MacroSwarm: A scala framework for swarm programming

MacroSwarm: a Scala Framework for Swarm Programming

Gianluca Aguzzi, Mirko Viroli

Alma Mater Studiorum - Università di Bologna, Italy

Abstract

Programming swarm behaviors is a challenging task, due to the need to express collective behaviors in terms of local interactions among simple agents. Even if several programming frameworks have been proposed, they are often based on low-level abstractions, which makes the development of swarm applications complex and error-prone. Thus, we present MacroSwarm, an aggregate programming framework for the development of swarm behaviors. With this framework, it is possible to define a large variety of swarm behaviors, starting from simple movements to more complex ones, such as aggregation, flocking, and collective decision-making. In this paper, we present the main features of the framework and some simple examples of its API usage.

Keywords: swarm programming, aggregate computing, collective adaptive systems

Metadata

1. Motivation and significance

The concepts of pervasive [1] and collective computing [2] foster a paradigm shift in which the physical world is starting to be populated with highly dynamic and interactive computational devices. These devices, through their interconnected nature, are not just mere independent entities; they are designed to collaborate seamlessly, striving towards collective goals. This vision is particularly significant in the context of large-scale systems, where the volume of devices necessitates an advanced form of organization. Here, the concept of "swarm" becomes central. Indeed, in such systems, the collective behavior emerges from the local interactions among the simple devices, mirroring the complexity and adaptability of social natural systems. Notable examples in this direction include swarm of drones [3], fleets of autonomous vehicles [4], and crowds of augmented humans [5].

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.5.3
C2	Permanent link to code/repository used for this code version	`https://zenodo.org/doi/10.5281/zenodo.10363375`
C3	Permanent link to Reproducible Capsule	`https://zenodo.org/doi/10.5281/zenodo.7829207`
C4	Legal Code License	Apache License 2.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Scala, ScaFi, Alchemist
C7	Compilation requirements, operating environments and dependencies	JDK 1.8+, SBT
C8	If available, link to developer documentation/manual	`https://scafi.github.io/macro-swarm/`
C9	Support email for questions	gianluca.aguzzi@unibo.it

Table 1: Code metadata (mandatory)

Numerous methods and programming languages (mostly originated from the context of *swarm intelligence* [6]) have been suggested for defining or programming swarm behavior, as seen in various studies and publications [7, 8, 9, 10, 11, 12, 13, 14, 15]. However, most of these lack a crucial aspect or provide it in a limited capacity: *compositionality*. This refers to the capability to merge simpler swarm behavior units to build more complex and controlled swarm systems.

Moreover, the majority of these existing methods are purely practical and lack a formal foundation. They facilitate the creation of specific swarm applications but offer inadequate support for the analysis and structured design of intricate applications, as indicated in some works (e.g., [12, 9, 14, 8]). Some exceptions offer a formal methodology, yet these are often too theoretical, necessitating extra efforts for the actual programming and implementation of swarm control applications [16].

Closing this gap is essential for the development of swarm systems , therefore in this paper we present MacroSwarm (already presented in [17]). It is different from the others since it introduces a *formally*-grounded API, *expressive* and *practical* enough to concisely and elegantly encode a wide array of swarm behaviors.

This framework is based on the API of the ScaFi language [18], which is a functional language for programming aggregate systems. ScaFi has been already used for programming swarm behaviours [19, 20, 21], but it lacked a

body of reusable components, finally given by MacroSwarm.

The behaviors exposed by the library do not depend on specific devices or communication protocols, but they are defined in terms of the local interactions among the agents, therefore they can be used in different contexts. Currently, MacroSwarm can be only tested in simulation (as the following examples), using the Alchemist simulator [18], but it is designed to be used in real-world applications. In the following sections, we will describe the main features of the framework, starting from the software architecture end then we discuss the key functionalities. Finally, we will present a case study to demonstrate the capabilities of MacroSwarm and discussing the related work in the field .

2. Software description

MacroSwarm aims to expose *essential* and *extensive* but *minimal* building blocks that capture the swarm's main patterns. These patterns can be combined aptly to create complex and high-level behaviors from simple ones, like obstacle avoidance and collective decision-making. The fundamental concept underlying MacroSwarm revolves around conceptualizing a unit of swarm behavior as *aggregate functions*, which are functions from computational fields to computational fields. *Computational fields* [22] are the main abstraction in aggregate computing, representing the spatial distribution of values in the aggregate system. These functions translate sensory and parameter inputs into actuation outputs, typically in the form of velocity vectors. In this way, the swarm behavior is defined as a composition of aggregate functions, which are then executed by the agents. Below, we describe the main features of the framework, beginning with the software architecture and then discussing the key functionalities.

2.1. Software architecture

The structure of the API in MacroSwarm is segmented into several *modules* (Figure 1), that are based on aggregate computing building blocks (red and blue rectangle in the figure). These modules encompass groups of behaviors that are logically connected. They include both basic, widely applicable behavior sets and those more tailored to specific applications, such as movement or team formation strategies. In order to use the API, the user must concretize a `MacroSwarmSupport` trait, which is the main entry point of the framework. Particularly, it needs a `Incarnation` instance, which is the main abstraction in ScaFi for representing an entire aggregate system (please refer to the reference paper for more details [18]). Currently, the only available incarnation is the `ScafiIncarnationForAlchemist`, which is used to run

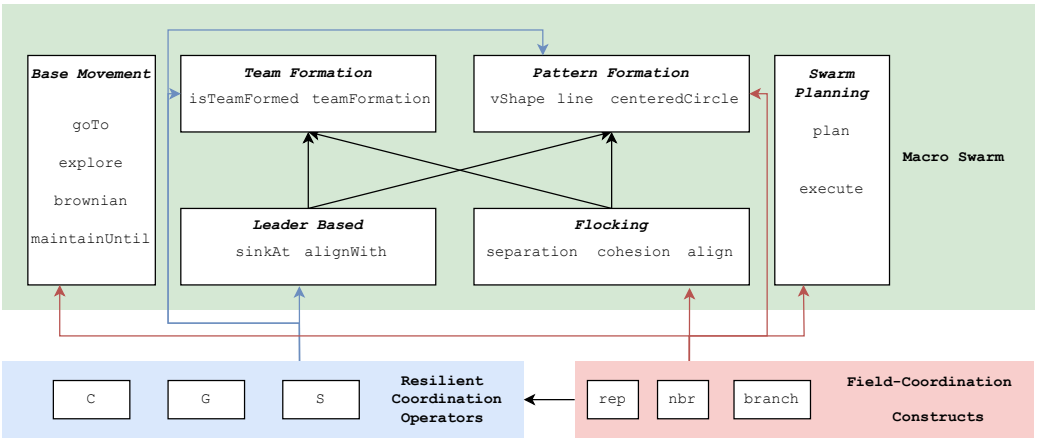


Figure 1: MacroSwarm architecture. The framework is composed of a set of modules, each one implementing a set of specific behaviors. The modules can be used together to create complex tasks.

simulations in the Alchemist simulator. More details about how to use the framework are given in Section 2.3.

2.2. Software functionalities

In the following, starting from Figure 1, we describe the functionalities of each module, highlighting the main behaviors that can be implemented using them.

Movement block. The module summarizes various movement-related blocks, offering functions for Brownian motion (i.e., random movement), GPS-based navigation, temporal movement control (i.e., repeating a movement for a certain duration/condition), and obstacle avoidance.

Flocking blocks. This module exposes a group of behaviors related to cohesive movement, collision avoidance between agents, and alignment in a common direction. It implements key blocks for agent flocking, drawing on models like Vicsek [23], Cucker-Smale [24], and Reynolds [25].

Leader-based blocks. In the domain of swarm systems, leadership plays a crucial role in coordination and influence. Leaders in these systems are usually identified through a Boolean computational field, where `true` indicates a *leader* agent and `false` signifies a *follower* agent. A leader essentially generates an *area of influence*, which significantly impacts the behavior of its followers.

Two critical components in this module are the `alignWithLeader` and `sinkAt` blocks. The `alignWithLeader` block propagates the leader's velocity

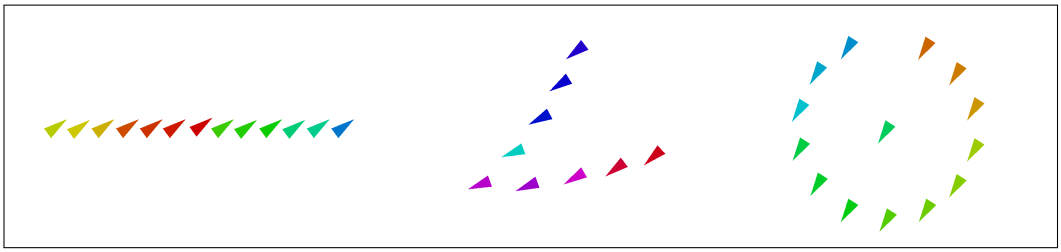


Figure 2: The formation of different shapes using the MacroSwarm API.

to its followers within its influence zone, thereby aligning their velocities accordingly. On the other hand, the `sinkAt` block creates a computational field that induces an attractive force toward the leader, ensuring cohesion within the group.

Team formation blocks. Team formation is a key feature that enables the division of a swarm into *sub-groups* or *teams* for specific tasks or interventions. This process involves creating distinct groups within the swarm, each with their objectives. Teams can be formed based on various criteria, such as spatial arrangements or leader-based structures. This module facilitates the formation of teams with the `teamFormation` API and can be conditioned on certain parameters like team size or proximity requirements. Leaders play a central role in coordinating these teams, employing various methods to direct and align the movements of team members.

Pattern formations blocks. The spatial structure of a swarm is a crucial aspect of swarm systems since it impacts the efficiency of the swarm's operations. In MacroSwarm, the process of forming these structures involves a leader agent who gathers distance data from follower agents and direct them to form the desired structure. Supported structures include v-like shapes, lines, and circular formations (see Figure 2), all characterized by their *self-healing* nature, enabling them to reassemble after disturbances.

Swarm Planning blocks. This section outlines a mechanism in MacroSwarm for expressing *dynamic plans* that guide swarm movements toward varying targets over time, referred to as "swarm planning", based on the notion of *aggregate plans* [26] . Plans are defined by a *behavior*, which is essentially the logic for producing a velocity vector, and a *goal*, determined by a boolean predicate condition. A swarm executes a sub-plan until the goal's condition is met, then moves to the next objective.

2.3. Sample code snippets

MacroSwarm is available on Maven Central and can be imported into a Scala project using the following dependency (SBT):

```
libraryDependencies +=
  "it.unibo.scafi" %% "macro-swarm-core" % "1.5.3"
libraryDependencies +=
  "it.unibo.scafi" %% "macro-swarm-alchemist" % "1.5.3"
```

This includes the core library and the Alchemist incarnation, which is currently the only available incarnation. The following demonstrates how to use the MacroSwarm API to program a simple swarm behavior. As a starting point, refer to this repository: `https://github.com/AggregateComputing/macro-swarm-demo`. To use MacroSwarm, a `MacroSwarmSupport` trait must be concretized.

The code snippet below shows how to import a `MacroSwarmSupport` instance for Alchemist:

```
// for macro swarm API
import it.unibo.scafi.macroswarm.MacroSwarmAlchemistSupport._
// for ScaFi API
import it.unibo.scafi.macroswarm.MacroSwarmAlchemistSupport.incarnation._
```

This imports all the functionalities of the framework into the namespace. To utilize them, an `MacroSwarmProgram` must be created, which is the main abstraction in ScaFi for representing an aggregate program.

From this, various modules of the framework can be mixed in, as shown in the following example:

```
// define a program that supports movement in an alchemist environment
class SimpleMovement extends MacroSwarmProgram
  with StandardSensors with TimeUtils // standard AC API
  with ScafiAlchemistSupport // support for alchemist
  // library for basic movement
  with BaseMovementLib {
    override protected def movementLogic(): Point3D = brownian()
}
```

Before running the program, a simulation environment must be defined, which in Alchemist is done by configuring a YAML file. In this file, the number of nodes, the communication radius, the simulation duration, and the initial positions of the nodes can be specified. For instance, the following YAML configuration expresses the setup for a simulation that launches the defined program:

```
## Simulator configuration
incarnation: scafi

launcher: { parameters: { batch: [], autoStart: false } }
# Attach the GUI to the simulator
monitors: { type: SwingGUI }

# Environment configuration
network-model: { type: ConnectWithinDistance, parameters: [350] }

_reactions:
  - program: &program # MacroSwarm program, executed every second
      - time-distribution: 1
        type: Event
        actions:
          - type: RunScafiProgram
            parameters: [example.SimpleMovement, 1.0]
      - program: send
  - move: &move # Related actuation, executed every second
      - time-distribution: 1
        type: Event
        actions: { type: MoveToTarget, parameters: [ destination, 10 ] }

deployments:
  type: Grid
  parameters: [-500, -500, 500, 500, 300, 300, 25, 25]
  programs: [ *program, *move ]
```

The configuration is divided into two parts: one for configuring the simulator and one for defining the environment. In the simulator configuration, the specific incarnation to use (i.e., `scafi`) and the type of launcher (i.e., `SingleRunSwingUI`) are specified.

In the environment configuration, the network model, the reactions, and the deployment of the program are defined. Specifically, nodes in the simulation are connected based on a network model that links nodes within a distance of 350 units (expressed with `network-model`). The behavior of the nodes is defined through reactions. One such reaction, labeled `program`, is a MacroSwarm program that runs every second. It includes the `SimpleMovement` MacroSwarm program with a parameter. Another reaction, named `MoveToTarget`, also runs every second and includes actions like moving to a target destination with a speed parameter of 10.

The deployment of the program is specified in a grid format. The grid starts from coordinates $(-500, -500)$ and ends at $(500, 500)$ with steps of 300 units in both the x and y directions. The positions of the nodes in the grid have a randomness variance of 25 units. The program is deployed in a 4×4 grid with the behaviors defined in the `program` and `move` reactions.

For a full YAML specification, refer to the documentation of the Alchemist simulator[1]. With this configuration, the simulation can be run using SBT by adding the following lines to the `build.sbt` file:

```
Compile / mainClass := Some("it.unibo.alchemist.Alchemist")
run / mainClass := Some("it.unibo.alchemist.Alchemist")
addCommandAlias("runAlchemist", "run run src/main/yaml/main.yml")
```

Where `src/main/yaml/main.yml` is the path to the YAML file. The simulation can be initiated with the command `sbt runAlchemist`. Once the GUI is open, start the simulation by clicking the play button. The nodes should be observed moving randomly in the environment. This is a brief overview; for more details, refer to the documentation [2].

3. Illustrative example

To demonstrate the capabilities of MacroSwarm, we will now present a case study previously discussed in [17]. Consider the following scenario: a swarm of robots is deployed in a disaster area to perform a search and rescue mission. In our case, we aim to deploy a group of drones to monitor a specific region. Within this zone, various emergencies, like fires or injuries, might occur. To address these incidents, a specialized drone, termed a *healer*, is tasked to handle them (the specifics of resolution are not covered in this study). For effective surveillance, the drones should operate in teams consisting of at least one healer and multiple *explorers*. These explorers will assist the healer in detecting critical situations.

To organize the desired behavior of the drone swarm, we divide the task into several steps:

1. Formation of teams guided by a healer drone, acting as a *leader*. Each team must have at least one leader (Figure 3a).
2. Teams should arrange themselves in a spatial pattern that enhances the efficiency of their search operations (Figure 3b),
3. The teams are tasked with covering and exploring the entire designated area (Figure 3c).
4. Upon detecting a hazardous zone, the detecting drone must immediately inform the healer drone.
5. The healer drone then proceeds to the identified danger zone to address the situation.

[1] `http://alchemistsimulator.github.io/`

[2] `https://scafi.github.io/macro-swarm/guide/quick.html`

6. Following the intervention, the team resumes its exploration activities.

Starting from this scenario, we outline the main steps for programming the swarm behavior using MacroSwarm. The process begins with forming teams using the **Team Formation** functions:

```
val teamFormedLogic =
  (leader: ID) => isTeamFormed(leader, minimumDistance + confidence)
def createTeam() =
  teamFormation(sense("healer"), minimumDistance, teamFormedLogic)
```

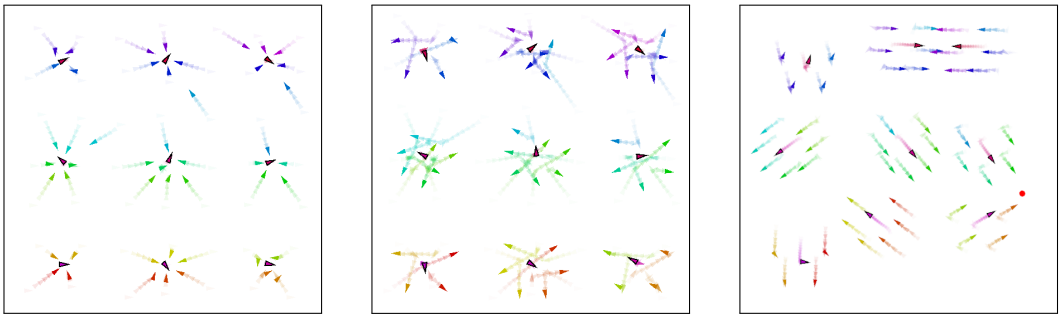
Here, `minimumDistance` represents the least distance to be maintained between drones during team formation, and `confidence` is the interval used to confirm if a team is properly formed via the `isTeamFormed` method.

Each team then carries out the steps described earlier, which can be programmed using the **Swarm Planning** API:

```
def insideTeamPlanning(team: Team): Point3D =
  team.insideTeam {
    healerId => {
      val leading = healerId == mid() // identifies the team leader
      execute.repeat(
        plan(formation(leading)).endWhen(circleIsFormed),
        plan(wanderInFormation(leading)).endWhen(dangerFound),
        plan(goToHealInFormation(leading, inDanger))
          .endWhen(dangerReached),
        plan(heal(healerId, inDanger)).endWhen(healed(dangerFound))
      ).run() // repeating the plan
    }
  }
```

To validate the code described above, we then deploy the script in a simulated environment, using the Alchemist simulator [27][3]. At the beginning, an array of 50 explorers and 5 healers are dispersed randomly over a $1\,\mathrm{km}^2$ region. Each drone is capable of reaching speeds up to 20 km/h and can communicate within a radius of 100 meters. Emergency scenarios are sporadically created throughout the designated area, occurring at varied intervals within a $[0, 50]$ minute timeframe. Teams are to be organized such that they include at least one healer and multiple explorers, ensuring a minimum separation of 50 meters between each node and its team leader. In Figure 3 is shown some snapshots from the simulations, but for more details please refer to [17].

[3]simulation at: `https://zenodo.org/doi/10.5281/zenodo.7829207`



(a) Team formation (b) Circle formation (c) Explore

Figure 3: The initial stages of the situation outlined in Section 3 involve several key steps. Initially, the system is organized into various teams (Figure 3a). Subsequently, these teams arrange themselves in a specific spatial pattern, which in this instance is circular (Figure 3b). The final step involves the teams commencing their exploration of the broader area /Figure 3c. To reproduce the snapshots please look at `https://scafi.github.io/macro-swarm/guide/alchemist.html`

4. Impact

Aggregate computing has been widely used in research for swarm-like scenarios [20, 21, 28, 29]. Therefore, creating a library of blocks that encapsulate the main behaviors offers the possibility to simplify this development process and at the same time allow it to be used easily even by those who are not properly versed in aggregate programming. Below, other details about the impact of the software are provided.

4.1. Close the reality gap of aggregate computing

MacroSwarm significantly contributes to bridging the gap between the theoretical models of aggregate computing and their practical applications, particularly in swarm-like scenarios. By offering a collection of behavior blocks that encapsulate the core aspects of swarm behaviors, MacroSwarm enables the implementation of complex swarm systems in a more accessible and manageable manner. This approach not only simplifies the development process but also extends the use of aggregate computing to a broader range of users, including those who may not have specialized knowledge in this area. The adaptability and practicality of MacroSwarm pave the way for its application in diverse fields, ranging from environmental monitoring to disaster response, thereby opening new avenues for research and innovation in swarm robotics and aggregate computing.

4.2. Flexible and composable swarm programming

The design of MacroSwarm emphasizes flexibility and composability in programming swarm behaviors. This framework allows for the assembly of

complex behaviors from simpler, modular components. Such an approach provides developers with the versatility to tailor swarm behaviors to specific tasks and scenarios, enhancing the effectiveness and efficiency of swarm systems in various applications. MacroSwarm's composable architecture thus represents a significant advancement in the field of swarm programming, enabling more robust, adaptable, and efficient swarm systems.

5. Related Work

In this section, we review related work on swarm programming languages and DSLs for decentralized swarm programming (Section 5.1), and task orchestration languages (Section 5.2).

5.1. Decentralized Swarm Programming

Buzz [14] is a mixed imperative-functional language for swarms, utilizing *virtual stigmergy* for swarm-wide consensus. MacroSwarm employs general, expressive primitives and supports swarm programming with reusable blocks, leveraging field calculi for formal analysis [30].

Voltron [13] uses a team abstraction for drone systems, focusing on task commands with spatiotemporal constraints. Meld [7], a logic-based language for modular ensembles, employs facts with side-effects, production rules, and aggregate rules, but primarily addresses shape formation and self-reconfiguring ensembles.

Other related calculi and languages (e.g., attribute-based communication languages [31]) are noted for potential but lack comprehensive implementations or libraries.

5.2. Centralized Orchestration Approaches

Task orchestration languages (e.g., TeCoLa [11], Dolphin [12], Maple-Swarm [10], PARoS [9], Resh [8]) use centralized entities for swarm control based on task descriptions. MacroSwarm provides decentralized control through a collective task description executed by individual agents.

TeCoLa, implemented in Python, coordinates robotic teams with a mission group concept managed by a coordinator. PARoS, a Java framework, uses a central coordinator for swarm orchestration and spatial organization. Maple-Swarm supports swarm capabilities through agent groups and hierarchical task plans. Dolphin, a Groovy DSL, defines vehicle sets for autonomous vehicle networks, supporting orchestration of individual behaviors. [15] proposes a mixed decentralised-centralised actor-based framework for swarm programming. Resh, a DSL for multi-robot orchestration, uses

tasks, robot capabilities, and spatiotemporal primitives for non-Turing complete task synthesis.

In summary, while many approaches focus on centralized control or specific applications, MacroSwarm emphasizes reusable, composable blocks for decentralized swarm programming.

6. Conclusions

In this paper, we introduced MacroSwarm, a framework designed for high-level programming of drone swarms. This framework offers a set of building blocks that represent common behaviors in decentralized swarms. It is based on aggregate computing, a well-established method for managing group behaviors, and is developed using the ScaFi toolkit/DSL. Through various examples and a simulated case study, we demonstrated that our approach is flexible, practical, and powerful.

7. Future Plans

Looking ahead, we aim to expand the MacroSwarm API to include a broader range of swarm behaviors, as outlined in well-known classifications of swarm activities [3]. Moreover, we plan to extend the framework to support other incarnations, like ARGoS (for simulations) and ROS (for real-case scenarios). We are also interested in exploring methods to automatically generate combinations of MacroSwarm blocks. One possibility is to follow a modern trend called "hybrid aggregate computing", in which aggregate programming is aptly mixed with sub-symbolic artificial intelligence techniques (e.g., machine learning) [32, 33, 34, 35, 28]. Finally, our goal is to implement and evaluate MacroSwarm in real-world scenarios.

Acknowledgements

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

References

[1] D. Saha, A. Mukherjee, S. Bandyopadhyay, D. Saha, A. Mukherjee, S. Bandyopadhyay, Pervasive computing, Networking Infrastructure for Pervasive Computing: Enabling Technologies and Systems (2003) 1–37.

[2] G. D. Abowd, Beyond weiser: From ubiquitous to collective computing, Computer 49 (1) (2016) 17–23.

[3] M. Schranz, M. Umlauft, M. Sende, W. Elmenreich, Swarm robotic behaviors and current applications, Frontiers Robotics AI 7 (2020) 36.
URL https://doi.org/10.3389/frobt.2020.00036

[4] A. Tahir, J. Böling, M. H. Haghbayan, H. T. Toivonen, J. Plosila, Swarms of unmanned aerial vehicles - A survey, J. Ind. Inf. Integr. 16 (2019) 100106.
URL https://doi.org/10.1016/j.jii.2019.100106

[5] O. Galinina, K. Mikhaylov, K. Huang, S. Andreev, Y. Koucheryavy, Wirelessly powered urban crowd sensing over wearables: Trading energy for data, IEEE Wirel. Commun. 25 (2) (2018) 140–149.
URL https://doi.org/10.1109/MWC.2018.1600468

[6] M. J. Mataric, Designing emergent behaviors: From local interactions to collective intelligence, in: Proceedings of the Second International Conference on Simulation of Adaptive Behavior, 1992, pp. 432–441.

[7] M. P. Ashley-Rollman, S. C. Goldstein, P. Lee, T. C. Mowry, P. Pillai, Meld: A declarative approach to programming ensembles, in: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2007, pp. 2794–2800.
URL https://doi.org/10.1109/IROS.2007.4399480

[8] M. Carroll, K. S. Namjoshi, I. Segall, The Resh programming language for multirobot orchestration, in: IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021, IEEE, 2021, pp. 4026–4032.
URL https://doi.org/10.1109/ICRA48506.2021.9561133

[9] D. Dedousis, V. Kalogeraki, A framework for programming a swarm of UAVs, in: 11th PErvasive Technologies Related to Assistive Environments Conference (PETRA'18), Proceedings, ACM, 2018, pp. 5–12.
URL https://doi.org/10.1145/3197768.3197772

[10] O. Kosak, L. Huhn, F. Bohn, C. Wanninger, A. Hoffmann, W. Reif, Maple-swarm: Programming collective behavior for ensembles by extending HTN-planning, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30,

2020, Proceedings, Part II, Vol. 12477 of Lecture Notes in Computer Science, Springer, 2020, pp. 507–524.
URL https://doi.org/10.1007/978-3-030-61470-6_30

[11] M. Koutsoubelias, S. Lalis, Tecola: A programming framework for dynamic and heterogeneous robotic teams, in: Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2016), ACM, 2016, pp. 115–124.
URL https://doi.org/10.1145/2994374.2994397

[12] K. Lima, E. R. B. Marques, J. Pinto, J. B. Sousa, Dolphin: A task orchestration language for autonomous vehicle networks, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018, IEEE, 2018, pp. 603–610.
URL https://doi.org/10.1109/IROS.2018.8594059

[13] L. Mottola, M. Moretta, K. Whitehouse, C. Ghezzi, Team-level programming of drone sensor networks, in: Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems (SenSys'14), ACM, 2014, pp. 177–190.
URL https://doi.org/10.1145/2668332.2668353

[14] C. Pinciroli, G. Beltrame, Buzz: An extensible programming language for heterogeneous swarm robotics, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016, Daejeon, South Korea, October 9-14, 2016, IEEE, 2016, pp. 3794–3800.
URL https://doi.org/10.1109/IROS.2016.7759558

[15] W. Yi, B. Di, R. Li, H. Dai, X. Yi, Y. Wang, X. Yang, An actor-based programming framework for swarm robotic systems, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021, IEEE, 2020, pp. 8012–8019.
URL https://doi.org/10.1109/IROS45743.2020.9341198

[16] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, M. Fisher, Formal specification and verification of autonomous robotic systems: A survey, ACM Comput. Surv. 52 (5) (2019) 100:1–100:41.
URL https://doi.org/10.1145/3342355

[17] G. Aguzzi, R. Casadei, M. Viroli, Macroswarm: A field-based compositional framework for swarm programming, in: International Conference on Coordination Languages and Models, Springer, 2023, pp. 31–51.

[18] R. Casadei, M. Viroli, G. Aguzzi, D. Pianini, ScaFi: A Scala DSL and toolkit for aggregate programming, SoftwareX 20 (2022) 101248.
URL https://doi.org/10.1016/j.softx.2022.101248

[19] 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

[20] G. Aguzzi, R. Casadei, D. Pianini, M. Viroli, Dynamic decentralization domains for the internet of things, IEEE Internet Comput. 26 (6) (2022) 16–23. doi:10.1109/MIC.2022.3216753.
URL https://doi.org/10.1109/MIC.2022.3216753

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

[22] M. Viroli, F. Damiani, J. Beal, A calculus of computational fields, in: C. Canal, M. Villari (Eds.), Advances in Service-Oriented and Cloud Computing - Workshops of ESOCC 2013, Málaga, Spain, September 11-13, 2013, Revised Selected Papers, Vol. 393 of Communications in Computer and Information Science, Springer, 2013, pp. 114–128. doi:10.1007/978-3-642-45364-9_11.
URL https://doi.org/10.1007/978-3-642-45364-9_11

[23] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, O. Shochet, Novel type of phase transition in a system of self-driven particles, Phys. Rev. Lett. 75 (1995) 1226–1229.
URL https://link.aps.org/doi/10.1103/PhysRevLett.75.1226

[24] F. Cucker, S. Smale, Emergent behavior in flocks, IEEE Transactions on Automatic Control 52 (5) (2007) 852–862. doi:10.1109/TAC.2007.895842.

[25] C. W. Reynolds, Flocks, herds and schools: A distributed behavioral model, in: M. C. Stone (Ed.), Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987, Anaheim, California, USA, July 27-31, 1987, ACM, 1987, pp. 25–34.
URL https://doi.org/10.1145/37401.37406

[26] M. Viroli, D. Pianini, A. Ricci, A. Croatti, Aggregate plans for multiagent systems, Int. J. Agent Oriented Softw. Eng. 5 (4) (2017) 336–365. doi:10.1504/IJAOSE.2017.10008554.
URL https://doi.org/10.1504/IJAOSE.2017.10008554

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

[28] G. Aguzzi, M. Viroli, L. Esterle, Field-informed reinforcement learning of collective tasks with graph neural networks, in: 2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), IEEE Computer Society, 2023, pp. 37–46.

[29] G. Aguzzi, G. Audrito, R. Casadei, F. Damiani, G. Torta, M. Viroli, A field-based computing approach to sensing-driven clustering in robot swarms, Swarm Intell. 17 (1-2) (2023) 27–62. doi:10.1007/S11721-022-00215-Y.
URL https://doi.org/10.1007/s11721-022-00215-y

[30] M. Viroli, J. Beal, F. Damiani, G. Audrito, R. Casadei, D. Pianini, From distributed coordination to field calculus and aggregate computing, J. Log. Algebraic Methods Program. 109.

[31] R. De Nicola, M. Loreti, R. Pugliese, F. Tiezzi, A formal approach to autonomic systems programming: The SCEL language, ACM Trans. Auton. Adapt. Syst. 9 (2) (2014) 7:1–7:29.

[32] G. Aguzzi, R. Casadei, M. Viroli, Towards reinforcement learning-based aggregate computing, in: M. H. ter Beek, M. Sirjani (Eds.), Coordination Models and Languages - 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings, Vol. 13271 of Lecture Notes in Computer Science, Springer, 2022, pp. 72–91.
URL https://doi.org/10.1007/978-3-031-08143-9_5

[33] D. Domini, F. Cavallari, G. Aguzzi, M. Viroli, Scarlib: A framework for cooperative many agent deep reinforcement learning in scala, in: S. Jongmans, A. Lopes (Eds.), Coordination Models and Languages - 25th IFIP WG 6.1 International Conference, COORDINATION 2023,

Held as Part of the 18th International Federated Conference on Distributed Computing Techniques, DisCoTec 2023, Lisbon, Portugal, June 19-23, 2023, Proceedings, Vol. 13908 of Lecture Notes in Computer Science, Springer, 2023, pp. 52–70. doi:10.1007/978-3-031-35361-1_3.
URL https://doi.org/10.1007/978-3-031-35361-1_3

[34] G. Aguzzi, R. Casadei, M. Viroli, Addressing collective computations efficiency: Towards a platform-level reinforcement learning approach, in: R. Casadei, E. D. Nitto, I. Gerostathopoulos, D. Pianini, I. Dusparic, T. Wood, P. R. Nelson, E. Pournaras, N. Bencomo, S. Götz, C. Krupitzer, C. Raibulet (Eds.), IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2022, Virtual, CA, USA, September 19-23, 2022, IEEE, 2022, pp. 11–20. doi:10.1109/ACSOS55765.2022.00019.
URL https://doi.org/10.1109/ACSOS55765.2022.00019

[35] G. Aguzzi, R. Casadei, M. Viroli, Machine learning for aggregate computing: a research roadmap, in: 42nd IEEE International Conference on Distributed Computing Systems, ICDCS Workshops, Bologna, Italy, July 10, 2022, IEEE, 2022, pp. 119–124. doi:10.1109/ICDCSW56584.2022.00032.
URL https://doi.org/10.1109/ICDCSW56584.2022.00032