

**SoftwareX**  
**ScaFi: a Scala DSL and Toolkit for Aggregate Programming**  
--Manuscript Draft--

<b>Manuscript Number:</b>	SOFTX-D-22-00177R1
<b>Article Type:</b>	Original software publication
<b>Section/Category:</b>	Domain Independent Tools and Technology
<b>Keywords:</b>	aggregate programming; collective adaptive systems; field-based distributed programming; Scala DSL
<b>Corresponding Author:</b>	Roberto Casadei University of Bologna - Cesena Campus Cesena, ITALY
<b>First Author:</b>	Roberto Casadei
<b>Order of Authors:</b>	Roberto Casadei  Mirko Viroli  Gianluca Aguzzi  Danilo Pianini
<b>Abstract:</b>	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.
<b>Suggested Reviewers:</b>	Michele Loreti michele.loreti@unicam.it He is an expert both on collective adaptive systems and on tools for their programming and quantitative analysis (cf. Sibilla).  Ilias Gerostathopoulos gerostat@in.tum.de He developed DEECO, an ensemble-based component system that also addresses collective adaptive systems development, though in a different way than SCAFI.  Maurice Ter Beek maurice.terbeek@isti.cnr.it He is an expert in the COORDINATION community (was COORDINATION PC chair in 2022 edition) and also recently worked on tools on choreography synthesis.  Andrea Vandin andrea.vandin@santannapisa.it He is an expert on collective adaptive systems and developed MultiVeStA, a statistical model checking tool for discrete event simulators.
<b>Response to Reviewers:</b>	Please see the revision letter PDF

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

## Revision Letter for “SCAFI: a Scala DSL and Toolkit for Aggregate Programming”

Roberto Casadei      Mirko Viroli      Gianluca Aguzzi  
Danilo Pianini

September 23, 2022

Dear Editors,

Thank you very much for giving us the chance to revise our article. We have taken into account all the suggestions from the reviewers and implemented corresponding revisions to our manuscript. Our replies and corrective actions are detailed below. Based on your comments, we have implemented the following main revisions:

- we have clarified the relationship of aggregate computing and ScaFi w.r.t. other approaches (cf. Comment 1.2, 1.5);
- we have improved the introductory explanation of various mentioned concepts (cf. Comment 1.1);
- we have removed less necessary self-citations (cf. Comment 1.1, 1.3, 1.4);
- we have improved formatting of code (cf. Comment 2.1) and provided some details about the implementation of ScaFi (cf. Comment 2.2).

To simplify the review, we attach below this response letter the revised manuscript with corrections/additions highlighted in PURPLE. We are convinced that you will be satisfied with the revised version of our paper and consider this improved version suitable for publication.

Sincerely yours,  
Gianluca Aguzzi, Roberto Casadei, Danilo Pianini, Mirko Viroli

1  
2  
3  
4  
5  
6  
7  
8  
9

## Reviewer 1

10  
11

### Reviewer Comment 1.1

12  
13  
14

In this work, the authors describe ScaFi, a Scala-based toolkit for designing, simulating, and orchestrating (through an Akka-based implementation) aggregate programming applications.

15  
16  
17  
18  
19  
20  
21  
22  
23

In its current form, this work seems more like a catalogue of the authors' previous works than a research paper. The discussion refers to many concepts without introducing them or a generic description, but it is always accompanied by a citation to the authors' previous works. Among others: *incarnation* (row 90), *BlockG*, *BlockC*, *BlockS* (row 103), the *spawn* function for concurrent aggregate processes (row 104), the *exchange* primitive (row 206), and the *situated tuples* coordination model (row 219). This kind of presentation makes some sentences relatively obscure for the reader.

24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34

**Author response:** The paper presents ScaFi software as a language and toolkit for aggregate programming, which is relatively rich paradigm for collective adaptive systems programming. The referred concepts were meant as a representative set of examples of features supported by the language, some of which are peculiar to aggregate computing (e.g., *spawn*, *exchange*—mentioned as not necessarily supported by other aggregate programming languages like Protelis and FCPP), while others recur, maybe in other terms, in self-organization approaches (e.g., *BlockG*, *BlockC*, *BlockS*). Though a full introduction of them goes beyond the scope of the paper, we have improved the introductory explanation of the mentioned concepts:

- 35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49
- *incarnation*: an object (as in the Cake pattern [1, 2]) providing access to a coherent family of types;
  - *BlockG*, *BlockC*, *BlockS*: these denote, respectively, algorithms for information propagation, information collection, and sparse choice (leader election);
  - *spawn* supports the definition of independent and overlapping aggregate computations;
  - *exchange*: a generalisation of communication primitives like *nbr* [3];
  - *situated tuples*: it is a Linda-like model [4] for coordinating processes where tuples and tuple operations are situated in space

50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

so that at least a reader familiar with collective adaptive systems can grasp the idea, and other readers can check out the reference for a comprehensive introduction and more details. Finally, to address the self-referentiality issue and provide other perspectives on those concepts, we have also

- added references to works from other researchers, such as [5–7] (numbered [6, 7, 29] in the manuscript); and

- removed the following five self-references [8–12] (possibly substituting them with references to works from other researchers).

### Reviewer Comment 1.2

Even the concept of aggregate programming is roughly defined in Sec. 1. It is introduced as a way to program the self-organising behaviour of a group of devices or agents. However, the authors do not explain the advantages of using aggregate programming instead of other distributed programming paradigms.

**Author response:** Though a detailed introduction to aggregate programming does not fit the length limitations for this article, we now clarify the benefits of aggregate programming w.r.t. other approaches in Section 1 which, in a nutshell, derive from the combination of four main elements: (i) macro-stance, (ii) compositionality, (iii) formality, (iv) practicality.

### Reviewer Comment 1.3

Besides, the references section seems unbalanced, with more than 65% of references containing at least one of the authors of this work. This concentration of related works on a few people does not denote a broad research community’s interest in the field. This feeling is exacerbated in the “Impact” section, where all but two cited works that used ScaFi have been written by the authors themselves.

**Author response:** We acknowledge that the impact of the approach/toolchain (which is generally a complex social matter) is not as extensive as we believe it deserves. However, on one hand, the paper has exactly the goal of presenting ScaFi to the community of researchers and practitioners, showing that the software is available and usable, and to hopefully promote its usage and contamination across research silos. On the other hand, we point out that several research groups and researchers have contributed to and/or adopted (possibly with the help of some of the authors of this paper) ScaFi at least once, for instance:

- Prof. Ferruccio Damiani and Dr. Giorgio Audrito (University of Turin, IT) [13];
- Prof. Danny Weyns (KU Leuven, BE) [14];
- Prof. Franco Zambonelli and Dr. Stefano Mariani (University of Modena and Reggio Emilia) [15];
- Prof. Guido Salvaneschi (University of St. Gallen, CH) [13];
- Prof. Schahram Dustdar and Dr. Christos Tsigkanos (TU Wien, AT) [16].

Much more researchers could be cited if we consider past adoption of “aggregate computing” in general beyond ScaFi. We hope that ScaFi, thanks to its

improved accessibility (using it is as easy as importing a JVM dependency) and support (cf. the ScaFi-Web playground), could improve the impact of the paradigm.

#### Reviewer Comment 1.4

Plus, even the (rough) comparison with other frameworks for aggregate programming lists 2/3 software developed by one of the authors.

**Author response:** The reference for Proto referred to a first formalisation that resulted out of a collaboration. To accomodate the Reviewer's observation, it has now been updated to [17] (numbered [19] in the manuscript) that more pertinently refers only to the original authors (Beal and Bachrach).

#### Reviewer Comment 1.5

Finally, the "Impact" section lacks a proper evaluation of the ScaFi framework. The authors say that ScaFi implementation was an inspiration for advancing theoretical research in aggregate programming, that it is more developer-friendly than its alternatives, and that it has been used to implement three use-cases, which are briefly and roughly described. However, nothing is said about novel applications enabled by ScaFi or to which extent it provides better performance (in terms of runtime processing or developing time) than other approaches. In particular, no empirical evaluation or quantitative analysis of at least one of the use-cases are provided to demonstrate the actual value of ScaFi, motivating why a researcher should consider ScaFi to implement its workload.

**Author response:** We believe that a quantitative evaluation and comparison are beyond the scope of this article.

A preliminary comparison between aggregate programming and other traditional approaches has been performed in [13]. This provides motivation for aggregate programming languages in general. A brief account on this has also been added as a response to Comment 1.2.

Regarding the relationship between aggregate programming languages, namely between ScaFi, Protelis, Proto, FCPP, much of the discussion revolves around the distinction between internal (ScaFi, FCPP, Proto) and external DSLs (Protelis), and, for internal DSLs, around the capabilities provided by the host language and the corresponding community (Scala for ScaFi, C++ for FCPP, and Scheme for Proto). A discussion about these aspects can be found in [18]; a brief excerpt was included in Section 4 to extend the motivation for ScaFi.

1  
2  
3  
4  
5  
6  
7  
8  
9

## Reviewer 2

10  
11

### Reviewer Comment 2.1

12  
13  
14  
15  
16  
17  
18  
19

The paper describes a Scala toolkit providing for aggregate primitives and systems. The main contribution is to fill the gap between research and industry, whose code contains enough components and facilitate developers to customize their own aggregate programs.

20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
Here are some suggestions:

30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
3  
It is better to rearrange the code and assign more color to distinguish different types in Figure 2. Current codes in Figure 2 seem somehow confused.

40  
41  
42  
43  
44  
45  
46  
47  
**Author response:** The code in Figure 3 has been improved by:

- 48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65
- improving colouring (e.g., strings in orange);
  - rearranging code (so that, e.g., type declarations come at the beginning, then follow all value declarations come at the beginning of their scope, and blank spaces separate logical sections);
  - implementing minor simplifications.

50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
5  
Reviewer Comment 2.2

61  
62  
63  
64  
65

2) The author could provide more details about the aggregate computing and its internal related implementation rather than simply introduce how to use the library if possible.

66  
67  
68  
69  
70  
71  
**Author response:** We have provided more details on aggregate computing and its implementation as follows:

- 72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
5510  
5511  
5512  
5513  
5514  
5515  
5516  
5517  
5518  
5519  
5520  
5521  
5522  
5523  
5524  
5525  
5526  
5527  
5528  
5529  
5530  
5531  
5532  
5533  
5534  
5535  
5536  
5537  
5538  
5539  
5540  
5541  
5542  
5543  
5544  
5545  
5546  
5547  
5548  
5549  
5550  
5551  
5552  
5553  
5554  
5555  
5556  
5557  
5558  
5559  
55510  
55511  
55512  
55513  
55514  
55515  
55516  
55517  
55518  
55519  
55520  
55521  
55522  
55523  
55524  
55525  
55526  
55527  
55528  
55529  
55530  
55531  
55532  
55533  
55534  
55535  
55536  
55537  
55538  
55539  
55540  
55541  
55542  
55543  
55544  
55545  
55546  
55547  
55548  
55549  
55550  
55551  
55552  
55553  
55554  
55555  
55556  
55557  
55558  
55559  
55560  
55561  
55562  
55563  
55564  
55565  
55566  
55567  
55568  
55569  
55570  
55571  
55572  
55573  
55574  
55575  
55576  
55577  
55578  
55579  
55580  
55581  
55582  
55583  
55584  
55585  
55586  
55587  
55588  
55589  
55590  
55591  
55592  
55593  
55594  
55595  
55596  
55597  
55598  
55599  
555100  
555101  
555102  
555103  
555104  
555105  
555106  
555107  
555108  
555109  
555110  
555111  
555112  
555113  
555114  
555115  
555116  
555117  
555118  
555119  
555120  
555121  
555122  
555123  
555124  
555125  
555126  
555127  
555128  
555129  
555130  
555131  
555132  
555133  
555134  
555135  
555136  
555137  
555138  
555139  
555140  
555141  
555142  
555143  
555144  
555145  
555146  
555147  
555148  
555149  
555150  
555151  
555152  
555153  
555154  
555155  
555156  
555157  
555158  
555159  
555160  
555161  
555162  
555163  
555164  
555165  
555166  
555167  
555168  
555169  
555170  
555171  
555172  
555173  
555174  
555175  
555176  
555177  
555178  
555179  
555180  
555181  
555182  
555183  
555184  
555185  
555186  
555187  
555188  
555189  
555190  
555191  
555192  
555193  
555194  
555195  
555196  
555197  
555198  
555199  
555200  
555201  
555202  
555203  
555204  
555205  
555206  
555207  
555208  
555209  
555210  
555211  
555212  
555213  
555214  
555215  
555216  
555217  
555218  
555219  
555220  
555221  
555222  
555223  
555224  
555225  
555226  
555227  
555228  
555229  
555230  
555231  
555232  
555233  
555234  
555235  
555236  
555237  
555238  
555239  
555240  
555241  
555242  
555243  
555244  
555245  
555246  
555247  
555248  
555249  
555250  
555251  
555252  
555253  
555254  
555255  
555256  
555257  
555258  
555259  
555260  
555261  
555262  
555263  
555264  
555265  
555266  
555267  
555268  
555269  
555270  
555271  
555272  
555273  
555274  
555275  
555276  
555277  
555278  
555279  
555280  
555281  
555282  
555283  
555284  
555285  
555286  
555287  
555288  
555289  
555290  
555291  
555292  
555293  
555294  
555295  
555296  
555297  
555298  
555299  
555300  
555301  
555302  
555303  
555304  
555305  
555306  
555307  
555308  
555309  
555310  
555311  
555312  
555313  
555314  
555315  
555316  
555317  
555318  
555319  
555320  
555321  
555322  
555323  
555324  
555325  
555326  
555327  
555328  
555329  
555330  
555331  
555332  
555333  
555334  
555335  
555336  
555337  
555338  
555339  
555340  
555341  
555342  
555343  
555344  
555345  
555346  
555347  
555348  
555349  
555350  
555351  
555352  
555353  
555354  
555355  
555356  
555357  
555358  
555359  
555360  
555361  
555362  
555363  
555364  
555365  
555366  
555367  
555368  
555369  
555370  
555371  
555372  
555373  
555374  
555375  
555376  
555377  
555378  
555379  
555380  
555381  
555382  
555383  
555384  
555385  
555386  
555387  
555388  
555389  
555390  
555391  
555392  
555393  
555394  
555395  
555396  
555397  
555398  
555399  
555400  
555401  
555402  
555403  
555404  
555405  
555406  
555407  
555408  
555409  
555410  
555411  
555412  
555413  
555414  
555415  
555416  
555417  
555418  
555419  
555420  
555421  
555422  
555423  
555424  
555425  
555426  
555427  
555428  
555429  
555430  
555431  
555432  
555433  
555434  
555435  
555436  
555437  
555438  
555439  
555440  
555441  
555442  
555443  
555444  
555445  
555446  
555447  
555448  
555449  
555450  
555451  
555452  
555453  
555454  
555455  
555456  
555457  
555458  
555459  
555460  
555461  
555462  
555463  
555464  
555465  
555466  
555467  
555468  
555469  
555470  
555471  
555472  
555473  
555474  
555475  
555476  
555477  
555478  
555479  
555480  
555481  
555482  
555483  
555484  
555485  
555486  
555487  
555488  
555489  
555490  
555491  
555492  
555493  
555494  
555495  
555496  
555497  
555498  
555499  
555500  
555501  
555502  
555503  
555504  
555505  
555506  
555507  
555508  
555509  
555510  
555511  
555512  
555513  
555514  
555515  
555516  
555517  
555518  
555519  
555520  
555521  
555522  
555523  
555524  
555525  
555526  
555527  
555528  
555529  
555530  
555531  
555532  
555533  
555534  
555535  
555536  
555537  
555538  
555539  
555540  
555541  
555542  
555543  
555544  
555545  
555546  
555547  
555548  
555549  
555550  
555551  
555552  
555553  
555554  
555555  
555556  
555557  
555558  
555559  
555560  
555561  
555562  
555563  
555564  
555565  
555566  
555567  
555568  
555569  
555570  
555571  
555572  
555573  
555574  
555575  
555576  
555577  
555578  
555579  
555580  
555581  
555582  
555583  
555584  
555585  
555586  
555587  
555588  
555589  
555590  
555591  
555592  
555593  
555594  
555595  
555596  
555597  
555598  
555599  
5555100  
5555101  
5555102  
5555103  
5555104  
5555105  
5555106  
5555107  
5555108  
5555109  
5555110  
5555111  
5555112  
5555113  
5555114  
5555115  
5555116  
5555117  
5555118  
5555119  
5555120  
5555121  
5555122  
5555123  
5555124  
5555125  
5555126  
5555127  
5555128  
5555129  
5555130  
5555131  
5555132  
5555133  
5555134  
5555135  
5555136  
5555137  
5555138  
5555139  
5555140  
5555141  
5555142  
5555143  
5555144  
5555145  
5555146  
5555147  
5555148  
5555149  
5555150  
5555151  
5555152  
5555153  
5555154  
5555155  
5555156  
5555157  
5555158  
5555159  
5555160  
5555161  
5555162  
5555163  
5555164  
5555165  
5555166  
5555167  
5555168  
5555169  
5555170  
5555171  
5555172  
5555173  
5555174  
5555175  
5555176  
5555177  
5555178  
5555179  
5555180  
5555181  
5555182  
5555183  
5555184  
5555185  
5555186  
5555187  
5555188  
5555189  
5555190  
5555191  
5555192  
5555193  
5555194  
5555195  
5555196  
5555197  
5555198  
5555199  
5555200  
5555201  
5555202  
5555203  
5555204  
5555205  
5555206  
5555207  
5555208  
5555209  
5555210  
5555211  
5555212  
5555213  
5555214  
5555215  
5555216  
5555217  
5555218  
5555219  
5555220  
5555221  
5555222  
5555223  
5555224  
5555225  
5555226  
5555227  
5555228  
5555229  
5555230  
5555231  
5555232  
5555233  
5555234  
5555235  
5555236  
5555237  
5555238  
5555239  
5555240  
5555241  
5555242  
5555243  
5555244  
5555245  
5555246  
5555247  
5555248  
5555249  
5555250  
5555251  
5555252  
5555253  
5555254  
5555255  
5555256  
5555257  
5555258  
5555259  
5555260  
5555261  
5555262  
5555263  
5555264  
5555265  
5555266  
5555267  
5555268  
5555269  
5555270  
5555271  
5555272  
5555273  
5555274  
5555275  
5555276  
5555277  
5555278  
5555279  
5555280  
5555281  
5555282  
5555283  
5555284  
5555285  
5555286  
5555287  
5555288  
5555289  
5555290  
5555291  
5555292  
5555293  
5555294  
5555295  
5555296  
5555297  
5555298  
5555299  
5555300  
5555301  
5555302  
5555303  
5555304  
5555305  
5555306  
5555307  
5555308  
5555309  
5555310  
5555311  
5555312  
5555313  
5555314  
5555315  
5555316  
5555317  
5555318  
5555319  
5555320  
5555321  
5555322  
5555323  
5555324  
5555325  
5555326  
5555327  
5555328  
5555329  
5555330  
5555331  
5555332  
5555333  
5555334  
5555335  
5555336  
5555337  
5555338  
5555339  
5555340  
5555341  
5555342  
5555343  
5555344  
5555345  
5555346  
5555347  
5555348  
5555349  
5555350  
5555351  
5555352  
5555353  
5555354  
5555355  
5555356  
5555357  
5555358  
5555359  
5555360  
5555361  
5555362  
5555363  
5555364  
5555365  
5555366  
5555367  
5555368  
5555369  
5555370  
5555371  
5555372  
5555373  
5555374  
5555375  
5555376  
5555377  
5555378  
5555379  
5555380  
5555381  
5555382  
5555383  
5555384  
5555385  
5555386  
5555387  
5555388  
5555389  
5555390  
5555391  
5555392  
5555393  
5555394  
5555395  
5555396  
5555397  
5555398  
5555399  
5555400  
5555401  
5555402  
5555403  
5555404  
5555405  
5555406  
5555407  
5555408  
5555409  
5555410  
5555411  
5555412  
5555413  
5555414  
5555415  
5555416  
5555417  
5555418  
5555419  
5555420  
5555421  
5555422  
5555423  
5555424  
5555425  
5555426  
5555427  
5555428  
5555429  
5555430  
5555431  
5555432  
5555433  
5555434  
5555435  
5555436  
5555437  
5555438  
5555439  
5555440  
5555441  
5555442  
5555443

- 1  
2  
3  
4  
5  
6  
7  
8  
9 [2] J. Hunt, *Cake Pattern*, Springer International Publishing, Cham, 2013, pp.  
10 115–119. doi:[10.1007/978-3-319-02192-8\\_13](https://doi.org/10.1007/978-3-319-02192-8_13)  
11 URL [https://doi.org/10.1007/978-3-319-02192-8\\_13](https://doi.org/10.1007/978-3-319-02192-8_13)
- 12 [3] G. Audrito, M. Viroli, F. Damiani, D. Pianini, J. Beal, A higher-order  
13 calculus of computational fields, *ACM Trans. Comput. Log.* 20 (1) (2019)  
14 5:1–5:55. doi:[10.1145/3285956](https://doi.org/10.1145/3285956).  
15 URL <https://doi.org/10.1145/3285956>
- 16 [4] D. Gelernter, Generative communication in linda, *ACM Trans. Program.  
Lang. Syst.* 7 (1) (1985) 80–112. doi:[10.1145/2363.2433](https://doi.org/10.1145/2363.2433).  
17 URL <https://doi.org/10.1145/2363.2433>
- 18 [5] T. D. Wolf, T. Holvoet, Designing self-organising emergent systems based  
19 on information flows and feedback-loops, in: Proceedings of the First Interna-  
20 tional Conference on Self-Adaptive and Self-Organizing Systems, SASO  
21 2007, Boston, MA, USA, July 9–11, 2007, IEEE Computer Society, 2007,  
22 pp. 295–298. doi:[10.1109/SASO.2007.16](https://doi.org/10.1109/SASO.2007.16).  
23 URL <https://doi.org/10.1109/SASO.2007.16>
- 24 [6] J. Beal, J. Bachrach, D. Vickery, M. M. Tobenkin, Fast self-healing gra-  
25 dients, in: R. L. Wainwright, H. Haddad (Eds.), *Proceedings of the 2008  
26 ACM Symposium on Applied Computing (SAC)*, Fortaleza, Ceara, Brazil,  
27 March 16–20, 2008, ACM, 2008, pp. 1969–1975. doi:[10.1145/1363686.1364163](https://doi.org/10.1145/1363686.1364163).  
28 URL <https://doi.org/10.1145/1363686.1364163>
- 29 [7] L. Testa, G. Audrito, F. Damiani, G. Torta, Aggregate pro-  
30 cesses as distributed adaptive services for the industrial inter-  
31 net of things, *Pervasive and Mobile Computing* 85 (2022) 101658.  
32 doi:<https://doi.org/10.1016/j.pmcj.2022.101658>.  
33 URL <https://www.sciencedirect.com/science/article/pii/S1574119222000797>
- 34 [8] G. Audrito, R. Casadei, F. Damiani, M. Viroli, Compositional blocks for  
35 optimal self-healing gradients, in: 11th IEEE International Conference on  
36 Self-Adaptive and Self-Organizing Systems, SASO 2017, Tucson, AZ, USA,  
37 September 18–22, 2017, IEEE Computer Society, 2017, pp. 91–100. doi:  
38 [10.1109/SASO.2017.8186416](https://doi.ieee.org/10.1109/SASO.2017.8186416).  
39 URL <http://doi.ieee.org/10.1109/SASO.2017.8186416>
- 40 [9] R. Casadei, G. Fortino, D. Pianini, W. Russo, C. Savaglio, M. Viroli, Mod-  
41 elling and simulation of opportunistic iot services with aggregate comput-  
42 ing, *Future Gener. Comput. Syst.* 91 (2019) 252–262. doi:[10.1016/j.future.2018.09.005](https://doi.org/10.1016/j.future.2018.09.005).  
43 URL <https://doi.org/10.1016/j.future.2018.09.005>
- 44 [10] C. Saito, A. Igarashi, M. Viroli, Lightweight family polymor-  
45 phism, *J. Funct. Program.* 18 (3) (2008) 285–331. doi:[10.1017/jfp.2008.005](https://doi.org/10.1017/jfp.2008.005).  
46 URL <https://doi.org/10.1017/jfp.2008.005>
- 47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

1  
2  
3  
4  
5  
6  
7  
8  
9 S0956796807006405.  
10 URL <https://doi.org/10.1017/S0956796807006405>

- 11 [11] A. Bucciarone, M. D’Angelo, D. Pianini, G. Cabri, M. De Sanctis,  
12 M. Viroli, R. Casadei, S. Dobson, On the social implications of collec-  
13 tive adaptive systems, *IEEE Technol. Soc. Mag.* 39 (3) (2020) 36–46.  
14 doi:[10.1109/MTS.2020.3012324](https://doi.org/10.1109/MTS.2020.3012324).  
15 URL <https://doi.org/10.1109/MTS.2020.3012324>
- 16 [12] M. Viroli, R. Casadei, D. Pianini, On execution platforms for large-scale  
17 aggregate computing, in: P. Lukowicz, A. Krüger, A. Bulling, Y. Lim,  
18 S. N. Patel (Eds.), *Proceedings of the 2016 ACM International Joint Con-  
19 ference on Pervasive and Ubiquitous Computing, UbiComp Adjunct 2016,*  
20 Heidelberg, Germany, September 12-16, 2016, ACM, 2016, pp. 1321–1326.  
21 doi:[10.1145/2968219.2979129](https://doi.org/10.1145/2968219.2979129).  
22 URL <https://doi.org/10.1145/2968219.2979129>
- 23 [13] G. Audrito, R. Casadei, F. Damiani, G. Salvaneschi, M. Viroli, Functional  
24 programming for distributed systems with XC, in: K. Ali, J. Vitek (Eds.),  
25 36th European Conference on Object-Oriented Programming, ECOOP  
26 2022, June 6-10, 2022, Berlin, Germany, Vol. 222 of LIPICS, Schloss  
27 Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 20:1–20:28. doi:  
28 [10.4230/LIPIcs.ECOOP.2022.20](https://doi.org/10.4230/LIPIcs.ECOOP.2022.20).  
29 URL <https://doi.org/10.4230/LIPIcs.ECOOP.2022.20>
- 30 [14] R. Casadei, D. Pianini, M. Viroli, D. Weyns, Digital twins, virtual devices,  
31 and augmentations for self-organising cyber-physical collectives, *Applied  
32 Sciences* 12 (1) (2022). doi:[10.3390/app12010349](https://doi.org/10.3390/app12010349).  
33 URL <https://www.mdpi.com/2076-3417/12/1/349>
- 34 [15] D. Pianini, R. Casadei, M. Viroli, S. Mariani, F. Zambonelli, Time-fluid  
35 field-based coordination through programmable distributed schedulers,  
36 *Log. Methods Comput. Sci.* 17 (4) (2021). doi:[10.46298/lmcs-17\(4:13\)2021](https://doi.org/10.46298/lmcs-17(4:13)2021).  
37 URL [https://doi.org/10.46298/lmcs-17\(4:13\)2021](https://doi.org/10.46298/lmcs-17(4:13)2021)
- 38 [16] R. Casadei, C. Tsigkanos, M. Viroli, S. Dustdar, Engineering resilient col-  
39 laborative edge-enabled iot, in: E. Bertino, C. K. Chang, P. Chen, E. Damiani,  
40 M. Goul, K. Oyama (Eds.), *2019 IEEE International Conference on  
41 Services Computing, SCC 2019*, Milan, Italy, July 8-13, 2019, IEEE, 2019,  
42 pp. 36–45. doi:[10.1109/SCC.2019.00019](https://doi.org/10.1109/SCC.2019.00019).  
43 URL <https://doi.org/10.1109/SCC.2019.00019>
- 44 [17] J. Beal, J. Bachrach, Infrastructure for engineered emergence on sen-  
45 sor/actuator networks, *IEEE Intell. Syst.* 21 (2) (2006) 10–19. doi:  
46 [10.1109/MIS.2006.29](https://doi.org/10.1109/MIS.2006.29).  
47 URL <https://doi.org/10.1109/MIS.2006.29>

- 1  
2  
3  
4  
5  
6  
7  
8  
9 [18] G. Audrito, R. Casadei, F. Damiani, M. Viroli, Computation against a  
10 neighbour: Addressing large-scale distribution and adaptivity with func-  
11 tional programming and scala (2020). doi:10.48550/ARXIV.2012.08626.  
12 URL <https://arxiv.org/abs/2012.08626>
- 13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

# SCAFI: a Scala DSL and Toolkit for Aggregate Programming

Roberto Casadei<sup>a</sup>, Mirko Viroli<sup>a</sup>, Gianluca Aguzzi<sup>a</sup>, Danilo Pianini<sup>a</sup>

<sup>a</sup>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	<a href="https://github.com/scafi/scafi/releases/tag/v1.1.5">https://github.com/scafi/scafi/releases/tag/v1.1.5</a>
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	<a href="http://scafi.github.io/docs/">http://scafi.github.io/docs/</a>
C9	Support email for questions	roby.casadei@unibo.it

Table 1: Code metadata (mandatory)

Nr.	(Executable) software meta-data description	
S1	Current software version	1.1.5
S2	Permanent link to executables of this version	<a href="https://index.scala-lang.org/scafi/scafi/artifacts/">https://index.scala-lang.org/scafi/scafi/artifacts/</a>
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	<a href="http://scafi.github.io/docs/">http://scafi.github.io/docs/</a>
S7	Support email for questions	roby.casadei@unibo.it

Table 2: Software metadata (optional)

## 1. Motivation and significance

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

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

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

1  
2  
3  
4  
5  
6  
7 By letting every device in the system work according to the aggregate ex-  
8 ecution model and repeatedly evaluate the aggregate program against its  
9 up-to-date local context, it is possible to promote the emergence of robust,  
10 collective behaviour [11, 12].

11  
12 With respect to other approaches for collective adaptive systems pro-  
13 gramming [6, 13, 14] and more classical approaches for multi-agent (e.g.,  
14 JaCaMo [15]) and distributed systems programming (e.g. actors [16]), ag-  
15 gregate computing arguably provides benefits to development productivity  
16 as a result of the following: (i) *macro-level stance* [17], promoting the ability  
17 to address system-level behaviour globally; (ii) *compositionality*, promoting  
18 construction of complex behaviour out of simpler behaviours; (iii) *formality*,  
19 enabling theoretical investigations and analyses; and (iv) *practicality*, with  
20 tools supporting actual *programming* and simulation of resulting collective  
21 adaptive systems. A comparison with metrics against traditional approaches  
22 can be found in [18].

23 So, engineering *aggregate systems* involves devising an aggregate program  
24 and setting up the *aggregate computing distributed protocol* for its collective  
25 execution according to the aggregate execution model. In practice, the ag-  
26 gregate program could be written in any programming framework featuring  
27 library-level or programming-level aggregate computing mechanisms (e.g.,  
28 [19, 20, 21, 22]). Then, the system should be evaluated and tested by sim-  
29 ulation before getting deployed on the execution platform of choice. Proper  
30 software tooling is essential to support these phases and hence the investi-  
31 gation of new self-organising algorithms and variants or extensions of the  
32 programming model, promoting scientific and technological progress.

33 In the following, we present the SCAFI (*Scala-Fields*) software<sup>1</sup>: an ag-  
34 gregate programming toolkit that comprises an internal DSL (language and  
35 virtual machine) as well as supporting components for the simulation and  
36 execution of aggregate systems.

## 45 46 2. Software description

47  
48 SCAFI is a multi-module Scala project hosted on GitHub<sup>2</sup>. It pro-  
49 vides a DSL and API modules for writing, testing, and running aggre-  
50 gate programs, namely programs expressed according to the aggregate pro-  
51 gramming paradigm [7, 8]. Stable versions of SCAFI are delivered through  
52 the *Maven Central Repository*. All the artifacts are collected under group  
53  
54

---

55  
56  
57 <sup>1</sup><https://scafi.github.io>

58 <sup>2</sup><https://github.com/scafi/scafi>

1  
 2  
 3  
 4  
 5  
 6     59 **it.unibo.scafi**. SCAFI’s build process and dependency management lever-  
 7     60 ages the *Simple Build Tool (SBT)*, and a continuous integration/delivery  
 8     61 pipeline on *Github Actions (GHA)* is in place to ensure that changes do not  
 9     62 break existing functionality. SCAFI cross-compiles for Scala 2.11, 2.12, 2.13  
 10     63 and targets both the JVM and the JavaScript platform (through *Scala.js*).  
 11     64 Besides functional testing, the quality assurance pipeline includes tools that  
 12     65 enforce a consistent programming style (*ScalaStyle*), perform static analysis  
 13     66 for early intercepting code smells (*codiga.io*), track and report code coverage  
 14     67 (*codecov.io*), and enforce git commit messages consistency (*commitlint*).  
 15  
 16  
 17  
 18  
 19     68 *2.1. Software Architecture*  
 20  
 21     69 The high-level architecture of SCAFI is depicted in Figure 1. It consists  
 22     70 of the following main components (where each component is an SBT module  
 23     71 and deployable artifact):  
 24  
 25     72     • **scafi-commons** — provides basic abstractions and utilities (e.g., spatial  
 26     73 and temporal abstractions);  
 27  
 28     74     • **scafi-core** — provides an aggregate programming DSL (syntax, se-  
 29     75 mantics, and a virtual machine for evaluation of programs), together  
 30     76 with a “standard library” of reusable functions;  
 31  
 32     77     • **scafi-stdlib-ext** — provides extra library functionality that requires  
 33     78 external dependencies and is hence kept separated from the minimalist  
 34     79 **scafi-core**;  
 35  
 36     80     • **scafi-simulator**: provides basic support for simulating aggregate sys-  
 37     81 tems;  
 38  
 39     82     • **scafi-simulator-gui** — provides a GUI for visualising and interact-  
 40     83 ing with simulations of aggregate systems;  
 41  
 42     84     • **spala** (“spatial Scala”—i.e., a general Aggregate Computing plat-  
 43     85 form<sup>3</sup>) — provides an actor-based aggregate computing middleware  
 44     86 (independent of the SCAFI DSL and potentially applicable to other ag-  
 45     87 gregate programming languages as well) based on the Akka toolkit [24];  
 46  
 47     88     • **scafi-distributed** — ScaFi integration-layer for **spala**, which can  
 48     89 be leveraged to set up actor-based deployments of SCAFI-programmed  
 49     90 systems.  
 50  
 51  
 52  
 53  
 54  
 55  
 56  
 57  
 58

---

<sup>3</sup>Aggregate computing is rooted in spatial computing [23].

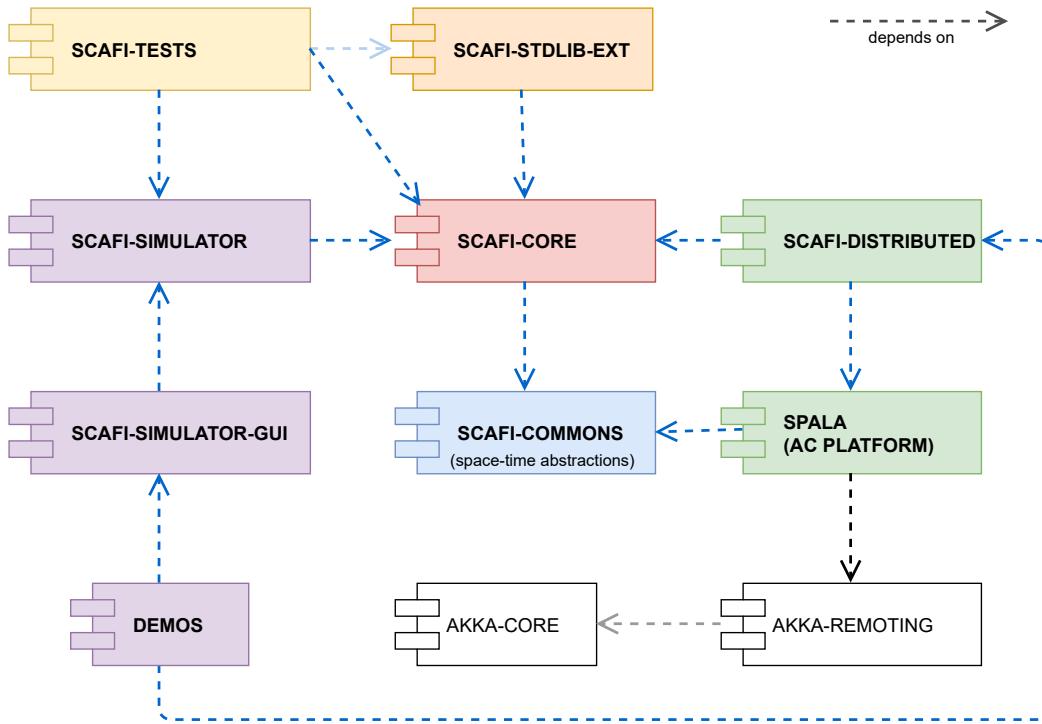


Figure 1: High-level architecture of the SCAFI toolkit.

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`.

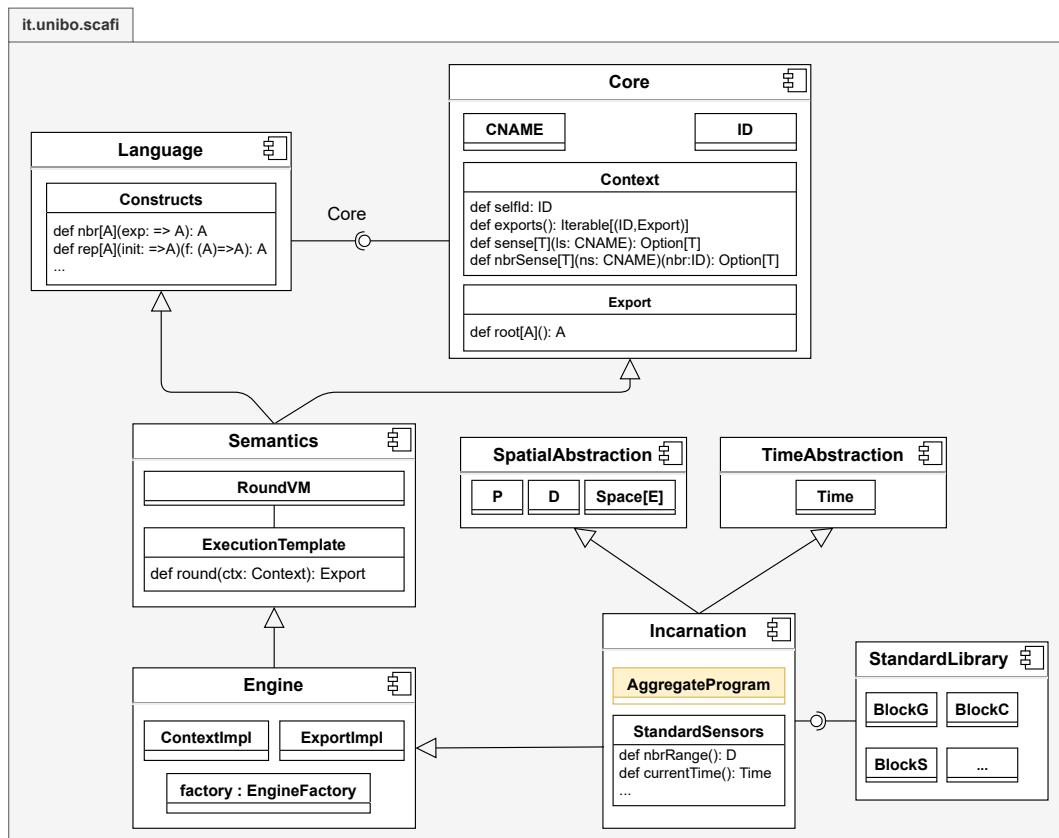


Figure 2: Design of the core of SCAFI (DSL).

1  
2  
3  
4  
5  
6  
7     107 *2.2. Software Functionalities*  
8     108 *Expressing aggregate programs through a Scala DSL*  
9         109 Module `scafi-core` exposes, through incarnations, an  
10      110 `AggregateProgram` trait that provides access to aggregate program-  
11      111 ming constructs—following a variant of the field calculus [10, 8] formalised  
12      112 in [22, 27]. This single program defines – from a global perspective – the  
13      113 collective adaptive behaviour of an entire ensemble of computational devices.  
14      114 Besides the core constructs, this module also provides “standard library”  
15      115 traits providing access to reusable functions of aggregate functionality. For  
16      116 instance, by mixing trait `Gradients` into an `AggregateProgram` subclass,  
17      117 a developer gets access to *gradient functions* [28, 11], used to continuously  
18      118 compute (over space and time) the self-healing field of minimum distances  
19      119 of each node from a set of source nodes. Several such traits are available  
20      120 to provide other key building blocks for self-organising applications [29, 11]  
21      121 (e.g., `BlockG` for gradient-wise information propagation, `BlockC` for gradient-  
22      122 wise information collection, `BlockS` for sparse choice or leader election)  
23      123 or experimental language features (e.g., the `spawn` function for concurrent  
24      124 aggregate processes [12, 30], for modelling independent and overlapping  
25      125 aggregate computations). Even more functionality is available in module  
26      126 `scafi-stdlib-ext`, which currently provides Shapeless-leveraging [31]  
27      127 typeclasses to extend `Boundedness` constraints (required by some library  
28      128 functions) to arbitrary product types.  
29  
30  
31  
32  
33  
34  
35  
36  
37     129 *Virtual machine for the local execution of aggregate programs*  
38         130 An `AggregateProgram` instance is a function mapping a `Context` (the set  
39         131 of inputs needed by an individual device to properly evaluate the program lo-  
40         132 cally) to an `Export` (the tree of values that has to be shared with neighbours  
41         133 to effectively coordinate and promote emergence of collective behaviours).  
42         134 Using this API, a developer can integrate “aggregate functionality” into its  
43         135 system—what remains to be specified are the details of the aggregate ex-  
44         136 ecution model and the communication among devices, that may change in  
45         137 different applications. Devices must continuously run the aggregate program,  
46         138 but the scheduling of these computation rounds can be tuned as the applica-  
47         139 tion needs [32]. `Exports` must be shared with neighbouring devices to allow  
48         140 them to properly set up their `Contexts`, but the network protocol to be used  
49         141 to do so can be selected independently of the program.  
50  
51  
52  
53  
54  
55     142 *Simulation support*  
56         143 In order to simulate an “aggregate system”, it is necessary to (i) define the  
57         144 set of computational devices that make up the aggregate, including their sen-  
58         145 sors and actuators; (ii) define the aggregate topology, i.e., some application-

1  
2  
3  
4  
5  
6  
7     146 specific *neighbouring relationship* from which the set of *neighbours* of each  
8     147 device can be determined; (iii) define the aggregate program to be executed;  
9     148 (iv) define a certain dynamics of the system by proper scheduling of computa-  
10     149 tion rounds, and the environment by proper scheduling of changes in sensor  
11     150 values. Module **scafi-simulator** provides this basic support. It exposes  
12     151 some factory methods to configure simulations properly (e.g., it supports ad-  
13     152 hoc and spatial distance-based connectivity rules) and an API to run and  
14     153 interact with simulations. Then, module **scafi-simulator-gui** provides a  
15     154 convenient graphical user interface to launch and visually show simulations in  
16     155 execution. We remark that these modules currently support basic simulation  
17     156 scenarios and are mainly meant for quick experiments or as a starting basis  
18     157 for ad-hoc simulation frameworks; a further option for sophisticated simula-  
19     158 tions and data analysis is to use SCAFI within the Alchemist simulator for  
20     159 pervasive computing systems [33, 34].  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

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

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

164     Regarding the construction of actual systems, SCAFI provides an actor-  
165     based implementation of the aggregate execution model [35], in the **spala**  
166     (Spatial Scala) module, which is instrumental for integrating aggregate com-  
167     puting into existing systems and distributed architectures [35]. Indeed, ag-  
168     gregate computing systems can be designed, deployed, and executed ac-  
169     cording to different architectural styles and concrete architectures [9]. So,  
170     SCAFI provides *two* main implementations of the middleware, in package  
171     **it.unibo.scafi.distrib.actor**, for purely peer-to-peer (sub-package **p2p**)  
172     and server-based designs (sub-package **server**). The main abstraction is the  
173     **DeviceActor**, which exposes a message-based interface for controlling and  
174     interacting with an individual logical node of the aggregate system. Then, an  
175     object-oriented façade API is provided to set up a system of middleware-level  
176     actors.

### 177     3. Illustrative Examples

178     3.1. *Hello SCAFI: building an aggregate system that computes a gradient,*  
179     *from scratch*

180     This complete example, shown in Figure 3 and available online<sup>4</sup>, illus-  
181     trates how SCAFI can be used to program a (simulated) aggregate system

---

<sup>4</sup><https://github.com/scafi/hello-scafi>

1  
2  
3  
4  
5  
6  
7 182 for computing a self-stabilising *gradient* field [28, 11] where the output of  
8 each device self-stabilises to its minimum distance from an appointed *source*  
9 device. Development comes into two parts: (i) definition of the aggregate  
10 program, namely the logic of collective behaviour (Figure 3a)<sup>5</sup><sup>6</sup>; and (ii) def-  
11 inition of an “aggregate execution protocol” determining how devices com-  
12 municate and act upon their environment (Figure 3b).  
13  
14

15 188 *3.2. Self-organising Coordination Regions in Simulation*  
16  
17 189 As a more complex example, consider a SCAFI implementation of the  
18 Self-Organising Coordination Regions (SCR) pattern [36]. The idea of SCR  
19 is to organise a distributed activity into multiple spatial *regions* (inducing  
20 a partition of the system), each one controlled by a *leader* device, which  
21 collects data from the area members and spreads decisions to enact some  
22 area-wide policy. This pattern can be easily implemented in SCAFI using its  
23 standard library functions, and simulated through the feature provided by  
24 **scafi-simulator**.  
25  
26 197 For instance, consider the following scenario: temperature monitoring  
27 and control in a large environment. For distributed summarisation, we could  
28 create areas of uniform sizes and let the devices collectively compute the  
29 area’s average temperature. Then, we could create an alarm based on col-  
30 lective information, for more coarse-grained analysis and intervention. We  
31 implemented this scenario in the repository<sup>7</sup>: Figure 4 shows a simple SCAFI  
32 implementation of SCR and a snapshot taken from the SCAFI simulator.  
33  
34  
35  
36

37 204 **4. Impact**  
38  
39

40 205 SCAFI has been used in aggregate computing-related research [12, 18,  
41 37, 38, 39, 40, 41, 42, 43, 27], touching themes such as software engi-  
42 neering, computational models, and distributed systems/algorithms. This  
43 thread has also several intersections with fields like multi-agent systems, self-  
44 organisation, collective intelligence, and scenarios like the Internet of Things,  
45 cyber-physical systems, and edge computing. Artifacts published on perma-  
46 nent repositories (like Zenodo) using SCAFI include [44, 45, 46]. Aggregate  
47 programming languages have been used in industry [47, 48]. The impact of  
48 SCAFI can be understood in terms of existing and prospective contributions,  
49 discussed in the following.  
50  
51  
52

53  
54 55 <sup>5</sup>For a detailed explanation of this gradient implementation, please refer to e.g. [12].  
56  
57 56 <sup>6</sup>Concerning source code listings, we highlight symbols as follows: we use blue for Scala  
58 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).  
59  
60

61 7<https://github.com/scafi/scafi-softwarex-scr-example>  
62  
63  
64  
65

```

1
2
3
4
5
6
7
8 // 1. Define/import an incarnation, which provides ScaFi types and classes
9 object MyIncarnation extends
10    it.unibo.scafi.incarnations.BasicAbstractIncarnation
11 // 2. Bring into scope the stuff from the chosen incarnation
12 import MyIncarnation._
13 // 3. Define an "aggregate program" using the ScaFi DSL
14 // by extending AggregateProgram and specifying a "main" expression
15 class GradientProgram extends AggregateProgram {
16   def isSource: Boolean = sense("source")
17   override def main(): Any = rep(Double.PositiveInfinity)(d => {
18     mux(isSource){ 0.0 } {
19       foldhoodPlus(Double.PositiveInfinity)(Math.min){ nbr(d) + 1.0 }
20     }})
21 }
```

(a) Program definition

```

22 // 4. In your program, implement an "execution loop" whereby
23 // your device or system executes the aggregate program
24 object HelloScafi extends App {
25   case class DeviceState(self: ID, exports: Map[ID, EXPORT],
26     localSensors: Map[CNAME, Any], nbrSensors: Map[CNAME, Map[ID, Any]]) {
27     def updateExport(dev: ID, export: EXPORT): DeviceState =
28       this.copy(exports = exports + (dev -> export))
29   }
30   val devices = 1 to 5 // (1,2,3,4,5), i.e., 5 devices
31   val sourceId = 2 // device 2 is the source of the gradient
32   val scheduling = devices ++ devices ++ devices ++ devices
33   val program = new GradientProgram()
34   def neighboursFrom(id: ID): Seq[Int] = // topology: [1]-[2]-[3]-[4]-[5]
35     Seq(id - 1, id, id + 1).filter(n => n > 0 && n < 6)
36   // Now let's build a simplified system to illustrate the execution model
37   var state: Map[ID, DeviceState] = (for {
38     d <- devices
39   } yield d -> DeviceState(d, Map.empty, Map("source" -> false),
40     Map(NBR_RANGE -> (neighboursFrom(d).toSet[ID]
41       .map(nbr -> nbr -> Math.abs(d - nbr).toDouble)).toMap))).toMap
42   state = state + (sourceId -> state(sourceId).copy(localSensors =
43     state(sourceId).localSensors + ("source" -> true))) // set source
44   // The cycle simulates scheduling&communication by read/write on 'state'
45   for(d <- scheduling){ // run 5 rounds for each device 'd', round-robin
46     // build the local context for device d
47     val ctx = factory.context(selfId = d, exports = state(d).exports,
48       lsens = state(d).localSensors, nbsens = state(d).nbrSensors)
49     println(s"RUN: ${d}\nCONTEXT: ${state(d)}")
50     // run the program against the local context
51     val export = program.round(ctx)
52     // update d's state
53     state += d -> state(d).updateExport(d, export)
54     // Simulate sending of messages to neighbours
55     state(d).nbrSensors(NBR_RANGE).keySet.foreach(
56       nbr -> state += nbr -> state(nbr).updateExport(d, export))
57     println(s"\tEXPORT: ${export}\n\tOUTPUT: ${export.root()}\n-----")
58   }
59 }
```

(b) System and execution definition

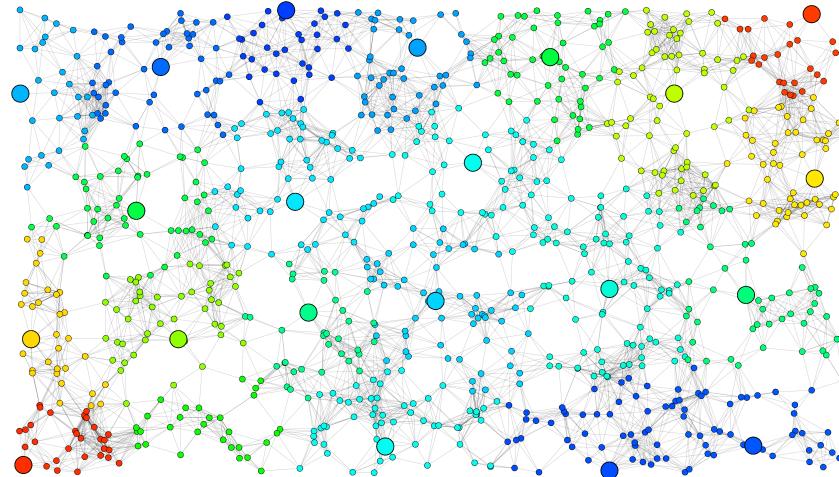
Figure 3: Complete example: an aggregate system computing a gradient.

```

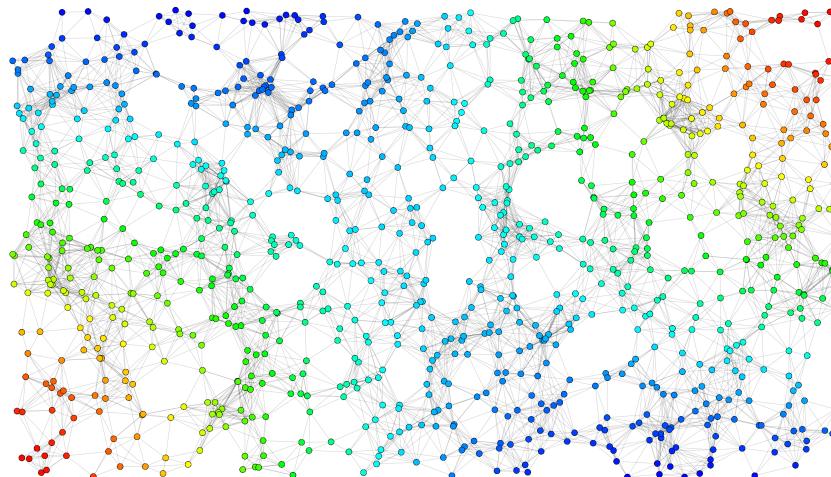
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
    class SCR extends AggregateProgram with BuildingBlocks with StandardSensors {
      val radius = 300 // average 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).

1  
2  
3  
4  
5  
6 215 *Interplay between programming language design and foundational research*  
7 216 The implementation of the SCAFI DSL has inspired a variant of the  
8 217 field calculus which arguably supports easier embeddability into mainstream  
9 218 programming languages [22, 27].  
10  
11

12  
13 219 *Platform for experimenting new aggregate programming language features*  
14 220 SCAFI includes extensions to the basic field calculus. In particular, it  
15 221 supports the *aggregate process* abstraction [12], enabled by the `spawn` con-  
16 222 struct [49], which provides a way to specify a dynamic number of collective  
17 223 computations running on dynamic ensembles of devices. Another exten-  
18 224 sion is the `exchange` primitive [18, 44], which subsumes previous commu-  
19 225 nication primitives (`like` `nbr` [10]) and enables differentiated messages for  
20 226 neighbours. In general, as the aggregate programming DSL is exposed as a  
21 227 “plain-old library”, it is arguably easier to implement new features, as the  
22 228 developer does not need to deal with parser, compilers, type systems, or  
23 229 language workbenches—of course, at the expense of (syntactic and analytic)  
24 230 constraints exerted by the host language. Moreover, the research orienta-  
25 231 tion of Scala [50] makes it a powerful environment for experimenting new  
26 232 language features and mechanisms.  
27  
28  
29  
30  
31

32 233 *High-level programming models*  
33  
34 234 The previous discussion makes the case for “DSL stacking” [51]. Indeed,  
35 235 by leveraging the aforementioned aggregate process extension, it is possi-  
36 236 ble to reduce the abstraction gap needed to implement *situated tuples* [42],  
37 237 which is a Linda-like model [52] for coordinating processes where tuples and  
38 238 tuple operations are situated in space. By mapping high-level specifications  
39 239 into aggregate programs, it is sometimes straightforward to develop resilient  
40 240 distributed implementations—as in [53], where translation rules from spatial  
41 241 logic formulas to field calculus expressions enable seamless construction of  
42 242 decentralised monitors for such formulas.  
43  
44  
45  
46  
47

48 243 *Web-friendliness*  
49  
50 244 By leveraging Scala.js [54], SCAFI can be easily accessed through  
51 245 JavaScript, which promotes cross-platform language design and reuse of func-  
52 246 tionality in the browser (to support web applications without the need of  
53 247 server-side components). This paved the path to SCAFI-WEB [55], a web  
54 248 playground for aggregate programming.  
55  
56

57 249 *Developer-friendliness*  
58  
59 250 With respect to other programming frameworks for aggregate computing  
60 251 like Proto [19], Protelis [20], and FCPP [21], the SCAFI toolkit provides a  
61  
62  
63  
64  
65

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
252 privileged environment for developers. Proto has been discontinued. Its suc-  
253 cessor, Protelis, is a standalone DSL with duck typing and no support for the  
254 definition of new data structures, and whose support for syntax highlighting  
255 and code completion is only available for the Eclipse IDE (being based on the  
256 Xtext framework [56]). Relatively to FCPP, which is based on C++, SCAFI  
257 benefits from the higher level of abstraction provided by Scala and the inte-  
258 gration with the Java ecosystem. A more detailed account of this comparison  
259 between aggregate programming languages can be found in [8, 27].

260 *Engineering of complex systems and collective intelligence (and related re-*  
261 *search)*

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

## 275 5. Conclusion

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

## 286 Conflict of Interest

287 No conflict of interest exists.

1  
2  
3  
4  
5  
6 288 Acknowledgements  
7

8 289 This work has been partially supported by the MUR PRIN 2020 Project  
9 290 “COMMON-WEARS” (2020HCWWLP) and the EU/MUR FSE REACT-  
10 291 EU PON R&I 2014-2020.

11 292 We also would like to thank Prof. Ferruccio Damiani and Dr. Giorgio  
12 293 Audrito for their contribution to the formal underpinnings of the SCAFI DSL,  
13 294 and the students at the University of Bologna that contributed to the tool.  
14  
15  
16  
17

18 295 References  
19

- 20 [1] G. D. Abowd, Beyond weiser: From ubiquitous to collective computing,  
21 Computer 49 (1) (2016) 17–23. doi:10.1109/MC.2016.22.  
22 URL <https://doi.org/10.1109/MC.2016.22>
- 23 [2] M. Satyanarayanan, Pervasive computing: vision and challenges, IEEE  
24 Wirel. Commun. 8 (4) (2001) 10–17. doi:10.1109/98.943998.  
25 URL <https://doi.org/10.1109/98.943998>
- 26 [3] J. Ferber, Multi-agent systems - an introduction to distributed artificial  
27 intelligence, Addison-Wesley-Longman, 1999.
- 28 [4] J. O. Kephart, D. M. Chess, The vision of autonomic computing, Computer  
29 36 (1) (2003) 41–50. doi:10.1109/MC.2003.1160055.  
30 URL <https://doi.org/10.1109/MC.2003.1160055>
- 31 [5] F. He, Y. Pan, Q. Lin, X. Miao, Z. Chen, Collective intelligence: A  
32 taxonomy and survey, IEEE Access 7 (2019) 170213–170225. doi:10.  
33 307 1109/ACCESS.2019.2955677.  
34 URL <https://doi.org/10.1109/ACCESS.2019.2955677>
- 35 [6] R. D. Nicola, S. Jähnichen, M. Wirsing, Rigorous engineering of col-  
36 lective adaptive systems: special section, Int. J. Softw. Tools Technol.  
37 Transf. 22 (4) (2020) 389–397. doi:10.1007/s10009-020-00565-0.  
38 URL <https://doi.org/10.1007/s10009-020-00565-0>
- 39 [7] J. Beal, D. Pianini, M. Viroli, Aggregate programming for the internet  
40 of things, Computer 48 (9) (2015) 22–30. doi:10.1109/MC.2015.261.  
41 URL <https://doi.org/10.1109/MC.2015.261>
- 42 [8] M. Viroli, J. Beal, F. Damiani, G. Audrito, R. Casadei, D. Pianini, From  
43 distributed coordination to field calculus and aggregate computing, J.  
44 Log. Algebraic Methods Program. 109. doi:10.1016/j.jlamp.2019.  
45

- 1  
2  
3  
4  
5  
6  
7       321      100486.  
8       322      URL <https://doi.org/10.1016/j.jlamp.2019.100486>  
9  
10      323 [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:  
11           324      10.3390/fi12110203.  
12           325  
13           326  
14           327      URL <https://doi.org/10.3390/fi12110203>  
15  
16  
17      328 [10] G. Audrito, M. Viroli, F. Damiani, D. Pianini, J. Beal, A higher-order  
18           329      calculus of computational fields, ACM Trans. Comput. Log. 20 (1)  
19           330      (2019) 5:1–5:55. doi:10.1145/3285956.  
20           331      URL <https://doi.org/10.1145/3285956>  
21  
22  
23      332 [11] M. Viroli, G. Audrito, J. Beal, F. Damiani, D. Pianini, Engineering  
24           333      resilient collective adaptive systems by self-stabilisation, ACM  
25           334      Trans. Model. Comput. Simul. 28 (2) (2018) 16:1–16:28. doi:10.1145/  
26           335      3177774.  
27           336      URL <https://doi.org/10.1145/3177774>  
28  
29  
30  
31      337 [12] R. Casadei, M. Viroli, G. Audrito, D. Pianini, F. Damiani, Engineering  
32           338      collective intelligence at the edge with aggregate processes, Eng.  
33           339      Appl. Artif. Intell. 97 (2021) 104081. doi:10.1016/j.engappai.2020.  
34           340      104081.  
35           341      URL <https://doi.org/10.1016/j.engappai.2020.104081>  
36  
37  
38      342 [13] C. Pincioli, G. Beltrame, Buzz: A programming language for robot  
39           343      swarms, IEEE Softw. 33 (4) (2016) 97–100. doi:10.1109/MS.2016.95.  
40           344      URL <https://doi.org/10.1109/MS.2016.95>  
41  
42  
43      345 [14] Y. A. Alrahman, R. D. Nicola, M. Loreti, Programming interactions  
44           346      in collective adaptive systems by relying on attribute-based communica-  
45           347      tion, Sci. Comput. Program. 192 (2020) 102428. doi:10.1016/j.  
46           348      scico.2020.102428.  
47           349      URL <https://doi.org/10.1016/j.scico.2020.102428>  
48  
49  
50      350 [15] O. Boissier, R. H. Bordini, J. Hubner, A. Ricci, Multi-agent oriented  
51           351      programming: programming multi-agent systems using JaCaMo, MIT  
52           352      Press, 2020.  
53  
54  
55      353 [16] A. Ricci, P. Haller (Eds.), Programming with Actors - State-of-the-Art  
56           354      and Research Perspectives, Vol. 10789 of Lecture Notes in Computer  
57           355      Science, Springer, 2018. doi:10.1007/978-3-030-00302-9.  
58           356      URL <https://doi.org/10.1007/978-3-030-00302-9>

- 1  
2  
3  
4  
5  
6  
7 [357] [17] R. Newton, G. Morrisett, M. Welsh, The regiment macroprogramming  
8 system, in: T. F. Abdelzaher, L. J. Guibas, M. Welsh (Eds.), Pro-  
9 ceedings of the 6th International Conference on Information Processing  
10 in Sensor Networks, IPSN 2007, Cambridge, Massachusetts, USA, April  
11 25-27, 2007, ACM, 2007, pp. 489–498. doi:10.1145/1236360.1236422.  
12 URL <https://doi.org/10.1145/1236360.1236422>  
13  
14  
15 [363] [18] G. Audrito, R. Casadei, F. Damiani, G. Salvaneschi, M. Viroli, Func-  
16 tional programming for distributed systems with XC, in: K. Ali, J. Vitek  
17 (Eds.), 36th European Conference on Object-Oriented Programming,  
18 ECOOP 2022, June 6-10, 2022, Berlin, Germany, Vol. 222 of LIPIcs,  
19 Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 20:1–20:28.  
20 doi:10.4230/LIPIcs.ECOOP.2022.20.  
21 URL <https://doi.org/10.4230/LIPIcs.ECOOP.2022.20>  
22  
23  
24 [370] [19] J. Beal, J. Bachrach, Infrastructure for engineered emergence on sen-  
25 sor/actuator networks, IEEE Intell. Syst. 21 (2) (2006) 10–19. doi:  
26 10.1109/MIS.2006.29.  
27 URL <https://doi.org/10.1109/MIS.2006.29>  
28  
29  
30 [374] [20] D. Pianini, M. Viroli, J. Beal, Protelis: practical aggregate program-  
31 ming, in: R. L. Wainwright, J. M. Corchado, A. Bechini, J. Hong (Eds.),  
32 Proceedings of the 30th Annual ACM Symposium on Applied Comput-  
33 ing, Salamanca, Spain, April 13-17, 2015, ACM, 2015, pp. 1846–1853.  
34 doi:10.1145/2695664.2695913.  
35 URL <https://doi.org/10.1145/2695664.2695913>  
36  
37  
38 [380] [21] G. Audrito, FCPP: an efficient and extensible field calculus framework,  
39 in: IEEE International Conference on Autonomic Computing and Self-  
40 Organizing Systems, ACSOS 2020, Washington, DC, USA, August 17-  
41 21, 2020, IEEE, 2020, pp. 153–159. doi:10.1109/ACSOS49614.2020.  
42 00037.  
43 URL <https://doi.org/10.1109/ACSOS49614.2020.00037>  
44  
45  
46 [386] [22] R. Casadei, M. Viroli, G. Audrito, F. Damiani, Fscafì : A core cal-  
47 culus for collective adaptive systems programming, in: T. Margaria,  
48 B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verifi-  
49 cation and Validation: Engineering Principles - 9th International Sym-  
50 posium on Leveraging Applications of Formal Methods, ISoLA 2020,  
51 Rhodes, Greece, October 20-30, 2020, Proceedings, Part II, Vol. 12477  
52 of Lecture Notes in Computer Science, Springer, 2020, pp. 344–360.  
53 doi:10.1007/978-3-030-61470-6\_21.  
54 URL [https://doi.org/10.1007/978-3-030-61470-6\\_21](https://doi.org/10.1007/978-3-030-61470-6_21)  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

- 1  
2  
3  
4  
5  
6  
7 [23] J. Beal, S. Dulman, K. Usbeck, M. Viroli, N. Correll, Organizing  
8 the aggregate: Languages for spatial computing, CoRR abs/1202.5509.  
9 arXiv:1202.5509.  
10 URL <http://arxiv.org/abs/1202.5509>
- 11  
12 [24] R. Roestenburg, R. Bakker, R. Williams, Akka in Action, 1st Edition,  
13 Manning Publications Co., USA, 2015.
- 14  
15 [25] M. Odersky, M. Zenger, Scalable component abstractions, in: R. E.  
16 Johnson, R. P. Gabriel (Eds.), Proceedings of the 20th Annual ACM  
17 SIGPLAN Conference on Object-Oriented Programming, Systems,  
18 Languages, and Applications, OOPSLA 2005, October 16-20, 2005, San  
19 Diego, CA, USA, ACM, 2005, pp. 41–57. doi:10.1145/1094811.  
20 1094815.  
21 URL <https://doi.org/10.1145/1094811.1094815>
- 22  
23 [26] J. Hunt, Cake Pattern, Springer International Publishing, Cham, 2013,  
24 pp. 115–119. doi:10.1007/978-3-319-02192-8\_13.  
25 URL [https://doi.org/10.1007/978-3-319-02192-8\\_13](https://doi.org/10.1007/978-3-319-02192-8_13)
- 26  
27 [27] G. Audrito, R. Casadei, F. Damiani, M. Viroli, Computation against  
28 a neighbour: Addressing large-scale distribution and adaptivity with  
29 functional programming and scala (2020). doi:10.48550/ARXIV.2012.  
30 08626.  
31 URL <https://arxiv.org/abs/2012.08626>
- 32  
33 [28] J. Beal, J. Bachrach, D. Vickery, M. M. Tobenkin, Fast self-healing  
34 gradients, in: R. L. Wainwright, H. Haddad (Eds.), Proceedings of the  
35 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara,  
36 Brazil, March 16-20, 2008, ACM, 2008, pp. 1969–1975. doi:10.1145/  
37 1363686.1364163.  
38 URL <https://doi.org/10.1145/1363686.1364163>
- 39  
40 [29] T. D. Wolf, T. Holvoet, Designing self-organising emergent systems  
41 based on information flows and feedback-loops, in: Proceedings of the  
42 First International Conference on Self-Adaptive and Self-Organizing  
43 Systems, SASO 2007, Boston, MA, USA, July 9-11, 2007, IEEE Com-  
44 puter Society, 2007, pp. 295–298. doi:10.1109/SASO.2007.16.  
45 URL <https://doi.org/10.1109/SASO.2007.16>
- 46  
47 [30] L. Testa, G. Audrito, F. Damiani, G. Torta, Aggregate pro-  
48 cesses as distributed adaptive services for the industrial internet  
49 of things, Pervasive and Mobile Computing 85 (2022) 101658.
- 50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

- 1  
 2  
 3  
 4  
 5  
 6       doi:<https://doi.org/10.1016/j.pmcj.2022.101658>.  
 7       URL     <https://www.sciencedirect.com/science/article/pii/S1574119222000797>  
 8  
 9  
 10  
 11 [31] D. Gurnell, The Type Astronaut's Guide to Shapeless, Lulu.com, 2017.  
 12       URL <https://books.google.it/books?id=c9evDgAAQBAJ>  
 13  
 14 [32] D. Pianini, R. Casadei, M. Viroli, S. Mariani, F. Zambonelli, Time-fluid  
 15 field-based coordination through programmable distributed schedulers,  
 16 Log. Methods Comput. Sci. 17 (4). doi:[10.46298/lmcs-17\(4:13\)2021](https://doi.org/10.46298/lmcs-17(4:13)2021).  
 17       URL [https://doi.org/10.46298/lmcs-17\(4:13\)2021](https://doi.org/10.46298/lmcs-17(4:13)2021)  
 18  
 19  
 20 [33] D. Pianini, S. Montagna, M. Viroli, Chemical-oriented simulation of  
 21 computational systems with ALCHEMIST, J. Simulation 7 (3) (2013)  
 22 202–215. doi:[10.1057/jos.2012.27](https://doi.org/10.1057/jos.2012.27).  
 23       URL <https://doi.org/10.1057/jos.2012.27>  
 24  
 25  
 26 [34] M. Viroli, R. Casadei, D. Pianini, Simulating large-scale aggregate MASs  
 27 with Alchemist and Scala, in: M. Ganzha, L. A. Maciaszek, M. Pa-  
 28 przycki (Eds.), Proceedings of the 2016 Federated Conference on Com-  
 29 puter Science and Information Systems, FedCSIS 2016, Gdańsk, Poland,  
 30 September 11-14, 2016, Vol. 8 of Annals of Computer Science and Infor-  
 31 mation Systems, IEEE, 2016, pp. 1495–1504. doi:[10.15439/2016F407](https://doi.org/10.15439/2016F407).  
 32       URL <https://doi.org/10.15439/2016F407>  
 33  
 34  
 35  
 36 [35] R. Casadei, M. Viroli, Programming actor-based collective adaptive  
 37 systems, in: A. Ricci, P. Haller (Eds.), Programming with Actors  
 38 - State-of-the-Art and Research Perspectives, Vol. 10789 of Lecture  
 39 Notes in Computer Science, Springer, 2018, pp. 94–122. doi:[10.1007/978-3-030-00302-9\\_4](https://doi.org/10.1007/978-3-030-00302-9_4).  
 40       URL [https://doi.org/10.1007/978-3-030-00302-9\\_4](https://doi.org/10.1007/978-3-030-00302-9_4)  
 41  
 42  
 43  
 44  
 45 [36] D. Pianini, R. Casadei, M. Viroli, A. Natali, Partitioned integration and  
 46 coordination via the self-organising coordination regions pattern, Future  
 47 Gener. Comput. Syst. 114 (2021) 44–68. doi:[10.1016/j.future.2020.07.032](https://doi.org/10.1016/j.future.2020.07.032).  
 48       URL <https://doi.org/10.1016/j.future.2020.07.032>  
 49  
 50  
 51  
 52 [37] G. Aguzzi, R. Casadei, M. Viroli, Towards reinforcement learning-based  
 53 aggregate computing, in: M. H. ter Beek, M. Sirjani (Eds.), Coordina-  
 54 tion Models and Languages - 24th IFIP WG 6.1 International Con-  
 55 ference, COORDINATION 2022, Held as Part of the 17th Interna-  
 56 tional Federated Conference on Distributed Computing Techniques, Dis-  
 57 CoTec 2022, Lucca, Italy, June 2022, Proceedings, Vol. 13271  
 58  
 59  
 60  
 61  
 62  
 63  
 64  
 65

- 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](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](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>

- 1  
2  
3  
4  
5  
6  
7 [504] [44] R. Casadei, scafi/artifact-2021-ecoop-xc: v1.2 (2022). doi:10.5281/  
8 ZENODO.6538810.  
9 URL <https://zenodo.org/record/6538810>  
10  
11 [507] [45] R. Casadei, scafi/artifact-2021-ecoop-smartc: v1.2 (2022). doi:10.  
12 5281/ZENODO.6538822.  
13 URL <https://zenodo.org/record/6538822>  
14  
15 [510] [46] G. Aguzzi, D. Pianini, cric96/experiment-2022-ieee-decentralised-  
16 system: 1.0.1 (2022). doi:10.5281/ZENODO.6477039.  
17 URL <https://zenodo.org/record/6477039>  
18  
19 [513] [47] A. Paulos, S. Dasgupta, J. Beal, Y. Mo, K. D. Hoang, L. J. Bryan, P. P.  
20 Pal, R. E. Schantz, J. Schewe, R. K. Sitaraman, A. Wald, C. Wayllace,  
21 W. Yeoh, A framework for self-adaptive dispersal of computing services,  
22 in: IEEE 4th International Workshops on Foundations and Applications  
23 of Self\* Systems, FAS\*W@SASO/ICCAC 2019, Umea, Sweden, June 16-  
24 20, 2019, IEEE, 2019, pp. 98–103. doi:10.1109/FAS-W.2019.00036.  
25 URL <https://doi.org/10.1109/FAS-W.2019.00036>  
26  
27 [520] [48] J. Beal, K. Usbeck, J. P. Loyall, M. Rowe, J. M. Metzler, Adaptive  
28 opportunistic airborne sensor sharing, ACM Trans. Auton. Adapt. Syst.  
29 13 (1) (2018) 6:1–6:29. doi:10.1145/3179994.  
30 URL <https://doi.org/10.1145/3179994>  
31  
32 [524] [49] R. Casadei, M. Viroli, G. Audrito, D. Pianini, F. Damiani, Aggregate  
33 processes in field calculus, in: H. R. Nielson, E. Tuosto (Eds.), Coordi-  
34 nation Models and Languages - 21st IFIP WG 6.1 International Con-  
35 ference, COORDINATION 2019, Held as Part of the 14th International  
36 Federated Conference on Distributed Computing Techniques, DisCoTec  
37 2019, Kongens Lyngby, Denmark, June 17-21, 2019, Proceedings, Vol.  
38 11533 of Lecture Notes in Computer Science, Springer, 2019, pp. 200–  
39 217. doi:10.1007/978-3-030-22397-7\\_12.  
40 URL [https://doi.org/10.1007/978-3-030-22397-7\\_12](https://doi.org/10.1007/978-3-030-22397-7_12)  
41  
42 [533] [50] M. Odersky, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman,  
43 M. Zenger, et al., An overview of the scala programming language, Tech.  
44 rep. (2004).  
45  
46 [536] [51] B. G. Humm, R. S. Engelschall, Language-oriented programming via  
47 DSL stacking, in: J. A. M. Cordeiro, M. Virvou, B. Shishkov (Eds.),  
48 ICSOFT 2010 - Proceedings of the Fifth International Conference on

- 6                 539         Software and Data Technologies, Volume 2, Athens, Greece, July 22-24,  
 7                 540         2010, SciTePress, 2010, pp. 279–287.
- 8
- 9
- 10                541 [52] D. Gelernter, Generative communication in linda, ACM Trans. Program.  
 11                542         Lang. Syst. 7 (1) (1985) 80–112. doi:10.1145/2363.2433.  
 12                543         URL <https://doi.org/10.1145/2363.2433>
- 13
- 14
- 15                544 [53] G. Audrito, R. Casadei, F. Damiani, V. Stolz, M. Viroli, Adaptive dis-  
 16                545         tributed monitors of spatial properties for cyber-physical systems, J.  
 17                546         Syst. Softw. 175 (2021) 110908. doi:10.1016/j.jss.2021.110908.  
 18                547         URL <https://doi.org/10.1016/j.jss.2021.110908>
- 19
- 20
- 21                548 [54] S. Doeraene, Cross-platform language design in scala.js (keynote), in:  
 22                549         S. Erdweg, B. C. d. S. Oliveira (Eds.), Proceedings of the 9th ACM  
 23                550         SIGPLAN International Symposium on Scala, SCALA@ICFP 2018, St.  
 24                551         Louis, MO, USA, September 28, 2018, ACM, 2018, p. 1. doi:10.1145/  
 25                552         3241653.3266230.  
 26                553         URL <https://doi.org/10.1145/3241653.3266230>
- 27
- 28
- 29
- 30                554 [55] G. Aguzzi, R. Casadei, N. Maltoni, D. Pianini, M. Viroli, Scafi-web:  
 31                555         A web-based application for field-based coordination programming, in:  
 32                556         F. Damiani, O. Dardha (Eds.), Coordination Models and Languages -  
 33                557         23rd IFIP WG 6.1 International Conference, COORDINATION 2021,  
 34                558         Held as Part of the 16th International Federated Conference on Dis-  
 35                559         tributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June  
 36                560         14-18, 2021, Proceedings, Vol. 12717 of Lecture Notes in Computer Sci-  
 37                561         ence, Springer, 2021, pp. 285–299. doi:10.1007/978-3-030-78142-2\\_\\_18.  
 38                562         URL [https://doi.org/10.1007/978-3-030-78142-2\\_18](https://doi.org/10.1007/978-3-030-78142-2_18)
- 39
- 40
- 41
- 42
- 43
- 44                563 [56] L. Bettini, Implementing Domain-Specific Languages with Xtext and  
 45                564         Xtend, Birmingham, ISBN: 9781786464965, Packt Publishing Ltd., UK,  
 46                565         2016.
- 47
- 48
- 49
- 50                566 [57] G. E. Mobus, M. C. Kalton, Principles of Systems Science, Springer  
 51                567         Publishing Company, Incorporated, 2014.
- 52
- 53
- 54                568 [58] F. Yates, Self-Organizing Systems: The Emergence of Order, Life Sci-  
 55                569         ence Monographs, Springer US, 2012.  
 56                570         URL <https://books.google.it/books?id=IiTvBwAAQBAJ>
- 57
- 58
- 59                571 [59] D. Miorandi, V. Maltese, M. Rovatsos, A. Nijholt, J. Stewart, Social  
 60                572         Collective Intelligence: Combining the Powers of Humans and Machines  
 61                573         29

1  
2  
3  
4  
5  
6 574 to Build a Smarter Society, Springer Publishing Company, Incorporated,  
7 575 2014.  
8  
9

- 10 576 [60] S. Kalantari, E. Nazemi, B. Masoumi, Emergence phenomena in self-  
11 577 organizing systems: a systematic literature review of concepts, re-  
12 578 searches, and future prospects, *J. Organ. Comput. Electron. Commer.*  
13 579 30 (3) (2020) 224–265. doi:10.1080/10919392.2020.1748977.  
14 580 URL <https://doi.org/10.1080/10919392.2020.1748977>

17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

# SCAFI: a Scala DSL and Toolkit for Aggregate Programming

Roberto Casadei<sup>a</sup>, Mirko Viroli<sup>a</sup>, Gianluca Aguzzi<sup>a</sup>, Danilo Pianini<sup>a</sup>

<sup>a</sup>*Alma Mater Studiorum—Università di Bologna, Italy  
{robby.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	<a href="https://github.com/scafi/scafi/releases/tag/v1.1.5">https://github.com/scafi/scafi/releases/tag/v1.1.5</a>
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	<a href="http://scafi.github.io/docs/">http://scafi.github.io/docs/</a>
C9	Support email for questions	robby.casadei@unibo.it

Table 1: Code metadata (mandatory)

Nr.	(Executable) software meta-data description	
S1	Current software version	1.1.5
S2	Permanent link to executables of this version	<a href="https://index.scala-lang.org/scafi/scafi/artifacts/">https://index.scala-lang.org/scafi/scafi/artifacts/</a>
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	<a href="http://scafi.github.io/docs/">http://scafi.github.io/docs/</a>
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.

1  
2  
3  
4  
5     24 By letting every device in the system work according to the aggregate ex-  
6     25 ecution model and repeatedly evaluate the aggregate program against its  
7     26 up-to-date local context, it is possible to promote the emergence of robust,  
8     27 collective behaviour [11, 12].

9  
10    28 With respect to other approaches for collective adaptive systems pro-  
11    29 gramming [6, 13, 14] and more classical approaches for multi-agent (e.g.,  
12    30 JaCaMo [15]) and distributed systems programming (e.g. actors [16]), ag-  
13    31 gregate computing arguably provides benefits to development productivity  
14    32 as a result of the following: (i) *macro-level stance* [17], promoting the ability  
15    33 to address system-level behaviour globally; (ii) *compositionality*, promoting  
16    34 construction of complex behaviour out of simpler behaviours; (iii) *formality*,  
17    35 enabling theoretical investigations and analyses; and (iv) *practicality*, with  
18    36 tools supporting actual *programming* and simulation of resulting collective  
19    37 adaptive systems. A comparison with metrics against traditional approaches  
20    38 can be found in [18].

21  
22    39 So, engineering *aggregate systems* involves devising an aggregate program  
23    40 and setting up the *aggregate computing distributed protocol* for its collective  
24    41 execution according to the aggregate execution model. In practice, the ag-  
25    42 gregate program could be written in any programming framework featuring  
26    43 library-level or programming-level aggregate computing mechanisms (e.g.,  
27    44 [19, 20, 21, 22]). Then, the system should be evaluated and tested by sim-  
28    45 ulation before getting deployed on the execution platform of choice. Proper  
29    46 software tooling is essential to support these phases and hence the investi-  
30    47 gation of new self-organising algorithms and variants or extensions of the  
31    48 programming model, promoting scientific and technological progress.

32  
33    49 In the following, we present the SCAFI (*Scala-Fields*) software<sup>1</sup>: an ag-  
34    50 gregate programming toolkit that comprises an internal DSL (language and  
35    51 virtual machine) as well as supporting components for the simulation and  
36    52 execution of aggregate systems.

37  
38    53 **2. Software description**

39  
40    54 SCAFI is a multi-module Scala project hosted on GitHub<sup>2</sup>. It pro-  
41    55 vides a DSL and API modules for writing, testing, and running aggre-  
42    56 gate programs, namely programs expressed according to the aggregate pro-  
43    57 gramming paradigm [7, 8]. Stable versions of SCAFI are delivered through  
44    58 the *Maven Central Repository*. All the artifacts are collected under group

---

55    1<sup>1</sup><https://scafi.github.io>

56    2<sup>2</sup><https://github.com/scafi/scafi>

1  
2  
3  
4  
5     59 **it.unibo.scafi**. SCAFI’s build process and dependency management lever-  
6     60 ages the *Simple Build Tool (SBT)*, and a continuous integration/delivery  
7     61 pipeline on *Github Actions (GHA)* is in place to ensure that changes do not  
8     62 break existing functionality. SCAFI cross-compiles for Scala 2.11, 2.12, 2.13  
9     63 and targets both the JVM and the JavaScript platform (through *Scala.js*).  
10    64 Besides functional testing, the quality assurance pipeline includes tools that  
11    65 enforce a consistent programming style (*ScalaStyle*), perform static analysis  
12    66 for early intercepting code smells (*codiga.io*), track and report code coverage  
13    67 (*codecov.io*), and enforce git commit messages consistency (*commitlint*).  
14  
15  
16

17     68 *2.1. Software Architecture*

18  
19     69 The high-level architecture of SCAFI is depicted in Figure 1. It consists  
20     70 of the following main components (where each component is an SBT module  
21     71 and deployable artifact):  
22  
23

- 24     72     ● **scafi-commons** — provides basic abstractions and utilities (e.g., spatial  
25     73 and temporal abstractions);  
26  
27     74     ● **scafi-core** — provides an aggregate programming DSL (syntax, se-  
28     75 mantics, and a virtual machine for evaluation of programs), together  
29     76 with a “standard library” of reusable functions;  
30  
31     77     ● **scafi-stdlib-ext** — provides extra library functionality that requires  
32     78 external dependencies and is hence kept separated from the minimalist  
33     79 **scafi-core**;  
34  
35     77     ● **scafi-simulator**: provides basic support for simulating aggregate sys-  
36     78 tems;  
37  
38     80     ● **scafi-simulator-gui** — provides a GUI for visualising and interact-  
39     81 ing with simulations of aggregate systems;  
40  
41     82     ● **spala** (“spatial Scala”—i.e., a general Aggregate Computing plat-  
42     83 form<sup>3</sup>) — provides an actor-based aggregate computing middleware  
43     84 (independent of the SCAFI DSL and potentially applicable to other ag-  
44     85 gregate programming languages as well) based on the Akka toolkit [24];  
45  
46     84     ● **scafi-distributed** — ScaFi integration-layer for **spala**, which can  
47     85 be leveraged to set up actor-based deployments of SCAFI-programmed  
48     86 systems.  
49  
50  
51  
52  
53  
54  
55

---

56     56     <sup>3</sup>Aggregate computing is rooted in spatial computing [23].  
57  
58  
59  
60  
61  
62  
63  
64  
65

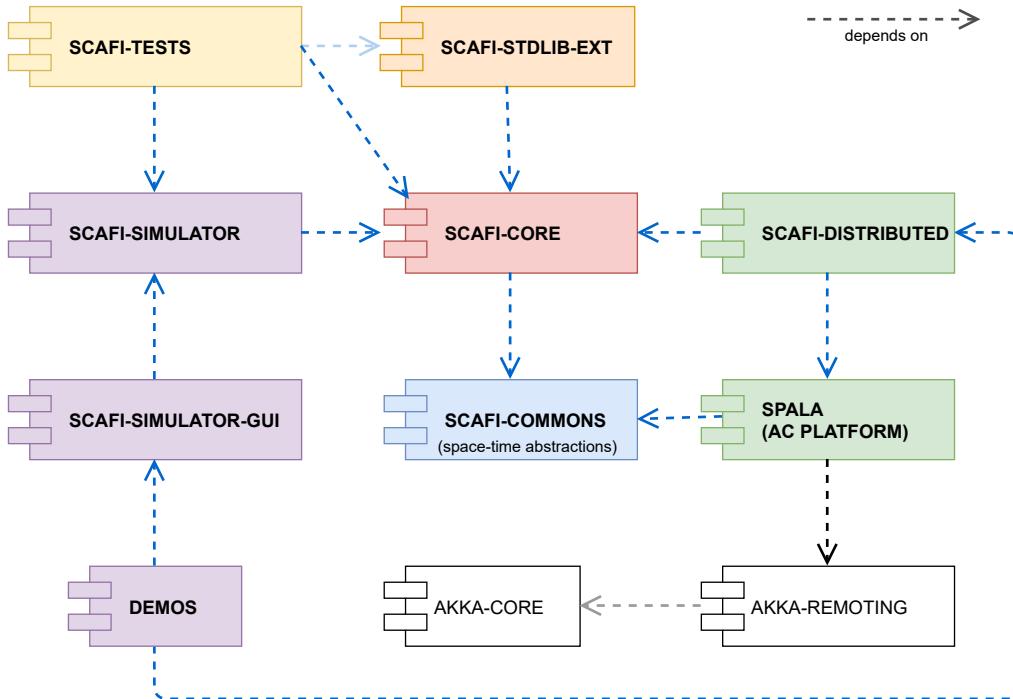


Figure 1: High-level architecture of the SCAFI toolkit.

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**.

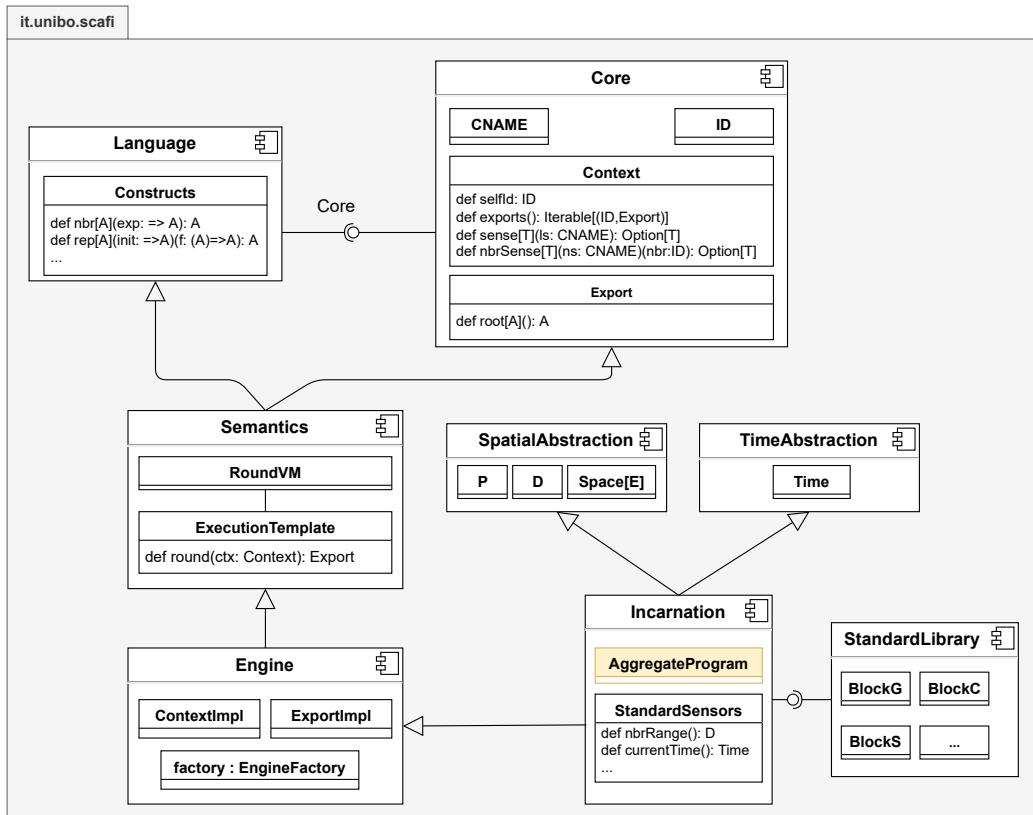


Figure 2: Design of the core of SCAFI (DSL).

1  
2  
3  
4  
5        107 *2.2. Software Functionalities*  
6        108 *Expressing aggregate programs through a Scala DSL*  
7        109      Module `scafi-core` exposes, through incarnations, an  
8        110 `AggregateProgram` trait that provides access to aggregate program-  
9        111 ming constructs—following a variant of the field calculus [10, 8] formalised  
10        112 in [22, 27]. This single program defines – from a global perspective – the  
11        113 collective adaptive behaviour of an entire ensemble of computational devices.  
12        114 Besides the core constructs, this module also provides “standard library”  
13        115 traits providing access to reusable functions of aggregate functionality. For  
14        116 instance, by mixing trait `Gradients` into an `AggregateProgram` subclass,  
15        117 a developer gets access to *gradient functions* [28, 11], used to continuously  
16        118 compute (over space and time) the self-healing field of minimum distances  
17        119 of each node from a set of source nodes. Several such traits are available  
18        120 to provide other key building blocks for self-organising applications [29, 11]  
19        121 (e.g., `BlockG` for gradient-wise information propagation, `BlockC` for gradient-  
20        122 wise information collection, `BlockS` for sparse choice or leader election)  
21        123 or experimental language features (e.g., the `spawn` function for concurrent  
22        124 aggregate processes [12, 30], for modelling independent and overlapping  
23        125 aggregate computations). Even more functionality is available in module  
24        126 `scafi-stdlib-ext`, which currently provides Shapeless-leveraging [31]  
25        127 typeclasses to extend `Boundedness` constraints (required by some library  
26        128 functions) to arbitrary product types.  
27  
28  
29  
30  
31  
32  
33  
34        129 *Virtual machine for the local execution of aggregate programs*  
35        130      An `AggregateProgram` instance is a function mapping a `Context` (the set  
36        131 of inputs needed by an individual device to properly evaluate the program lo-  
37        132 cally) to an `Export` (the tree of values that has to be shared with neighbours  
38        133 to effectively coordinate and promote emergence of collective behaviours).  
39        134 Using this API, a developer can integrate “aggregate functionality” into its  
40        135 system—what remains to be specified are the details of the aggregate ex-  
41        136 ecution model and the communication among devices, that may change in  
42        137 different applications. Devices must continuously run the aggregate program,  
43        138 but the scheduling of these computation rounds can be tuned as the applica-  
44        139 tion needs [32]. `Exports` must be shared with neighbouring devices to allow  
45        140 them to properly set up their `Contexts`, but the network protocol to be used  
46        141 to do so can be selected independently of the program.  
47  
48  
49  
50  
51  
52        142 *Simulation support*  
53        143      In order to simulate an “aggregate system”, it is necessary to (i) define the  
54        144 set of computational devices that make up the aggregate, including their sen-  
55        145 sors and actuators; (ii) define the aggregate topology, i.e., some application-

1  
2  
3  
4  
5     146 specific *neighbouring relationship* from which the set of *neighbours* of each  
6     147 device can be determined; (iii) define the aggregate program to be executed;  
7     148 (iv) define a certain dynamics of the system by proper scheduling of computa-  
8     149 tion rounds, and the environment by proper scheduling of changes in sensor  
9     150 values. Module **scafi-simulator** provides this basic support. It exposes  
10    151 some factory methods to configure simulations properly (e.g., it supports ad-  
11    152 hoc and spatial distance-based connectivity rules) and an API to run and  
12    153 interact with simulations. Then, module **scafi-simulator-gui** provides a  
13    154 convenient graphical user interface to launch and visually show simulations in  
14    155 execution. We remark that these modules currently support basic simulation  
15    156 scenarios and are mainly meant for quick experiments or as a starting basis  
16    157 for ad-hoc simulation frameworks; a further option for sophisticated simula-  
17    158 tions and data analysis is to use SCAFI within the Alchemist simulator for  
18    159 pervasive computing systems [33, 34].

19  
20  
21  
22  
23     160 *Experimental or work-in-progress features: 3D simulation frontend and actor-*  
24     161 *based middleware*

25  
26     162 SCAFI also includes a front-end for 3D simulations (**renderer-3d**), which  
27     163 are already supported by an execution perspective.

28  
29     164 Regarding the construction of actual systems, SCAFI provides an actor-  
30     165 based implementation of the aggregate execution model [35], in the **spala**  
31     166 (**Spatial Scala**) module, which is instrumental for integrating aggregate com-  
32     167 puting into existing systems and distributed architectures [35]. Indeed, ag-  
33     168 gregate computing systems can be designed, deployed, and executed ac-  
34     169 cording to different architectural styles and concrete architectures [9]. So,  
35  
36     170 SCAFI provides *two* main implementations of the middleware, in package  
37     171 **it.unibo.scafi.distrib.actor**, for purely peer-to-peer (sub-package **p2p**)  
38     172 and server-based designs (sub-package **server**). The main abstraction is the  
39     173 **DeviceActor**, which exposes a message-based interface for controlling and  
40     174 interacting with an individual logical node of the aggregate system. Then, an  
41     175 object-oriented façade API is provided to set up a system of middleware-level  
42     176 actors.

43  
44  
45  
46     177 **3. Illustrative Examples**

47  
48     178 *3.1. Hello SCAFI: building an aggregate system that computes a gradient,*  
49     179 *from scratch*

50  
51     180 This complete example, shown in Figure 3 and available online<sup>4</sup>, illus-  
52     181 trates how SCAFI can be used to program a (simulated) aggregate system

---

53  
54  
55     182 <sup>4</sup><https://github.com/scafi/hello-scafi>

1  
2  
3  
4  
5     182 for computing a self-stabilising *gradient* field [28, 11] where the output of  
6     183 each device self-stabilises to its minimum distance from an appointed *source*  
7     184 device. Development comes into two parts: (i) definition of the aggregate  
8     185 program, namely the logic of collective behaviour (Figure 3a)<sup>5</sup><sup>6</sup>; and (ii) def-  
9     186 inition of an “aggregate execution protocol” determining how devices com-  
10     187 municate and act upon their environment (Figure 3b).

11  
12  
13     188 *3.2. Self-organising Coordination Regions in Simulation*

14  
15     189 As a more complex example, consider a SCAFI implementation of the  
16     190 Self-Organising Coordination Regions (SCR) pattern [36]. The idea of SCR  
17     191 is to organise a distributed activity into multiple spatial *regions* (inducing  
18     192 a partition of the system), each one controlled by a *leader* device, which  
19     193 collects data from the area members and spreads decisions to enact some  
20     194 area-wide policy. This pattern can be easily implemented in SCAFI using its  
21     195 standard library functions, and simulated through the feature provided by  
22     196 **scafi-simulator**.

23  
24     197 For instance, consider the following scenario: temperature monitoring  
25     198 and control in a large environment. For distributed summarisation, we could  
26     199 create areas of uniform sizes and let the devices collectively compute the  
27     200 area’s average temperature. Then, we could create an alarm based on col-  
28     201 lective information, for more coarse-grained analysis and intervention. We  
29     202 implemented this scenario in the repository<sup>7</sup>: Figure 4 shows a simple SCAFI  
30     203 implementation of SCR and a snapshot taken from the SCAFI simulator.

31  
32     204 **4. Impact**

33  
34     205 SCAFI has been used in aggregate computing-related research [12, 18,  
35     206 37, 38, 39, 40, 41, 42, 43, 27], touching themes such as software engi-  
36     207 neering, computational models, and distributed systems/algorithms. This  
37     208 thread has also several intersections with fields like multi-agent systems, self-  
38     209 organisation, collective intelligence, and scenarios like the Internet of Things,  
39     210 cyber-physical systems, and edge computing. Artifacts published on perma-  
40     211 nent repositories (like Zenodo) using SCAFI include [44, 45, 46]. Aggregate  
41     212 programming languages have been used in industry [47, 48]. The impact of  
42     213 SCAFI can be understood in terms of existing and prospective contributions,  
43     214 discussed in the following.

50  
51     52     For a detailed explanation of this gradient implementation, please refer to e.g. [12].

52     53     6Concerning source code listings, we highlight symbols as follows: we use blue for Scala  
54     54 keywords, red for SCAFI DSL constructs, purple for SCAFI library functions, and brown  
55     55 for other SCAFI API symbols (e.g., types, objects, constants, and methods).

56     56     7<https://github.com/scafi/scafi-softwarex-scr-example>

```

1
2
3
4
5
6
7 // 1. Define/import an incarnation, which provides ScaFi types and classes
8 object MyIncarnation extends
9     it.unibo.scafi.incarnations.BasicAbstractIncarnation
10    // 2. Bring into scope the stuff from the chosen incarnation
11    import MyIncarnation._
12    // 3. Define an "aggregate program" using the ScaFi DSL
13    // by extending AggregateProgram and specifying a "main" expression
14    class GradientProgram extends AggregateProgram {
15        def isSource: Boolean = sense("source")
16        override def main(): Any = rep(Double.PositiveInfinity)(d => {
17            mux(isSource){ 0.0 } {
18                foldhoodPlus(Double.PositiveInfinity)(Math.min){ nbr(d) + 1.0 }
19            })
20        })

```

(a) Program definition

```

21 // 4. In your program, implement an "execution loop" whereby
22 // your device or system executes the aggregate program
23 object HelloScafi extends App {
24     case class DeviceState(self: ID, exports: Map[ID, EXPORT],
25         localSensors: Map[CNAME, Any], nbrSensors: Map[CNAME, Map[ID, Any]]) {
26         def updateExport(dev: ID, export: EXPORT): DeviceState =
27             this.copy(exports = exports + (dev -> export))
28     }
29     val devices = 1 to 5 // (1,2,3,4,5), i.e., 5 devices
30     val sourceId = 2 // device 2 is the source of the gradient
31     val scheduling = devices ++ devices ++ devices ++ devices
32     val program = new GradientProgram()
33     def neighboursFrom(id: ID): Seq[Int] = // topology: [1]-[2]-[3]-[4]-[5]
34         Seq(id - 1, id, id + 1).filter(n => n > 0 && n < 6)
35     // Now let's build a simplified system to illustrate the execution model
36     var state: Map[ID, DeviceState] = (for {
37         d <- devices
38     } yield d -> DeviceState(d, Map.empty, Map("source" -> false),
39         Map(NBR_RANGE -> (neighboursFrom(d).toSet[ID]
40             .map(nbr -> nbr -> Math.abs(d - nbr).toDouble)).toMap))).toMap
41     state = state + (sourceId -> state(sourceId).copy(localSensors =
42         state(sourceId).localSensors + ("source" -> true))) // set source
43     // The cycle simulates scheduling&communication by read/write on 'state'
44     for(d <- scheduling){ // run 5 rounds for each device 'd', round-robin
45         // build the local context for device d
46         val ctx = factory.context(selfId = d, exports = state(d).exports,
47             lsens = state(d).localSensors, nbsens = state(d).nbrSensors)
48         println(s"RUN: ${d}\nCONTEXT: ${state(d)}")
49         // run the program against the local context
50         val export = program.round(ctx)
51         // update d's state
52         state += d -> state(d).updateExport(d, export)
53         // Simulate sending of messages to neighbours
54         state(d).nbrSensors(NBR_RANGE).keySet.foreach(
55             nbr -> state += nbr -> state(nbr).updateExport(d, export))
56         println(s"\nEXPORT: ${export}\nOUTPUT: ${export.root()}\n-----")
57     }
58 }

```

(b) System and execution definition

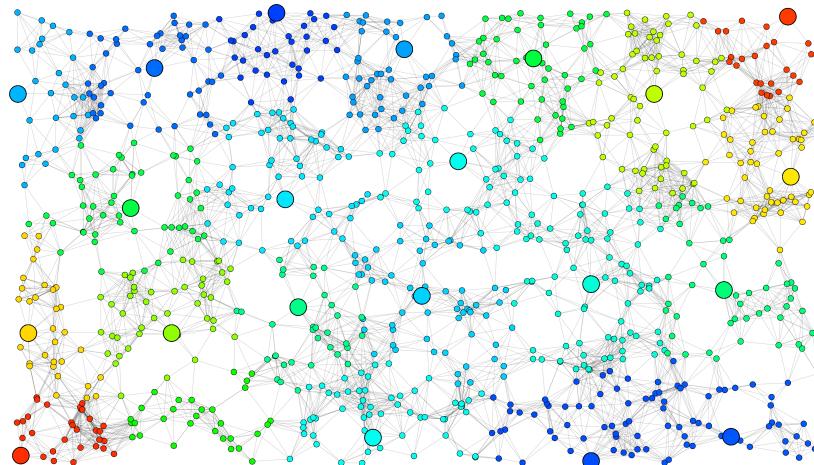
Figure 3: Complete example: an aggregate system computing a gradient.

```

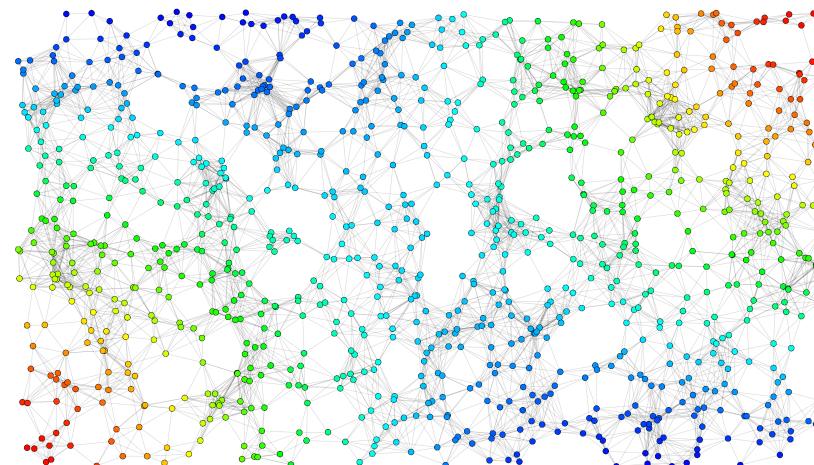
1
2
3
4
5
6
7   class SCR extends AggregateProgram with BuildingBlocks with StandardSensors {
8     val radius = 300 // average area of interest
9     val leader = S(radius, nbrRange)
10    val potential = distanceTo(leader)
11    val averageTemperature = collectMean(potential, temperature)
12    val zoneTemperature = broadcast(leader, averageTemperature)
13    (leader, zoneTemperature)
14    // 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).

1  
2  
3  
4  
5     215 *Interplay between programming language design and foundational research*  
6         216 The implementation of the SCAFI DSL has inspired a variant of the  
7         217 field calculus which arguably supports easier embeddability into mainstream  
8         218 programming languages [22, 27].  
9  
10  
11     219 *Platform for experimenting new aggregate programming language features*  
12         220 SCAFI includes extensions to the basic field calculus. In particular, it  
13         221 supports the *aggregate process* abstraction [12], enabled by the `spawn` con-  
14         222 struct [49], which provides a way to specify a dynamic number of collective  
15         223 computations running on dynamic ensembles of devices. Another exten-  
16         224 sion is the `exchange` primitive [18, 44], which subsumes previous commu-  
17         225 nication primitives (like `nbr` [10]) and enables differentiated messages for  
18         226 neighbours. In general, as the aggregate programming DSL is exposed as a  
19         227 “plain-old library”, it is arguably easier to implement new features, as the  
20         228 developer does not need to deal with parser, compilers, type systems, or  
21         229 language workbenches—of course, at the expense of (syntactic and analytic)  
22         230 constraints exerted by the host language. Moreover, the research orienta-  
23         231 tion of Scala [50] makes it a powerful environment for experimenting new  
24         232 language features and mechanisms.  
25  
26  
27  
28  
29  
30     233 *High-level programming models*  
31         234 The previous discussion makes the case for “DSL stacking” [51]. Indeed,  
32         235 by leveraging the aforementioned aggregate process extension, it is possi-  
33         236 ble to reduce the abstraction gap needed to implement *situated tuples* [42],  
34         237 which is a Linda-like model [52] for coordinating processes where tuples and  
35         238 tuple operations are situated in space. By mapping high-level specifications  
36         239 into aggregate programs, it is sometimes straightforward to develop resilient  
37         240 distributed implementations—as in [53], where translation rules from spatial  
38         241 logic formulas to field calculus expressions enable seamless construction of  
39         242 decentralised monitors for such formulas.  
40  
41  
42  
43  
44     243 *Web-friendliness*  
45         244 By leveraging Scala.js [54], SCAFI can be easily accessed through  
46         245 JavaScript, which promotes cross-platform language design and reuse of func-  
47         246 tionality in the browser (to support web applications without the need of  
48         247 server-side components). This paved the path to SCAFI-WEB [55], a web  
49         248 playground for aggregate programming.  
50  
51  
52  
53     249 *Developer-friendliness*  
54         250 With respect to other programming frameworks for aggregate computing  
55         251 like Proto [19], Protelis [20], and FCPP [21], the SCAFI toolkit provides a

1  
2  
3  
4  
5     252 privileged environment for developers. Proto has been discontinued. Its suc-  
6     253 cessor, Protelis, is a standalone DSL with duck typing and no support for the  
7     254 definition of new data structures, and whose support for syntax highlighting  
8     255 and code completion is only available for the Eclipse IDE (being based on the  
9     256 Xtext framework [56]). Relatively to FCPP, which is based on C++, SCAFI  
10    257 benefits from the higher level of abstraction provided by Scala and the inte-  
11    258 gration with the Java ecosystem. A more detailed account of this comparison  
12    259 between aggregate programming languages can be found in [8, 27].  
13  
14  
15

16     260 *Engineering of complex systems and collective intelligence (and related re-*  
17     261 *search)*

18  
19     262 The paradigm embodied by SCAFI provides a means to explore  
20     263 *complex systems* themes [57] (including collective intelligence [5], self-  
21     264 organisation [58], socio-technical collectives [59], emergence [60], etc.), and  
22     265 to do so by an *engineering and programming perspective*. For instance,  
23     266 in [12] the ability to self-organise into dynamic groups is exploited to provide  
24     267 forms of intelligent behaviour at the edge; in [36], a self-organisation pat-  
25     268 tern has been discovered that enables dynamic adjustment of the diameter  
26     269 of feedback-regulated networks and hence of the level of decentralisation in a  
27     270 system, for intelligent use of resources. In [37], reinforcement learning is used  
28     271 to learn policies for determining what actions to execute, in “holes” of SCAFI  
29     272 programs, to improve the dynamics of collective algorithms. We foresee that  
30     273 accessible software toolkits such as SCAFI aimed at programming collective  
31     274 adaptive systems could have an important role in these research threads.  
32  
33  
34  
35

36  
37     275 **5. Conclusion**  
38

39  
40     276 This paper presents SCAFI, an open-source Scala-based toolkit for aggre-  
41     277 gate computing, enabling the development of collective adaptive systems. It  
42     278 provides an internal DSL for the field calculus, a library of reusable aggre-  
43     279 gate behaviour functions, as well as support components for simulating and  
44     280 executing aggregate systems. Compared to other aggregate programming  
45     281 languages such as Protelis and FCPP, it provides a more high-level platform  
46     282 that might support agile prototyping for research and easier integration with  
47     283 other tools and environments for distributed systems (cf. the Web and An-  
48     284 droid). We believe it represents a valuable tool for potential scientific and  
49     285 technological developments related to intelligent collective systems.  
50  
51

52  
53     286 **Conflict of Interest**  
54

55  
56     287 No conflict of interest exists.  
57  
58  
59  
60  
61  
62  
63  
64  
65

1  
2  
3  
4  
5 288 Acknowledgements  
6

7 This work has been partially supported by the MUR PRIN 2020 Project  
8 “COMMON-WEARS” (2020HCWWLP) and the EU/MUR FSE REACT-  
9 EU PON R&I 2014-2020.  
10

11 We also would like to thank Prof. Ferruccio Damiani and Dr. Giorgio  
12 Audrito for their contribution to the formal underpinnings of the SCAFi DSL,  
13 and the students at the University of Bologna that contributed to the tool.  
14

15  
16 295 References  
17

- 18 [1] G. D. Abowd, Beyond weiser: From ubiquitous to collective computing,  
19 Computer 49 (1) (2016) 17–23. doi:10.1109/MC.2016.22.  
20 URL <https://doi.org/10.1109/MC.2016.22>  
21  
22 [2] M. Satyanarayanan, Pervasive computing: vision and challenges, IEEE  
23 Wirel. Commun. 8 (4) (2001) 10–17. doi:10.1109/98.943998.  
24 URL <https://doi.org/10.1109/98.943998>  
25  
26 [3] J. Ferber, Multi-agent systems - an introduction to distributed artificial  
27 intelligence, Addison-Wesley-Longman, 1999.  
28  
29 [4] J. O. Kephart, D. M. Chess, The vision of autonomic computing, Computer  
30 36 (1) (2003) 41–50. doi:10.1109/MC.2003.1160055.  
31 URL <https://doi.org/10.1109/MC.2003.1160055>  
32  
33 [5] F. He, Y. Pan, Q. Lin, X. Miao, Z. Chen, Collective intelligence: A  
34 taxonomy and survey, IEEE Access 7 (2019) 170213–170225. doi:10.  
35 1109/ACCESS.2019.2955677.  
36 URL <https://doi.org/10.1109/ACCESS.2019.2955677>  
37  
38 [6] R. D. Nicola, S. Jähnichen, M. Wirsing, Rigorous engineering of col-  
39 lective adaptive systems: special section, Int. J. Softw. Tools Technol.  
40 Transf. 22 (4) (2020) 389–397. doi:10.1007/s10009-020-00565-0.  
41 URL <https://doi.org/10.1007/s10009-020-00565-0>  
42  
43 [7] J. Beal, D. Pianini, M. Viroli, Aggregate programming for the internet  
44 of things, Computer 48 (9) (2015) 22–30. doi:10.1109/MC.2015.261.  
45 URL <https://doi.org/10.1109/MC.2015.261>  
46  
47 [8] M. Viroli, J. Beal, F. Damiani, G. Audrito, R. Casadei, D. Pianini, From  
48 distributed coordination to field calculus and aggregate computing, J.  
49 Log. Algebraic Methods Program. 109. doi:10.1016/j.jlamp.2019.  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

- 1  
2  
3  
4  
5       321      100486.  
6       322      URL <https://doi.org/10.1016/j.jlamp.2019.100486>  
7  
8       323 [9] R. Casadei, D. Pianini, A. Placuzzi, M. Viroli, D. Weyns, Pulverization  
9       324      in cyber-physical systems: Engineering the self-organizing logic  
10      325      separated from deployment, Future Internet 12 (11) (2020) 203. doi:  
11      326      10.3390/fi12110203.  
12      327      URL <https://doi.org/10.3390/fi12110203>  
13  
14  
15      328 [10] G. Audrito, M. Viroli, F. Damiani, D. Pianini, J. Beal, A higher-order  
16      329      calculus of computational fields, ACM Trans. Comput. Log. 20 (1)  
17      330      (2019) 5:1–5:55. doi:10.1145/3285956.  
18      331      URL <https://doi.org/10.1145/3285956>  
19  
20  
21      332 [11] M. Viroli, G. Audrito, J. Beal, F. Damiani, D. Pianini, Engineering  
22      333      resilient collective adaptive systems by self-stabilisation, ACM  
23      334      Trans. Model. Comput. Simul. 28 (2) (2018) 16:1–16:28. doi:10.1145/  
24      335      3177774.  
25      336      URL <https://doi.org/10.1145/3177774>  
26  
27  
28      337 [12] R. Casadei, M. Viroli, G. Audrito, D. Pianini, F. Damiani, Engineering  
29      338      collective intelligence at the edge with aggregate processes, Eng.  
30      339      Appl. Artif. Intell. 97 (2021) 104081. doi:10.1016/j.engappai.2020.  
31      340      104081.  
32      341      URL <https://doi.org/10.1016/j.engappai.2020.104081>  
33  
34  
35      342 [13] C. Pincioli, G. Beltrame, Buzz: A programming language for robot  
36      343      swarms, IEEE Softw. 33 (4) (2016) 97–100. doi:10.1109/MS.2016.95.  
37      344      URL <https://doi.org/10.1109/MS.2016.95>  
38  
39  
40      345 [14] Y. A. Alrahman, R. D. Nicola, M. Loreti, Programming interactions  
41      346      in collective adaptive systems by relying on attribute-based communica-  
42      347      tion, Sci. Comput. Program. 192 (2020) 102428. doi:10.1016/j.  
43      348      scico.2020.102428.  
44      349      URL <https://doi.org/10.1016/j.scico.2020.102428>  
45  
46  
47      350 [15] O. Boissier, R. H. Bordini, J. Hubner, A. Ricci, Multi-agent oriented  
48      351      programming: programming multi-agent systems using JaCaMo, MIT  
49      352      Press, 2020.  
50  
51  
52      353 [16] A. Ricci, P. Haller (Eds.), Programming with Actors - State-of-the-Art  
53      354      and Research Perspectives, Vol. 10789 of Lecture Notes in Computer  
54      355      Science, Springer, 2018. doi:10.1007/978-3-030-00302-9.  
55      356      URL <https://doi.org/10.1007/978-3-030-00302-9>

- 1  
2  
3  
4  
5 [357] [17] R. Newton, G. Morrisett, M. Welsh, The regiment macroprogramming  
6 system, in: T. F. Abdelzaher, L. J. Guibas, M. Welsh (Eds.), Proceedings  
7 of the 6th International Conference on Information Processing  
8 in Sensor Networks, IPSN 2007, Cambridge, Massachusetts, USA, April  
9 25-27, 2007, ACM, 2007, pp. 489–498. doi:10.1145/1236360.1236422.  
10 URL <https://doi.org/10.1145/1236360.1236422>
- 11  
12  
13 [363] [18] G. Audrito, R. Casadei, F. Damiani, G. Salvaneschi, M. Viroli, Func-  
14 tional programming for distributed systems with XC, in: K. Ali, J. Vitek  
15 (Eds.), 36th European Conference on Object-Oriented Programming,  
16 ECOOP 2022, June 6-10, 2022, Berlin, Germany, Vol. 222 of LIPIcs,  
17 Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 20:1–20:28.  
18 doi:10.4230/LIPIcs.ECOOP.2022.20.  
19 URL <https://doi.org/10.4230/LIPIcs.ECOOP.2022.20>
- 20  
21  
22 [370] [19] J. Beal, J. Bachrach, Infrastructure for engineered emergence on sen-  
23 sor/actuator networks, IEEE Intell. Syst. 21 (2) (2006) 10–19. doi:  
24 10.1109/MIS.2006.29.  
25 URL <https://doi.org/10.1109/MIS.2006.29>
- 26  
27 [374] [20] D. Pianini, M. Viroli, J. Beal, Protelis: practical aggregate program-  
28 ming, in: R. L. Wainwright, J. M. Corchado, A. Bechini, J. Hong (Eds.), Pro-  
29 ceedings of the 30th Annual ACM Symposium on Applied Comput-  
30 ing, Salamanca, Spain, April 13-17, 2015, ACM, 2015, pp. 1846–1853.  
31 doi:10.1145/2695664.2695913.  
32 URL <https://doi.org/10.1145/2695664.2695913>
- 33  
34 [380] [21] G. Audrito, FCPP: an efficient and extensible field calculus framework,  
35 in: IEEE International Conference on Autonomic Computing and Self-  
36 Organizing Systems, ACSOS 2020, Washington, DC, USA, August 17-  
37 21, 2020, IEEE, 2020, pp. 153–159. doi:10.1109/ACSOS49614.2020.  
38 00037.  
39 URL <https://doi.org/10.1109/ACSOS49614.2020.00037>
- 40  
41 [386] [22] R. Casadei, M. Viroli, G. Audrito, F. Damiani, Fscafì : A core cal-  
42 culus for collective adaptive systems programming, in: T. Margaria,  
43 B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verifi-  
44 cation and Validation: Engineering Principles - 9th International Sym-  
45 posium on Leveraging Applications of Formal Methods, ISoLA 2020,  
46 Rhodes, Greece, October 20-30, 2020, Proceedings, Part II, Vol. 12477  
47 of Lecture Notes in Computer Science, Springer, 2020, pp. 344–360.  
48 doi:10.1007/978-3-030-61470-6\_21.  
49 URL [https://doi.org/10.1007/978-3-030-61470-6\\_21](https://doi.org/10.1007/978-3-030-61470-6_21)
- 50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

- 1  
2  
3  
4  
5 [395] [23] J. Beal, S. Dulman, K. Usbeck, M. Viroli, N. Correll, Organizing  
6 the aggregate: Languages for spatial computing, CoRR abs/1202.5509.  
7 arXiv:1202.5509.  
8  
9 URL <http://arxiv.org/abs/1202.5509>
- 10  
11 [399] [24] R. Roestenburg, R. Bakker, R. Williams, Akka in Action, 1st Edition,  
12 Manning Publications Co., USA, 2015.
- 13  
14 [401] [25] M. Odersky, M. Zenger, Scalable component abstractions, in: R. E.  
15 Johnson, R. P. Gabriel (Eds.), Proceedings of the 20th Annual ACM  
16 SIGPLAN Conference on Object-Oriented Programming, Systems,  
17 Languages, and Applications, OOPSLA 2005, October 16-20, 2005, San  
18 Diego, CA, USA, ACM, 2005, pp. 41–57. doi:10.1145/1094811.  
19 1094815.  
20  
21 URL <https://doi.org/10.1145/1094811.1094815>
- 22  
23  
24 [408] [26] J. Hunt, Cake Pattern, Springer International Publishing, Cham, 2013,  
25 pp. 115–119. doi:10.1007/978-3-319-02192-8\_13.  
26  
27 URL [https://doi.org/10.1007/978-3-319-02192-8\\_13](https://doi.org/10.1007/978-3-319-02192-8_13)
- 28  
29 [411] [27] G. Audrito, R. Casadei, F. Damiani, M. Viroli, Computation against  
30 a neighbour: Addressing large-scale distribution and adaptivity with  
31 functional programming and scala (2020). doi:10.48550/ARXIV.2012.  
32 08626.  
33  
34 URL <https://arxiv.org/abs/2012.08626>
- 35  
36 [416] [28] J. Beal, J. Bachrach, D. Vickery, M. M. Tobenkin, Fast self-healing  
37 gradients, in: R. L. Wainwright, H. Haddad (Eds.), Proceedings of the  
38 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara,  
39 Brazil, March 16-20, 2008, ACM, 2008, pp. 1969–1975. doi:10.1145/  
40 1363686.1364163.  
41  
42 URL <https://doi.org/10.1145/1363686.1364163>
- 43  
44 [422] [29] T. D. Wolf, T. Holvoet, Designing self-organising emergent systems  
45 based on information flows and feedback-loops, in: Proceedings of the  
46 First International Conference on Self-Adaptive and Self-Organizing  
47 Systems, SASO 2007, Boston, MA, USA, July 9-11, 2007, IEEE Com-  
48 puter Society, 2007, pp. 295–298. doi:10.1109/SASO.2007.16.  
49  
50 URL <https://doi.org/10.1109/SASO.2007.16>
- 51  
52 [428] [30] L. Testa, G. Audrito, F. Damiani, G. Torta, Aggregate pro-  
53 cesses as distributed adaptive services for the industrial internet  
54 of things, Pervasive and Mobile Computing 85 (2022) 101658.  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

- 1  
 2  
 3  
 4  
 5       431       doi:<https://doi.org/10.1016/j.pmcj.2022.101658>.  
 6       432       URL     <https://www.sciencedirect.com/science/article/pii/S1574119222000797>  
 7  
 8  
 9       434 [31] D. Gurnell, The Type Astronaut's Guide to Shapeless, Lulu.com, 2017.  
 10      435       URL <https://books.google.it/books?id=c9evDgAAQBAJ>  
 11  
 12      436 [32] D. Pianini, R. Casadei, M. Viroli, S. Mariani, F. Zambonelli, Time-fluid  
 13      437       field-based coordination through programmable distributed schedulers,  
 14      438       Log. Methods Comput. Sci. 17 (4). doi:[10.46298/lmcs-17\(4:13\)2021](https://doi.org/10.46298/lmcs-17(4:13)2021).  
 15      439       URL [https://doi.org/10.46298/lmcs-17\(4:13\)2021](https://doi.org/10.46298/lmcs-17(4:13)2021)  
 16  
 17  
 18      440 [33] D. Pianini, S. Montagna, M. Viroli, Chemical-oriented simulation of  
 19      441       computational systems with ALCHEMIST, J. Simulation 7 (3) (2013)  
 20      442       202–215. doi:[10.1057/jos.2012.27](https://doi.org/10.1057/jos.2012.27).  
 21      443       URL <https://doi.org/10.1057/jos.2012.27>  
 22  
 23  
 24      444 [34] M. Viroli, R. Casadei, D. Pianini, Simulating large-scale aggregate MASs  
 25      445       with Alchemist and Scala, in: M. Ganzha, L. A. Maciaszek, M. Pa-  
 26      446       przycki (Eds.), Proceedings of the 2016 Federated Conference on Com-  
 27      447       puter Science and Information Systems, FedCSIS 2016, Gdańsk, Poland,  
 28      448       September 11–14, 2016, Vol. 8 of Annals of Computer Science and Infor-  
 29      449       mation Systems, IEEE, 2016, pp. 1495–1504. doi:[10.15439/2016F407](https://doi.org/10.15439/2016F407).  
 30      450       URL <https://doi.org/10.15439/2016F407>  
 31  
 32  
 33  
 34      451 [35] R. Casadei, M. Viroli, Programming actor-based collective adaptive  
 35      452       systems, in: A. Ricci, P. Haller (Eds.), Programming with Actors  
 36      453       - State-of-the-Art and Research Perspectives, Vol. 10789 of Lecture  
 37      454       Notes in Computer Science, Springer, 2018, pp. 94–122. doi:[10.1007/978-3-030-00302-9\\_4](https://doi.org/10.1007/978-3-030-00302-9_4).  
 38      455       URL [https://doi.org/10.1007/978-3-030-00302-9\\_4](https://doi.org/10.1007/978-3-030-00302-9_4)  
 39  
 40  
 41  
 42      457 [36] D. Pianini, R. Casadei, M. Viroli, A. Natali, Partitioned integration and  
 43      458       coordination via the self-organising coordination regions pattern, Future  
 44      459       Gener. Comput. Syst. 114 (2021) 44–68. doi:[10.1016/j.future.2020.07.032](https://doi.org/10.1016/j.future.2020.07.032).  
 45      460       URL <https://doi.org/10.1016/j.future.2020.07.032>  
 46  
 47  
 48  
 49      462 [37] G. Aguzzi, R. Casadei, M. Viroli, Towards reinforcement learning-based  
 50      463       aggregate computing, in: M. H. ter Beek, M. Sirjani (Eds.), Coordina-  
 51      464       tion Models and Languages - 24th IFIP WG 6.1 International Con-  
 52      465       ference, COORDINATION 2022, Held as Part of the 17th Interna-  
 53      466       tional Federated Conference on Distributed Computing Techniques, Dis-  
 54      467       CoTec 2022, Lucca, Italy, June 13–17, 2022, Proceedings, Vol. 13271

- 1  
 2  
 3  
 4  
 5       of Lecture Notes in Computer Science, Springer, 2022, pp. 72–91.  
 6       doi:10.1007/978-3-031-08143-9\\_5.  
 7       URL [https://doi.org/10.1007/978-3-031-08143-9\\_5](https://doi.org/10.1007/978-3-031-08143-9_5)  
 8  
 9  
 10      [38] R. Casadei, M. Viroli, Coordinating computation at the edge: a de-  
 11       centralized, self-organizing, spatial approach, in: Fourth International  
 12       Conference on Fog and Mobile Edge Computing, FMEC 2019, Rome,  
 13       Italy, June 10-13, 2019, IEEE, 2019, pp. 60–67. doi:10.1109/FMEC.  
 14       2019.8795355.  
 15       URL <https://doi.org/10.1109/FMEC.2019.8795355>  
 16  
 17  
 18      [39] R. Casadei, C. Tsigkanos, M. Viroli, S. Dustdar, Engineering resilient  
 19       collaborative edge-enabled iot, in: E. Bertino, C. K. Chang, P. Chen,  
 20       E. Damiani, M. Goul, K. Oyama (Eds.), 2019 IEEE International Con-  
 21       ference on Services Computing, SCC 2019, Milan, Italy, July 8-13, 2019,  
 22       IEEE, 2019, pp. 36–45. doi:10.1109/SCC.2019.00019.  
 23       URL <https://doi.org/10.1109/SCC.2019.00019>  
 24  
 25  
 26      [40] R. Casadei, A. Aldini, M. Viroli, Towards attack-resistant aggregate  
 27       computing using trust mechanisms, Sci. Comput. Program. 167 (2018)  
 28       114–137. doi:10.1016/j.scico.2018.07.006.  
 29       URL <https://doi.org/10.1016/j.scico.2018.07.006>  
 30  
 31  
 32      [41] R. Casadei, G. Aguzzi, M. Viroli, A programming approach to collective  
 33       autonomy, J. Sens. Actuator Networks 10 (2) (2021) 27. doi:10.3390/  
 34       jsan10020027.  
 35       URL <https://doi.org/10.3390/jsan10020027>  
 36  
 37  
 38      [42] R. Casadei, M. Viroli, A. Ricci, G. Audrito, Tuple-based coordination  
 39       in large-scale situated systems, in: F. Damiani, O. Dardha (Eds.), Co-  
 40       ordination Models and Languages - 23rd IFIP WG 6.1 International  
 41       Conference, COORDINATION 2021, Held as Part of the 16th Inter-  
 42       national Federated Conference on Distributed Computing Techniques,  
 43       DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings, Vol.  
 44       12717 of Lecture Notes in Computer Science, Springer, 2021, pp. 149–  
 45       167. doi:10.1007/978-3-030-78142-2\\_10.  
 46       URL [https://doi.org/10.1007/978-3-030-78142-2\\_10](https://doi.org/10.1007/978-3-030-78142-2_10)  
 47  
 48  
 49      [43] R. Casadei, D. Pianini, M. Viroli, D. Weyns, Digital twins, virtual de-  
 50       vices, and augmentations for self-organising cyber-physical collectives,  
 51       Applied Sciences 12 (1). doi:10.3390/app12010349.  
 52       URL <https://www.mdpi.com/2076-3417/12/1/349>  
 53  
 54  
 55  
 56  
 57  
 58  
 59  
 60  
 61  
 62  
 63  
 64  
 65

- 1  
2  
3  
4  
5 [44] R. Casadei, scafi/artifact-2021-ecoop-xc: v1.2 (2022). doi:10.5281/  
6 ZENODO.6538810.  
7  
8 URL <https://zenodo.org/record/6538810>
- 9  
10 [45] R. Casadei, scafi/artifact-2021-ecoop-smartc: v1.2 (2022). doi:10.  
11 5281/ZENODO.6538822.  
12  
13 URL <https://zenodo.org/record/6538822>
- 14  
15 [46] G. Aguzzi, D. Pianini, cric96/experiment-2022-ieee-decentralised-  
16 system: 1.0.1 (2022). doi:10.5281/ZENODO.6477039.  
17  
18 URL <https://zenodo.org/record/6477039>
- 19  
20 [47] A. Paulos, S. Dasgupta, J. Beal, Y. Mo, K. D. Hoang, L. J. Bryan, P. P.  
21 Pal, R. E. Schantz, J. Schewe, R. K. Sitaraman, A. Wald, C. Wayllace,  
22 W. Yeoh, A framework for self-adaptive dispersal of computing services,  
23 in: IEEE 4th International Workshops on Foundations and Applications  
24 of Self\* Systems, FAS\*W@SASO/ICCAC 2019, Umea, Sweden, June 16-  
25 20, 2019, IEEE, 2019, pp. 98–103. doi:10.1109/FAS-W.2019.00036.  
26  
27 URL <https://doi.org/10.1109/FAS-W.2019.00036>
- 28  
29 [48] J. Beal, K. Usbeck, J. P. Loyall, M. Rowe, J. M. Metzler, Adaptive  
30 opportunistic airborne sensor sharing, ACM Trans. Auton. Adapt. Syst.  
31 13 (1) (2018) 6:1–6:29. doi:10.1145/3179994.  
32  
33 URL <https://doi.org/10.1145/3179994>
- 34  
35 [49] R. Casadei, M. Viroli, G. Audrito, D. Pianini, F. Damiani, Aggregate  
36 processes in field calculus, in: H. R. Nielson, E. Tuosto (Eds.), Coordi-  
37 nation Models and Languages - 21st IFIP WG 6.1 International Con-  
38 ference, COORDINATION 2019, Held as Part of the 14th International  
39 Federated Conference on Distributed Computing Techniques, DisCoTec  
40 2019, Kongens Lyngby, Denmark, June 17-21, 2019, Proceedings, Vol.  
41 11533 of Lecture Notes in Computer Science, Springer, 2019, pp. 200–  
42 217. doi:10.1007/978-3-030-22397-7\\_12.  
43  
44 URL [https://doi.org/10.1007/978-3-030-22397-7\\_12](https://doi.org/10.1007/978-3-030-22397-7_12)
- 45  
46 [50] M. Odersky, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman,  
47 M. Zenger, et al., An overview of the scala programming language, Tech.  
48 rep. (2004).
- 49  
50 [51] B. G. Humm, R. S. Engelschall, Language-oriented programming via  
51 DSL stacking, in: J. A. M. Cordeiro, M. Virvou, B. Shishkov (Eds.),  
52 ICSOFT 2010 - Proceedings of the Fifth International Conference on  
53  
54  
55

- 1  
2  
3  
4  
5       539 Software and Data Technologies, Volume 2, Athens, Greece, July 22-24,  
6       540 2010, SciTePress, 2010, pp. 279–287.  
7
- 8       541 [52] D. Gelernter, Generative communication in linda, ACM Trans. Program.  
9       542 Lang. Syst. 7 (1) (1985) 80–112. doi:10.1145/2363.2433.  
10      543 URL <https://doi.org/10.1145/2363.2433>
- 11      544 [53] G. Audrito, R. Casadei, F. Damiani, V. Stolz, M. Viroli, Adaptive dis-  
12     545 tributed monitors of spatial properties for cyber-physical systems, J.  
13     546 Syst. Softw. 175 (2021) 110908. doi:10.1016/j.jss.2021.110908.  
14     547 URL <https://doi.org/10.1016/j.jss.2021.110908>
- 15      548 [54] S. Doeraene, Cross-platform language design in scala.js (keynote), in:  
16     549 S. Erdweg, B. C. d. S. Oliveira (Eds.), Proceedings of the 9th ACM  
17     550 SIGPLAN International Symposium on Scala, SCALA@ICFP 2018, St.  
18     551 Louis, MO, USA, September 28, 2018, ACM, 2018, p. 1. doi:10.1145/  
19     552 3241653.3266230.  
20     553 URL <https://doi.org/10.1145/3241653.3266230>
- 21      554 [55] G. Aguzzi, R. Casadei, N. Maltoni, D. Pianini, M. Viroli, Scafi-web:  
22     555 A web-based application for field-based coordination programming, in:  
23     556 F. Damiani, O. Dardha (Eds.), Coordination Models and Languages -  
24     557 23rd IFIP WG 6.1 International Conference, COORDINATION 2021,  
25     558 Held as Part of the 16th International Federated Conference on Dis-  
26     559 tributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June  
27     560 14-18, 2021, Proceedings, Vol. 12717 of Lecture Notes in Computer Sci-  
28     561 ence, Springer, 2021, pp. 285–299. doi:10.1007/978-3-030-78142-2\  
29     562 \_18.  
30     563 URL [https://doi.org/10.1007/978-3-030-78142-2\\_18](https://doi.org/10.1007/978-3-030-78142-2_18)
- 31      564 [56] L. Bettini, Implementing Domain-Specific Languages with Xtext and  
32     565 Xtend, Birmingham, ISBN: 9781786464965, Packt Publishing Ltd., UK,  
33     566 2016.
- 34      567 [57] G. E. Mobus, M. C. Kalton, Principles of Systems Science, Springer  
35     568 Publishing Company, Incorporated, 2014.
- 36      569 [58] F. Yates, Self-Organizing Systems: The Emergence of Order, Life Sci-  
37     570 ence Monographs, Springer US, 2012.  
38     571 URL <https://books.google.it/books?id=IiTvBwAAQBAJ>
- 39      572 [59] D. Miorandi, V. Maltese, M. Rovatsos, A. Nijholt, J. Stewart, Social  
40     573 Collective Intelligence: Combining the Powers of Humans and Machines

1  
2  
3  
4  
5       574 to Build a Smarter Society, Springer Publishing Company, Incorporated,  
6       575 2014.  
7

- 8       576 [60] S. Kalantari, E. Nazemi, B. Masoumi, Emergence phenomena in self-  
9       577 organizing systems: a systematic literature review of concepts, re-  
10      578 searches, and future prospects, *J. Organ. Comput. Electron. Commer.*  
11      579 30 (3) (2020) 224–265. doi:10.1080/10919392.2020.1748977.  
12      580 URL <https://doi.org/10.1080/10919392.2020.1748977>

**Declaration of interests**

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.

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: