Modelling Flocks of Birds from the Bottom Up*

Rocco De Nicola[1], Luca Di Stefano[2,3], Omar Inverso[4], and Serenella Valiani[1](✉)

[1] IMT School of Advanced Studies, Lucca, Italy
[2] Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, Grenoble, France
[3] University of Gothenburg, Gothenburg, Sweden
[4] Gran Sasso Science Institute (GSSI), L'Aquila, Italy
serenella.valiani@imtlucca.it

Abstract. We argue that compositional specification based on formal languages can facilitate the modelling of, and reasoning about, sophisticated collective behaviour in many natural systems. One defines a system in terms of individual components and local rules, so that collective behaviours emerge naturally from the combined effect of the different actions of the individual components. With appropriate linguistic constructs, this can yield compact and intuitive models that are easy to refine and extend in small incremental steps. In addition, automated workflows implemented on top of this methodology can provide quick feedback, thereby allowing rapid design iterations. To support our argument, we consider flocking, a well-known example of emergent behaviour in collective adaptive systems. We build a minimalistic bottom-up model of a flock of birds incrementally, discussing specific language constructs as we go along. We then describe a prototype simulator, and use it to validate our model in a controlled experiment, where a flock is attacked by a bird of prey. The flock effectively reacts to the attack by splitting into smaller groups and regathering once the threat subsides, consistently with both natural observations and previous models from the literature.

1 Introduction

The organization of complex systems in nature, such as flocks of birds, colonies of ants, schools of fish, swarms of insects, and many more, has long since been attracting considerable interest. Researchers with different background have been resorting to different mathematical frameworks in order to study these phenomena. For instance, *flocking*, where a group of birds exhibits coherent patterns of collective motion, has been modelled using graph theory [24], distributed control laws [32], and statistical mechanics [3].

This way of modelling is not always practical because it relies on general-purpose formalisms that may not be very intuitive to use, in addition to the fact that the system needs to be modelled as a whole, regardless from its natural

* Work partially funded by MIUR project PRIN 2017FTXR7S IT MATTERS (Methods and Tools for Trustworthy Smart Systems).

structure and often by artificially introducing some kind of central control. In contrast, in different disciplines, including epidemiology, ecology, economics, and social sciences [19,14,33,5], there seems to be a growing interest towards *compositional* approaches where the model focusses on the individual components rather than on the whole system.

Along these lines, in this paper we advocate a *bottom-up* approach based on formal specification languages. One defines the system of interest in terms of individual components and local rules. The collective behaviour of the system as a whole is not specified explicitly, but can be observed to emerge from the combined effect of the different actions of the components. This can be of significant help to reproduce sophisticated collective dynamics intuitively, and, when combined with appropriate linguistic constructs, can yield compact and intuitive specifications that are easy to refine. The adoption of a formal language allows implementing automated workflows for simulation or formal analysis that can provide feedback quickly, thereby allowing rapid design iterations.

To illustrate our point, we develop a model of a flock by gradually defining the individual behaviour and features of a bird. As we progressively refine it, we aim at keeping the behaviour of individual birds as decentralized as possible. We write our increasingly complex models in an existing language [7], which we gradually extend with new constructs that keep the specifications compact and intuitive. Once the model is fully refined, we simulate the evolution of a flock obtained by composing a number of birds together, and show that it displays interesting collective features. Namely, when birds are attacked by an external bird of prey, they are able to first escape from it, and then reassemble into a coherent flock when they are no longer under threat. This kind of collective behaviour reflects the one emerging from other models in the literature, but relies on a rather simple model.

The rest of this paper is structured as follows. We define our model of flocking behaviour and discuss tailored linguistic constructs for the specification language in Section 2. We describe our experimental setup for simulation and our controlled experiment in Section 3. We discuss related work in Section 4. Lastly, in Section 5 we report some final remarks and discuss potential directions for future work.

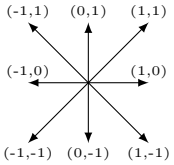
2 Specification

In this section, we develop a simplified model that resembles the dynamics of a flock. We start from describing a set of very simple birds, and then show how this description can be extended to implement our desired dynamics. As we do so, we also extend the modelling language itself with new constructs, aiming to keep the specifications succinct and intuitive.

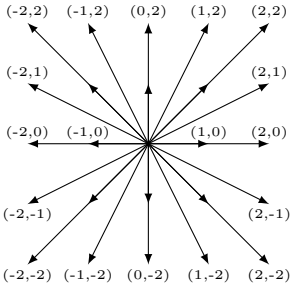
Description of a bird. Each bird in the flock can be described by two properties, namely its *position* and its *orientation* of movement. We model the former through a pair of coordinates (x, y) and the latter as a pair of integers $(\mathrm{dirx}, \mathrm{diry})$ representing a heading vector. This description allows to represent both the di-

Listing 1: Baseline agent modelling.

```
1  agent Bird {
2      Interface =
3          x: 0..G;
4          y: 0..G;
5          dirx: −D..D + 1;
6          diry: −D..D + 1
7
8      Behaviour = Move; Behaviour
9      Move = {
10         x ← x + dirx;
11         y ← y + diry
12     }
13 }
```



(a) $D = 1$.



(b) $D = 2$ (some labels omitted for readability).

Fig. 1: Possible heading vectors that a bird can assume for different values of D.

rection of the bird's displacement, i.e., the angle subtended by the heading vector, and the bird's velocity, represented by the length of the heading vector.

Listing 1 shows how we can model the above description.[1] In the first subsection (lines 2–6) we define the *interface* of the agent, where we define and initialize its observable features, or *attributes*. Attributes x and y are initialized non-deterministically and can assume any value corresponding to a valid coordinate on a grid that represents the arena where the flock is located. The grid is a square with edges of length G, thus the possible values vary from 0 to $G - 1$ included (lines 3–4). The initial values of dirx and diry range over $[-D, D]$ (lines 5–6). We use D to denote the maximum displacement along each coordinate of the grid: note that, as D increases, so does the number of possible heading vectors, as shown in Figure 1. The actual initial value of each attribute is chosen non-deterministically.

[1] In this paper, we present condensed, human-readable versions of the full, machine-readable specifications. These are available at `https://github.com/labs-lang/labs-examples/tree/isola2022/isola2022`.

Listing 2: Alignment.

```
1  agent Bird
2      Interface = ...
3
4      Behaviour = Move; Behaviour
5      Move = {
6          p := pick 1;
7          dirx ← dirx_p;
8          diry ← diry_p;
9          x ← x + dirx;
10         y ← y + diry
11     }
12 }
```

Behaviour of a bird. As for the *behaviour* of birds, let us initially model a system in which each bird simply moves along its heading vector set in the initial state, without ever changing it.

Listing 1 shows how to model such a behaviour. It is expressed through the recursive definition at line 8 that states that each agent repeatedly carries out the actions described in the **Move** process. More in detail, the statement x ← x+dirx at line 10 updates the attribute x, which represent a component of the position of the agent, with the evaluation of the expression x + dirx, i.e., the new position that the agent reaches by moving along its heading vector. Attribute y is updated similarly (line 11). Currently, we assume that agents never reach the edge. Please note that each assignment is executed atomically, but sequences of assignments may be subject to interleaving. To prevent interleaving between the assignments of the different agents, i.e. to execute multiple assignments atomically, these must be enclosed in curly brackets, as shown in lines 9–12.

Alignment. The specification introduced above does not lead to any kind of collective behaviour, as birds simply ignore each other and keep moving in their own, fixed directions. Therefore, we now have to specify birds that are somehow influenced by other flockmates. Indeed, it is commonly held that flocking behaviour is a result of a combination of local interaction mechanisms [13,29]. We start by considering *alignment*, i.e., the property whereby each bird adjusts its own direction according to that of its neighbours. A trivial method for achieving this is to let each bird imitate the heading of another bird in the flock. To model this behaviour, each bird must then be able to "watch" other birds and observe their heading.

Listing 2, lines 6–8 show the changes needed to implement the behaviour described above. We omit the interface for clarity, as it is the same as that of Listing 1. Before proceeding, we must stress that, although agents are anonymous to each other, they do have a concept of *identity*. This is provided internally by an *identifier* (id) that is unique to every agent in the system, performing a function similar to that of the keywords *this* or *self* in many general-purpose programming

Listing 3: Cohesion.

```
1  agent Bird {
2      Interface = ...
3
4      Behaviour = Move; Behaviour
5      Move = {
6          p ≔ pick 1;
7
8          a_x ≔ x_p + ω · dirx_p;
9          a_y ≔ y_p + ω · diry_p;
10         sgn_x ≔ 0 if x = a_x else (−1 if x > a_x else 1);
11         diff_x ≔ d((x, 0), (a_x, 0);
12         ... (Same for sgn_y, diff_y)
13         a_dirx ≔ sgn_x · (D:2 if diff_y > diff_x else D);
14         a_diry ≔ sgn_y · (D:2 if diff_y < diff_x else D);
15
16         dirx ← (dirx + a_dirx) : 2;
17         x ← x + dirx
18         ... (Same for diry, y)
19     }
20 }
```

languages. The fact that agents have identifiers allows us to introduce a new operator, by which an agent can non-deterministically select other agents in the system: namely, at line 6, the instruction p ≔ **pick** 1 selects the id of another agent and stores it into a *local* variable p. In general, **pick** k returns k distinct identifiers that are guaranteed to be different from that of the agent doing the selection. We use the operator ≔ to denote assignments to local variables; these are implicitly declared upon their first assignment.

Now that the bird has the identifier of an agent stored in p, it can read its heading vector by using the syntax $\mathrm{dirx_p}, \mathrm{diry_p}$. In this specification, the bird simply replaces its own heading vector by that of p (lines 7–8), and then moves by updating its own position (lines 9–10).

Cohesion. It is evident that birds in a real flock do not simply tend to move along the same direction, but also get close to each other and try to remain cohesive. However, the model of birds seen so far is not refined enough to display this kind of behaviour. In fact, two birds in distant positions will at best assume a coherent direction of movement, but this will not bring them closer to each other. Therefore, we now modify the behaviour described above in order to obtain both alignment and cohesion of the flock. Each bird first selects another bird of the flock; then, observing its direction, estimates the position where the selected bird will be in the future, and steers towards that position.

Listing 3 shows how to model this behaviour. Notice that, from now on, we use a **if** c **else** b to denote the ternary operator that evaluates to a when condition c holds and to b otherwise; the syntax $a : b$ denotes integer division

Listing 4: Flock dispersion and birds collision.

```
1  agent Bird {
2      Interface = ...
3
4      Behaviour = Move; Behaviour
5      Move = {
6          p ≔ pick 1;
7          pIsIsolated ≔ forall Bird b, (b = p) or d((x_p, y_p), (x_b, y_b)) > δ;
8          appId ≔ id if pIsIsolated else p;
9
10         a_x ≔ x_appId + ω · dirx_appId;
11         sgn_x ≔ 0 if x = a_x else (−1 if x > a_x, else 1);
12         diff_x ≔ d((x, 0), (a_x, 0);
13         ... (Same for a_y, sgn_y, diff_y)
14         a_dirx ≔ sgn_x · (D:2 if diff_y > diff_x else D);
15         a_diry ≔ sgn_y · (D:2 if diff_y < diff_x else D);
16
17         dirx ← (dirx + a_dirx) : 2;
18         diry ← (diry + a_diry) : 2;
19         posFree ≔ forall Bird b, (x_b ≠ x + dirx) or (y_b ≠ y + diry);
20         x ← x + dirx if posFree else x
21         y ← y + diry if posFree else y
22     }
23 }
```

with rounding; and $d((x_1, y_1), (x_2, y_2))$ denotes the Manhattan distance between two points, i.e., $|x_1 - x_2| + |y_1 - y_2|$. After picking the bird p to be approached (line 6), we estimate its position after ω steps (lines 8–9). Then, we determine an approach vector (a_dirx, a_diry) pointing towards that position. We compute this vector component-wise at lines 10–14: we omit the instructions for the y-component for sake of brevity. Lastly, we compute the bird's new heading vector as the average of its current one and the approach vector (line 16). This gives the bird a bit of inertia for a more realistic movement.

Avoiding flock dispersion and collisions. The specifications outlined so far may still cause undesired outcomes. For instance, the flock may disperse instead of compacting: this may occur when birds decide to approach other birds that are separated from the rest of the flock. Additionally, we may end up with collisions, i.e., two or more birds sharing the same grid location. To avoid the former, we need to provide birds with the capability of checking whether a bird is isolated. Similarly, to avoid collision, the bird has to check whether a location is free before moving.

In Listing 4, we refine our specifications as described above. At line 7, we check whether bird p is isolated, i.e., its distance from all other birds is greater than a parameter δ. To perform this check, quantified predicates are introduced, allowing to predicate over the attributes of all agents, or some agent, of given

Listing 5: Fleeing from a predator.

```
1  agent Predator { ... }
2
3  agent Bird {
4      Interface = ...
5
6      Behaviour = Move; Behaviour
7      Move = {
8          p ≔ pick 1 Bird;
9          ...
10         a_diry ≔ sgn_y · (D:2 if diff_y < diff_x else D);
11
12         e ≔ pick 1 Predator;
13         e_x ≔ x_e + ν · dirx_e;
14         esgn_x ≔ 1 if x ≥ e_x else −1;
15         ediff_x ≔ d((x, 0), (e_x, 0));
16         ... (Same for e_y, esgn_y, ediff_y)
17         e_dirx ≔ esgn_x · (D:2 if ediff_y > ediff_x else D);
18         e_diry ≔ esgn_y · (D:2 if ediff_y < ediff_x else D);
19
20         e_dist ≔ d((x, y), (e_x, e_y));
21         f_dirx ≔ e_dirx if e_dist < λ else a_dirx;
22         dirx ← (dirx + f_dirx) : 2;
23         ... (Same for f_diry, diry)
24         posFree ≔ forall Bird b, (x_b ≠ x + dirx) or (y_b ≠ y + diry);
25         x ← x + dirx if posFree else x
26     }
27 }
```

types. The bird will only approach p if it is not isolated; otherwise, it will continue along its current direction (line 8). Similarly, at lines 19–21, the bird only moves to the position pointed at by its heading vector if that position is free, i.e., if no other bird is currently there; otherwise, it stays in its current location.

Fleeing from a predator. Until now, we have considered a flock that is unperturbed by external threats. We now want to consider one that may be threatened, for instance, by a bird of prey. This means that birds should be able to recognize a predator and flee from it when it gets too close. At the same time, the flocking dynamics that we have gradually introduced so far should be preserved.

Listing 5 shows the implementation of this new kind of flock. Please notice that we refine the **pick** operator introduced in Listing 2 by making it typed. For instance, at line 8 the bird selects another member of the flock, and then performs the same operations seen in Listing 4. We omit some of the instructions for sake of brevity. Similarly, at line 12 the bird selects a Predator, and then evaluates its distance from itself. If this distance is too small, the bird will not perform its usual approach to its flockmate; instead, it will flee from the predator. We model

Listing 6: Constraints.

```
1 assume {
2     GridCentre = forall Bird b,
3         x_b > 490 and x_b ≤ 510 and y_b > 490 and y_b ≤ 510
4     DifferentPositions = forall Bird b1, forall Bird b2,
5         b1 = b2 or x_b1 ≠ x_b2 or y_b1 ≠ y_b2
6     DirectionNotNull = forall Bird b, dirx_b ≠ 0 or diry_b ≠ 0
7 }
```

this fleeing behaviour by computing a repulsive heading vector $(\text{e\_dirx}, \text{e\_diry})$ and letting the bird follow it if the predator is closer than a given parameter λ.

3 Simulation results

The aim of this section is to understand whether the specifications provided so far allow the flock to remain compact. We set up an experimental scenario in which all birds start from non-deterministically chosen positions in a small area, and a single bird of prey flies through the centre of this area, threatening the flock. We aim at showing that the attack of the predator perturbs the flock, which becomes scattered, and that the flock manages to regroup once the predator leaves.

Let us first assume that all agents are placed within an *arena*, modelled as a 1024×1024 square. If the birds could initially assume any position within the arena, they could be very scattered. Instead, we want the birds to start close to each other, as an unperturbed flock would be. Similarly, we want birds not to start from the same position as others, nor to be stationary (i.e., with a null heading vector).

Listing 6 shows how to model these initial constraints, by listing them into a new section of the specifications titled **assume**. Each constraint is expressed through a quantified predicate, like those seen in Section 2. Lines 2–3 establish that birds can only be placed in a 20×20 sub-grid at the centre of the arena. Please note that, due to this initial configuration and the limited number of steps we will analyse, it never occurs that the flock reaches the edges of the arena. Lines 4–5 state that two agents cannot assume the same initial position. Finally, line 6 prescribes non-null heading vectors for every bird.

Listing 7 specifies the predator agent. Our predator has the same attributes as the birds in the flock: a position (x, y) and a heading vector $(\text{dirx}, \text{diry})$. We give it a very simple behaviour, such that it moves in a straight line along its initial heading vector. To ensure that a predator intersects the flock, the initial position and the heading vector are given determined values (lines 3–6). We give the predator a longer heading vector than those of flock birds, modelling the fact that it moves faster. The predator's behaviour is shown at lines 8–12 and is exactly like the one seen in Listing 1, modelling movement in a straight line.

As mentioned earlier, our aim is to check whether the flock preserves cohesion after an attack. Specifically, as the predator closes in on the flock, birds will flee

Listing 7: Predator specifications.

```
1  agent Predator {
2      Interface =
3          x: 480;
4          y: 480;
5          dirx: 3;
6          diry: 3
7
8      Behaviour = Move; Behaviour
9      Move = {
10         x ← x + dirx;
11         y ← y + diry
12     }
13 }
```

Listing 8: Specifying a cohesion requirement.

```
1 check {
2     Cohesion = after B forall Bird b1, forall Bird b2,
3         id_b1 = id_b2 or d((x_b1, y_b1), (x_b2, y_b2)) ≤ k
4 }
```

from it and thus the distance between any two of them will increase. We want to study whether this distance manages to decrease again after the predator leaves. Listing 8 shows the formalization of this property within another section of the specifications, titled **check**. The property described above is shown at line 3. Here, **after** B denotes that the predicate, which asserts that any two birds are not farther apart than a parameter k, should hold B steps after the initial state. In LTL [27], this construct would be expressed as X^B, i.e., a sequence of B applications of the "next" operator X.

To quickly assess whether our flock is capable of displaying this kind of behaviour, we implemented a *simulation* workflow (Fig. 2) that produces random traces of our specification. Intuitively, we perform a structural encoding of our specifications into a sequential imperative program [10], and then feed the program into a reachability analysis tool to produce one or more random traces of a desired length. These traces are then automatically translated into the specification syntax and shown to the user. The simulation traces that we generate this way also contain information about the satisfaction of properties included in the specification.

To improve the performance of this workflow, we introduce a *concretization* step before feeding the program to the back end. This step is a source-to-source transformation in which we replace nondeterministic assignments in the program with deterministic assignments to *concrete* values, randomly-chosen among the feasible ones. Specifically, we concretize the initial values of the agents' attributes (based on their initial values and on the contents of the **assume** section), the

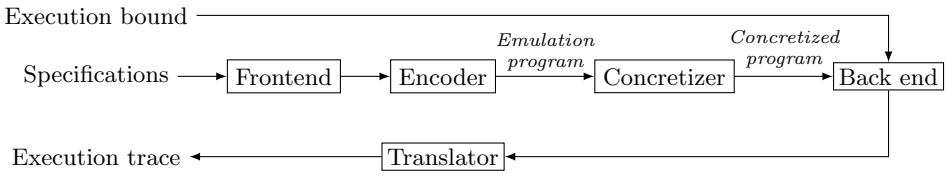


Fig. 2: Workflow to simulate our specifications.

Table 1: Parameters in our model and their values used in the simulation process.

Name	Description	Value
B	Bound for the cohesion property	600
D	Maximal absolute value of heading vector components for birds	2
G	Size of the arena	1024
k	Maximal distance to satisfy the cohesion property	40
δ	Isolation distance	32
λ	Safe distance from predator	32
ν	Used to estimate the future position of the predator	2
ω	Used to estimate the future position of the bird to approach	14
	Number of Bird agents	29
	Number of Predator agents	1

agents' scheduling, and the identifiers returned by **pick** statements. This way, we partially resolve nondeterminism upfront, alleviating the workload of the back end and leading to faster generation of traces. To implement this workflow, we extended SLiVER,[2] a tool originally aimed at formal verification of collective systems [10,11]. Namely, we added support for the new constructs described in Section 2, adapted its program generator to the simulation use case, and implemented the concretization step.

Table 1 sums up the parameters in our models and their values in our simulations, as well as the composition of the system. Notice that we use B both as the bound of the cohesion property and as the desired length of our simulations. We assume round-robin scheduling: thus, every trace is a sequence of *epochs* in which each agent performs exactly one action. It is worth recalling that, in this context, an atomic block is regarded as a single action. In our view, this assumption, though demanding, is reasonable when modelling a real-world system. Furthermore, it is significantly weaker than the implicit synchrony assumptions of other models [29,2], in which all agents are required to evolve in *lockstep*. In fact, this requirement implies that the future state of individual agents depends on the current state of the whole system, and that state changes happen simultaneously for all agents.

Fig. 3 shows the visual representation of a trace generated through this simulation process. Each bird is represented by a triangle pointing in the direction of its heading vector; the predator is the larger, red triangle with black outline.

[2] `https://github.com/labs-lang/sliver/`

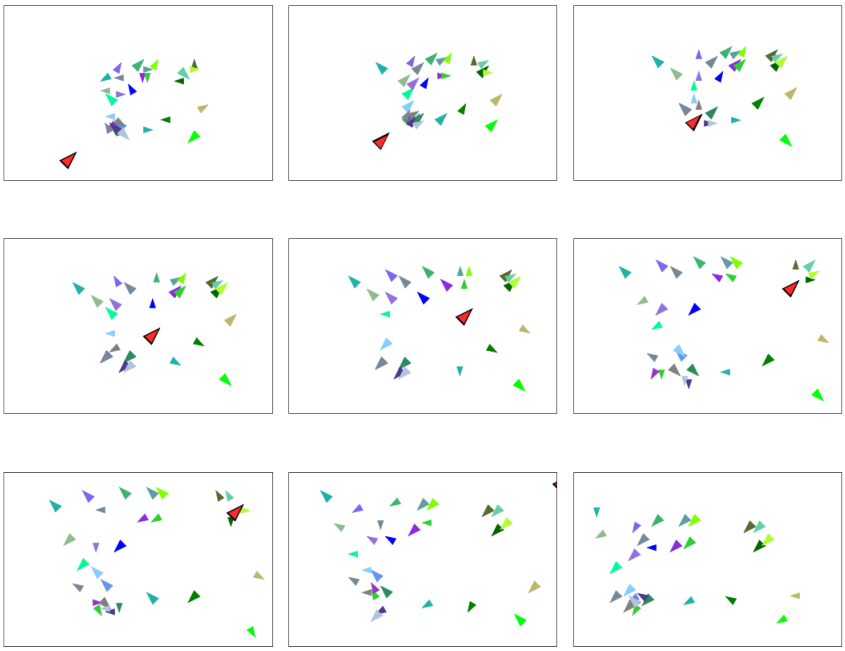


Fig. 3: A trace generated through simulation. The predator is the red triangle with black outline.

Notice that, in this trace, birds are never in the same position, and overlapping triangles are merely an artefact of the visualization. As we expected, the trace shows that the predator attack does introduce a certain amount of dispersion in the flock as birds move to avoid the threat; however, birds are eventually able to regroup and reorient themselves coherently, satisfying the property that we specified in Listing 8. As a final remark, we should stress that our simulation workflow helped us throughout the specification process: for instance, they made us realize the potential for flock dispersion in Listing 3, guiding us to develop the more refined Listing 4.

4 Related work

Models of flocking behaviour in the literature rely either on equational modelling, using for instance differential equations [34], discrete-time dynamics [2,29] or statistical mechanics [3]; on decentralized control laws, either developed ad-hoc [35] or synthesized from a centralized controller [23]; or on language-based specifications, such as the ones presented in this work.

An advantage of language-based approaches is that models can be gradually refined, or compared against each other, with little effort. For instance, the

framework of [20] has been used to model different predator tactics (such as attacking the centroid of the flock, the nearest prey, or the most isolated one) and different versions of flocking behaviour: simulations show that preys with a more individualistic behaviour are more likely to get caught, while more social flocks provide better chances of survival [8].

Formal specification languages also enable exhaustive exploration of the state space, which can provide strong guarantees about the behaviour of a system, or find subtle bugs that are hard to detect through simulations alone. As an example, the *alpha algorithm* [36], which was supposed to make a flock of scattered agents aggregate in a small region of space, has been found to be incorrect [18,1] by verifying models of the algorithm written in ISPL [22] or NuSMV [6]. Emulation programs may similarly enable formal analysis of high-level specification by means of structural encodings towards lower-level languages, allowing to reuse different existing verification technologies [10,12].

Bottom-up and simulation-aided design is also common in the engineering of robot swarms and related classes of robotic systems [4]. In this context, robots are typically programmed at the individual level, using either general-purpose languages such as C++ or Python, or higher-level, domain-specific formalisms [9,25], possibly relying on existing robotic middleware such as ROS [28]. The resulting programs are evaluated by simulating the robots under one of several available simulation platforms [17,26,30] to empirically check whether the swarm exhibits an adequate collective behaviour. These platforms also support physical simulations, allowing to check how real-world phenomena (like gravity, collisions, etc.) may interfere with the agents. These kinds of interactions with the environment are out of the scope of this work, but it might be worthwhile to integrate these platforms into our simulation workflow.

5 Conclusion

In this work, we have considered the natural collective behaviour known as flocking, and we have shown how compositional models can help reasoning about the individual dynamics that lead to its emergence. To this end, we gradually refined an extremely simple individual behaviour into a more elaborate, but still rather compact and intuitive, final specification. This specification allows a flock of birds to display some interesting collective features. By feeding it to an automated simulation workflow, we indeed showed that the birds are able to counter the threat of a predator by splitting into smaller groups that reassemble once the danger subsides. This successfully reproduces the behaviour observed both in real-life flocks and in other models [2].

We are considering several interesting directions for future work on this subject. Our simulation workflow is still experimental and, while it does simulate the model of Section 2 well, we do not expect it to work in every scenario. For instance, specifications that contain guarded statements may be hard to simulate, since some concretizations may fail to satisfy some guards and thus make it impossible to produce a trace of the desired length. To work around these is-

sues, we plan to customize the back end so that we can modify the concretization constraints until a valid trace is obtained.

We intend to complement the simulation-based approach shown in this work with exhaustive state space exploration techniques that may formally prove the emergence of desired collective features, regardless of the initial state or the system or the specific interactions between agents. We may achieve this by adapting existing techniques based on verification of emulation programs [10], possibly extending them to support expressive temporal logics such as LTL [27]. This goal may also benefit from a more rigorous formalization of the linguistic constructs introduced in Section 2, which is also reserved for future work. Since the cost of exhaustive analysis may be prohibitive for very large system, we plan to further extend our simulation workflow to enable lightweight formal methods, such as statistical model checking [31], allowing us to at least obtain statistical evidence on the correctness of these systems. Our framework's capability to check for property satisfaction during simulation can be seen as a rudimental form of runtime verification [21]. Extending this capability to larger classes of monitorable properties [15] is also planned as future work.

We can trivially parallelize our simulation workflow by running it on multiple machines at once; moreover, we might further improve performances by implementing distributed techniques in the back end [16]. Working in these two directions may allow us to generate large numbers of traces for massive systems. Generating effective visualizations from a textual trace is also essential to support the design process. So far, our automated visualization tool (which we used, for instance, to generate Fig. 3) is tailored to the flocking case study: building a more generic framework, or integrating our workflow into existing simulation platforms, would be interesting contributions.

References

1. Antuña, L.R., Araiza-Illan, D., Campos, S., Eder, K.: Symmetry reduction enables model checking of more complex emergent behaviours of swarm navigation algorithms. In: 16th Annual Conference Towards Autonomous Robotic Systems (TAROS). LNCS, vol. 9287, pp. 26–37. Springer (2015). https://doi.org/10.1007/978-3-319-22416-9_4
2. Ballerini, M., Cabibbo, N., Candelier, R., Cavagna, A., Cisbani, E., Giardina, I., Lecomte, V., Orlandi, A., Parisi, G., Procaccini, A., Viale, M., Zdravkovic, V.: Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study. Proceedings of the National Academy of Sciences **105**(4), 1232–1237 (2008). https://doi.org/10.1073/pnas.0711437105
3. Bialek, W., Cavagna, A., Giardina, I., Mora, T., Silvestri, E., Viale, M., Walczak, A.M.: Statistical mechanics for natural flocks of birds. Proceedings of the National Academy of Sciences **109**(13), 4786–4791 (2012)
4. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: A review from the swarm engineering perspective. Swarm Intelligence **7**(1), 1–41 (2013). https://doi.org/10.1007/s11721-012-0075-2

5. Cederman, L.E.: Endogenizing geopolitical boundaries with agent-based modeling. Proceedings of the National Academy of Sciences **99 Suppl 3**, 7296–7303 (2002). https://doi.org/10.1073/pnas.082081099
6. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An opensource tool for symbolic model checking. In: 14th International Conference on Computer Aided Verification (CAV). LNCS, vol. 2404, pp. 359–364. Springer (2002). https://doi.org/10.1007/3-540-45657-0_29
7. De Nicola, R., Di Stefano, L., Inverso, O.: Multi-agent systems with virtual stigmergy. Science of Computer Programming **187**, 102345 (2020). https://doi.org/10.1016/j.scico.2019.102345
8. Demsar, J., Lebar Bajec, I.: Simulated predator attacks on flocks: A comparison of tactics. Artificial Life **20**(3), 343–359 (2014). https://doi.org/10.1162/ARTL_a_00135
9. Desai, A., Saha, I., Yang, J., Qadeer, S., Seshia, S.A.: DRONA: A Framework for Safe Distributed Mobile Robotics. In: ICCPS (2017). https://doi.org/10.1145/3055004.3055022
10. Di Stefano, L., De Nicola, R., Inverso, O.: Verification of distributed systems via sequential emulation. ACM Transaction on Software Engineering and Methodology **31**(3) (2022). https://doi.org/10.1145/3490387
11. Di Stefano, L., Lang, F.: Verifying temporal properties of stigmergic collective systems using CADP. In: 10th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA). LNCS, vol. 13036, pp. 473–489. Springer (2021). https://doi.org/10.1007/978-3-030-89159-6_29
12. Di Stefano, L., Lang, F., Serwe, W.: Combining SLiVER with CADP to analyze multi-agent systems. In: 22nd International Conference on Coordination Models and Languages (COORDINATION). LNCS, vol. 12134, pp. 370–385. Springer (2020). https://doi.org/10.1007/978-3-030-50029-0_23
13. Emlen, J.T.: Flocking behavior in birds. The Auk **69**(2), 160–170 (1952)
14. Finkelshtein, D., Kondratiev, Y., Kutoviy, O.: Individual based model with competition in spatial ecology. SIAM Journal on Mathematical Analysis **41**(1), 297–317 (2009). https://doi.org/10.1137/080719376
15. Francalanza, A., Aceto, L., Ingólfsdóttir, A.: On verifying hennessy-milner logic with recursion at runtime. In: 6th International Conference on Runtime Verification (RV). LNCS, vol. 9333, pp. 71–86. Springer (Sep 2015). https://doi.org/10.1007/978-3-319-23820-3_5
16. Inverso, O., Trubiani, C.: Parallel and distributed bounded model checking of multi-threaded programs. In: 25th Symposium on Principles and Practice of Parallel Programming (PPoPP). pp. 202–216. ACM (2020). https://doi.org/10.1145/3332466.3374529
17. Koenig, N., Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). vol. 3, pp. 2149–2154 vol.3. IEEE (2004). https://doi.org/10.1109/IROS.2004.1389727
18. Kouvaros, P., Lomuscio, A.: A counter abstraction technique for the verification of robot swarms. In: 29th Conference on Artificial Intelligence (AAAI). pp. 2081–2088. AAAI (2015)
19. Kuylen, E., Liesenborgs, J., Broeckhove, J., Hens, N.: Using individual-based models to look beyond the horizon: The changing effects of household-based clustering

of susceptibility to measles in the next 20 years. In: 20th International Conference on Computational Science (ICCS). LNCS, vol. 12137, pp. 385–398. Springer (2020). https://doi.org/10.1007/978-3-030-50371-0_28
20. Lebar Bajec, I., Zimic, N., Mraz, M.: Simulating flocks on the wing: The fuzzy approach. Journal of theoretical biology **233**, 199–220 (2005). https://doi.org/10.1016/j.jtbi.2004.10.003
21. Leucker, M., Schallhart, C.: A brief account of runtime verification. Journal of Logic and Algebraic Programming **78**(5), 293–303 (2009). https://doi.org/10.1016/j.jlap.2008.08.004
22. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: An open-source model checker for the verification of multi-agent systems. Software Tools for Technology Transfer **19**(1), 9–30 (2017). https://doi.org/10.1007/s10009-015-0378-x
23. Mehmood, U., Roy, S., Grosu, R., Smolka, S.A., Stoller, S.D., Tiwari, A.: Neural flocking: MPC-based supervised learning of flocking controllers. In: 23rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS). LNCS, vol. 12077, pp. 1–16. Springer (2020). https://doi.org/10.1007/978-3-030-45231-5_1
24. Olfati-Saber, R.: Flocking for multi-agent dynamic systems: Algorithms and theory. IEEE Transactions on Automatic Control **51**(3), 401–420 (2006). https://doi.org/10.1109/TAC.2005.864190
25. Pinciroli, C., Beltrame, G.: Buzz: An extensible programming language for heterogeneous swarm robotics. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 3794–3800. IEEE (2016). https://doi.org/10.1109/IROS.2016.7759558
26. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L.M., Dorigo, M.: ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence **6**(4), 271–295 (2012). https://doi.org/10.1007/S11721-012-0072-5
27. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (FOCS). pp. 46–57. IEEE (1977). https://doi.org/10.1109/SFCS.1977.32
28. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.Y.: ROS: An open-source robot operating system. In: ICRA Workshop on Open Source Software (2009)
29. Reynolds, C.W.: Flocks, herds and schools: A distributed behavioral model. In: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987, Anaheim, California, USA, July 27-31, 1987. pp. 25–34. ACM (1987). https://doi.org/10.1145/37401.37406
30. Rohmer, E., Singh, S.P.N., Freese, M.: V-REP: A versatile and scalable robot simulation framework. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS. pp. 1321–1326. IEEE (2013). https://doi.org/10.1109/IROS.2013.6696520
31. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: 16th International Conference on Computer Aided Verification (CAV). LNCS, vol. 3114, pp. 202–215. Springer (2004). https://doi.org/10.1007/978-3-540-27813-9_16
32. Shi, H., Wang, L., Chu, T.: Flocking of multi-agent systems with a dynamic virtual leader. International Journal of Control **82**(1), 43–58 (2009). https://doi.org/10.1080/00207170801983091

33. Stiglitz, J.E., Gallegati, M.: Heterogeneous interacting agent models for understanding monetary economies. Eastern Economic Journal **37**(1), 6–12 (2011). https://doi.org/10.1057/eej.2010.33
34. Toner, J., Tu, Y.: Flocks, herds, and schools: A quantitative theory of flocking. Physical Review E **58**(4), 4828–4858 (1998). https://doi.org/10.1103/PhysRevE.58.4828
35. Vásárhelyi, G., Virágh, C., Somorjai, G., Tarcai, N., Szörényi, T., Nepusz, T., Vicsek, T.: Outdoor flocking and formation flight with autonomous aerial robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 3866–3873. IEEE (2014). https://doi.org/10.1109/IROS.2014.6943105
36. Winfield, A.F.T., Liu, W., Nembrini, J., Martinoli, A.: Modelling a wireless connected swarm of mobile robots. Swarm Intelligence **2**(2-4), 241–266 (2008). https://doi.org/10.1007/s11721-008-0018-0