli M, Casadei R, Pianini D. Simulating large-scale aggregate MASs with Alchemist and Scala. In: Ganzha M, Maciaszek LA, Paprzycki M, editors. Proceedings of the 2016 federated conference on computer science and information systems. Annals of computer science and information systems, vol. 8, IEEE; 2016, p. 1495–504. http://dx.doi.org/10.15439/2016F407.

[35] Casadei R, Viroli M. Programming actor-based collective adaptive systems. In: Ricci A, Haller P, editors. Programming with actors - state-of-the-art and research perspectives. Lecture notes in computer science, vol. 10789, Springer; 2018, p. 94–122. http://dx.doi.org/10.1007/978-3-030-00302-9_4.

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

[37] Aguzzi G, Casadei R, Viroli M. Towards reinforcement learning-based aggregate computing. In: ter Beek MH, Sirjani M, editors. 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. Lecture notes in computer science, vol. 13271, Springer; 2022, p. 72–91. http://dx.doi.org/10.1007/978-3-031-08143-9_5.

[38] Casadei R, Viroli M. Coordinating computation at the edge: A decentralized, self-organizing, spatial approach. In: Fourth international conference on fog and mobile edge computing. IEEE; 2019, p. 60–7. http://dx.doi.org/10.1109/FMEC.2019.8795355.

[39] Casadei R, Tsigkanos C, Viroli M, Dustdar S. Engineering resilient collaborative edge-enabled IoT. In: Bertino E, Chang CK, Chen P, Damiani E, Goul M, Oyama K, editors. 2019 IEEE international conference on services computing. IEEE; 2019, p. 36–45. http://dx.doi.org/10.1109/SCC.2019.00019.

[40] Casadei R, Aldini A, Viroli M. Towards attack-resistant aggregate computing using trust mechanisms. Sci Comput Program 2018;167:114–37. http://dx.doi.org/10.1016/j.scico.2018.07.006.

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

[42] Casadei R, Viroli M, Ricci A, Audrito G. Tuple-based coordination in large-scale situated systems. In: Damiani F, Dardha O, editors. 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. Lecture notes in computer science, vol. 12717, Springer; 2021, p. 149–67. http://dx.doi.org/10.1007/978-3-030-78142-2_10.

[43] Casadei R, Pianini D, Viroli M, Weyns D. Digital twins, virtual devices, and augmentations for self-organising cyber-physical collectives. Appl Sci 2022;12(1). http://dx.doi.org/10.3390/app12010349, URL https://www.mdpi.com/2076-3417/12/1/349.

[44] Casadei R. Scafi/artifact-2021-ecoop-xc: v1.2. 2022, http://dx.doi.org/10.5281/ZENODO.6538810, Zenodo, URL https://zenodo.org/record/6538810.

[45] Casadei R. Scafi/artifact-2021-ecoop-smartc: v1.2. 2022, http://dx.doi.org/10.5281/ZENODO.6538822, Zenodo, URL https://zenodo.org/record/6538822.

[46] Aguzzi G, Pianini D. Cric96/experiment-2022-ieee-decentralised-system: 1.0.1. 2022, http://dx.doi.org/10.5281/ZENODO.6477039, Zenodo, URL https://zenodo.org/record/6477039.

[47] Paulos A, Dasgupta S, Beal J, Mo Y, Hoang KD, Bryan LJ, et al. A framework for self-adaptive dispersal of computing services. In: IEEE 4th international workshops on foundations and applications of self* systems. IEEE; 2019, p. 98–103. http://dx.doi.org/10.1109/FAS-W.2019.00036.

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

[49] Casadei R, Viroli M, Audrito G, Pianini D, Damiani F. Aggregate processes in field calculus. In: Nielson HR, Tuosto E, editors. 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. Lecture notes in computer science, vol. 11533, Springer; 2019, p. 200–17. http://dx.doi.org/10.1007/978-3-030-22397-7_12.

[50] Odersky M, Micheloud S, Mihaylov N, Schinz M, Stenman E, Zenger M, et al. An Overview of the Scala Programming Language. Tech. Rep., 2004.

[51] Humm BG, Engelschall RS. Language-oriented programming via DSL stacking. In: Cordeiro JAM, Virvou M, Shishkov B, editors. ICSOFT 2010 - proceedings of the fifth international conference on software and data technologies, vol. 2. SciTePress; 2010, p. 279–87.

[52] Gelernter D. Generative communication in Linda. ACM Trans Program Lang Syst 1985;7(1):80–112. http://dx.doi.org/10.1145/2363.2433.

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

[54] Doeraene S. Cross-platform language design in Scala.js (keynote). In: Erdweg S, d. S. Oliveira BC, editors. Proceedings of the 9th ACM SIGPLAN international symposium on scala. ACM; 2018, p. 1. http://dx.doi.org/10.1145/3241653.3266230.

[55] Aguzzi G, Casadei R, Maltoni N, Pianini D, Viroli M. ScaFi-Web: A web-based application for field-based coordination programming. In: Damiani F, Dardha O, editors. 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. Lecture notes in computer science, vol. 12717, Springer; 2021, p. 285–99. http://dx.doi.org/10.1007/978-3-030-78142-2_18.

[56] Bettini L. Implementing domain-specific languages with Xtext and Xtend. Birmingham: Packt Publishing Ltd., UK; 2016.

[57] Mobus GE, Kalton MC. Principles of systems science. Springer Publishing Company, Incorporated; 2014.

[58] Yates F. Self-organizing systems: the emergence of order. Life science monographs, Springer US; 2012, URL https://books.google.it/books?id=IiTvBwAAQBAJ.

[59] Miorandi D, Maltese V, Rovatsos M, Nijholt A, Stewart J. Social collective intelligence: Combining the powers of humans and machines to build a smarter society. Springer Publishing Company, Incorporated; 2014.

[60] Kalantari S, Nazemi E, Masoumi B. Emergence phenomena in self-organizing systems: A systematic literature review of concepts, researches, and future prospects. J Organ Comput Electron Commer 2020;30(3):224–65. http://dx.doi.org/10.1080/10919392.2020.1748977.