Check for updates

RESEARCH ARTICLE

REVISED Towards an integrated automatic design process for robot swarms

[version 2; peer review: 3 approved]

Darko Bozhinoski, Mauro Birattari

IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

v2 **First published:** 27 Sep 2021, **1**:112
https://doi.org/10.12688/openreseurope.14025.1
Latest published: 04 Nov 2022, **1**:112
https://doi.org/10.12688/openreseurope.14025.2

Abstract

Background: The specification of missions to be accomplished by a robot swarm has been rarely discussed in the literature: designers do not follow any standardized processes or use any tool to precisely define a mission that must be accomplished.
Methods: In this paper, we introduce a fully integrated design process that starts with the specification of a mission to be accomplished and terminates with the deployment of the robots in the target environment. We introduce Swarm Mission Language (SML), a textual language that allows swarm designers to specify missions. Using model-driven engineering techniques, we define a process that automatically transforms a mission specified in SML into a configuration setup for an optimization-based design method. Upon completion, the output of the optimization-based design method is an instance of control software that is eventually deployed on real robots.
Results: We demonstrate the fully integrated process we propose on three different missions.
Conclusions: We aim to show that in order to create reliable, maintainable and verifiable robot swarms, swarm designers may benefit from following standardised automatic design processes that will facilitate the design of control software in all stages of the development.

Keywords

swarm robotics, integrated automatic design process, optimization-based design method, model-driven engineering, domain-specific languages (DSL),

H2020 This article is included in the Horizon 2020 gateway.

Open Peer Review

Approval Status ✓ ✓ ✓

	1	2	3
version 2 (revision) 04 Nov 2022			✓ view
version 1 27 Sep 2021	✓ view	✓ view	✓ view

1. **Adam Schroeder**, University of Toledo, Toledo, USA
2. **Alan Millard**, University of York, York, UK
3. **Edmund Hunt**, University of Bristol, Bristol, UK
James Ward, University of Bristol, Bristol, UK

Any reports and responses or comments on the article can be found at the end of the article.



This article is included in the European Research Council (ERC) gateway.



This article is included in the Robotics collection.

Corresponding authors: Darko Bozhinoski (darko.bozhinoski@ulb.be), Mauro Birattari (Mauro.Birattari@ulb.be)

Author roles: Bozhinoski D: Conceptualization, Data Curation, Formal Analysis, Methodology, Project Administration, Software, Validation, Visualization, Writing – Original Draft Preparation; **Birattari M**: Conceptualization, Funding Acquisition, Methodology, Project Administration, Supervision, Validation, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (DEMIURGE Project, grant agreement No [681872]) and from Belgium's Wallonia-Brussels Federation through the ARC Advanced Project GbO (Guaranteed by Optimization). DB and MB acknowledge support from the Belgian Fonds de la Recherche Scientifique (FNRS), of which they are a Postdoctoral Researcher and a Research Director, respectively.
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2022 Bozhinoski D and Birattari M. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: Bozhinoski D and Birattari M. **Towards an integrated automatic design process for robot swarms [version 2; peer review: 3 approved]** Open Research Europe 2022, **1**:112 https://doi.org/10.12688/openreseurope.14025.2

First published: 27 Sep 2021, **1**:112 https://doi.org/10.12688/openreseurope.14025.1

REVISED **Amendments from Version 1**

The new version is a minor revision which includes typographic corrections and clarifications based on reviewers' comments:

- rephrasing and clarification of certain concepts;

- updating the Listings and Figures that demonstrate the Swarm Mission Language (SML) syntax for better readability;

- extending the Conclusions by adding new research directions as future work.

Any further responses from the reviewers can be found at the end of the article

1 Introduction

In this paper, we make two original contributions: 1) we define a textual language for the specification of missions to be performed by a robot swarm and 2) we realize an engine that transforms a mission specification given in the aforementioned language into an objective function (and other configuration files) needed to automatically perform the design by optimization of a robot swarm that will accomplish the mission. These two original contributions, combined with an existing method for the design by optimization of control software for robot swarms[1], enable a fully integrated process that starts with the specification of a mission to be accomplished and terminates with the deployment of the robots in the target environment.

In swarm robotics, a large number of robots perform a mission that can not be accomplished by a single robot[2,3]. The collective behavior of the robot swarm is obtained through individual robots collaboration and cooperation. Hence, the collective behavior of a robot swarm is a result of the local interactions between the individual robot and its neighbors and its environment[4]. In the general case, the complex nature of these interactions are virtually impossible to trace to the behaviour of the individual robots, which creates a gap between the collective behaviour that one wishes to obtain and what each of the individual robots should do. Bridging this gap is one of the main challenges in swarm robotics and the lack of a general methodology to bridging this gap influences how control software for robot swarms is designed and realized.

So far, robot swarms have been mostly designed manually: an individual-level behavior is iteratively improved and tested until a desired collective behavior is obtained. This non-systematic approach is neither reliable, nor consistent. The quality of the resulting solution strongly depends on the experience and intuition of the designer. To avoid, or at least reduce, the uncertainty induced by the crucial role of the human designer, automatic and semi-automatic design approaches have been proposed[5–8].

Although a number of automatic approaches have been proposed in the software and system engineering literature[9], they have not been investigated in the context of swarm robotics. This is because these approaches, which focus on decoupling and automatizing the different phases of the robot life-cycle, appear to be inappropriate in swarm robotics. Indeed, they model the system to be realized at a level of abstraction that is too high and neglects the complex robot-robot and robot-environment interactions that characterize the operation of a robot swarm. For example, these approaches assume that it is possible to establish a mapping between high-level collective goals of the swarm and low-level individual behaviors of the robots comprised therein[10–12]. Unfortunately, the swarm robotics praxis indicates that making such a mapping explicit is not generally possible[2,3].

For this reason, the most promising approaches that have been proposed so far for the automatic design of robot swarms are in the area of design by optimization[13]. In design by optimization, the design problem is re-formulated into an optimization problem: an optimization algorithm searches a space of candidate solutions to maximize an objective function. In the context of the application of design by optimization to swarm robotics, a candidate solution is an instance of control software and the objective function is a mission-dependent metric that measures the performance of the swarm on the given mission[3]. Depending on whether the design phase happens before or after the deployment of the software on the robots, we can distinguish between two classes of design methods[14]: off-line and on-line. In this work, we focus on off-line automatic design[15], although the proposed ideas could be adapted to on-line design as well. Within an off-line design process, the performance of candidate designs are assessed by an optimization algorithm typically via computer-based simulations. After the optimization algorithm terminates, the selected design is deployed to the individual robots and the swarm is placed in its target environment.

In this paper, we present a first instance of a fully automatic and integrated process for the design of collective behaviors for robot swarms. The novel contribution that enables this integrated process is the definition of a formal and systematic approach to the specification of missions to be accomplished by a robot swarm. This approach is rooted in model-driven engineering[16]: a research direction in software engineering that aims at simplifying the design, implementation, and realization of complex software systems by shifting the designer's attention from code to models. Recently, model-driven engineering has been often used in the design of robot systems as it dispenses the designer from reasoning on complex robot behaviours at the code level, which is cumbersome and undesirable[17]. Indeed, in model-driven engineering, models are expressed at an appropriately high level of abstraction using domain-specific languages with concepts that are close to the problem domain and not directly bound to the robotic platform at hand. This makes the realization of complex systems manageable as models are easier than code to specify, understand, and maintain[18]. Specifically, in this paper we present and demonstrate an integrated automatic design process for robot swarms that, starting from requirements specified in a textual language, generates code and deploy it on real robots. We introduce a language that we call swarm mission language (SML) which allows one to specify missions to be accomplished by a robot swarm. In the paper, we present a first implementation of SML that supports the specification of missions in which rewards and penalties can be expressed with reference to regions: that is, rewards and penalties are computed according to whether robots (and/or relevant

objects) are in a particular part of the environment at specific moments in time. Many of the most studied swarm robotics missions like aggregation, foraging and collective exploration can be modeled through the concept of a region. Furthermore, we develop an engine that translates a range of missions specified in SML into all the resources needed to launch `Chocolate`[1], a state-of-the-art automatic method for the off-line design of robot swarms[19] that, using simulation performed by ARGoS3[20], produces control software that can be directly ported to e-puck robots[21]. To demonstrate SML and the integrated automatic design process, we specify three missions and we automatically generate the control software that allows a swarm of e-puck robots to accomplish them.

2 Related work

2.1 Automatic off-line design of robot swarms

In swarm robotics, neuro-evolutionary robotics[22,23] is the most studied automatic design approach. In neuro-evolutionary swarm robotics[6], each individual robot control software is a neural network. The parameters of the neural network are obtained via an evolutionary algorithm that optimizes a mission-specific objective function taking sensor readings as an input and returning actuation commands as an output. A large literature shows that neuro-evolutionary robotics is able to produce robot swarms that can perform well in a variety of missions[4,24]. However, the neuro-evolutionary approach does not appear to be able to scale in complexity for realistic robot swarm applications. One of the main causes is the difficulty to overcome the so called reality gap[4]. The reality gap is the discrepancy between reality and the simulation models used in the design process[25]. Because of the reality gap, the performance of control software developed in simulation typically drops when the control software is ported to the real robots. It has been argued that this drop in performance is the result of a sort of overfitting of the obtained solution to the particular conditions encountered during the design process[25–27].

AutoMoDe[7,28] is an alternative approach that deviates from traditional neuro-evolutionary robotics. It aims to address one of the main concerns in neuro-evolutionary robotics and that is the reality gap due to the excessive representation power of neural networks. Inspired by the notion of bias-variance tradeoff[29], AutoMoDe produces control software with restricted representational power. It does so by selecting, combining, and fine-tuning a set of predefined modules. AutoMoDe is a general, abstract framework. To define a design method that can be used to design control software, AutoMoDe must be specialized to the specific platform at hand, as formally described by a reference model. Also, a number of elements need to be defined, including the optimization algorithm to be adopted, the modules that will be used by the optimization algorithm, and the architecture into which the predefined modules should be combined. Up to now, AutoMoDe has been specialized for a specific version of the e-puck robot; the optimization algorithms that have been adopted are F-Race[30,31], Iterated F-Race[32], and simulation annealing[33]; and the architecture in which modules have been combined are probabilistic finite-state machines[1,34] and behaviour trees[35]. `Chocolate` is a state of the art automatic design method from the AutoMoDe family that achieves significantly better results in crossing the reality gap than neuro-evolutionary approaches[1]. It uses Iterated F-Race as an optimization algorithm and probabilistic finite-state machines as a control architecture.

An aspect that is rarely discussed in the literature is the specification of the mission for which the automatic design method must generate control software[3]. Designers of robot swarms do not follow any standardized processes or use any tool that precisely defines the mission to be accomplished. For example, the aforementioned ARGoS simulator enables specifying missions through a combination of XML files and loop functions defined in C++[20]. It provides a great deal of flexibility for designers to design missions the way they prefer. Designers can incorporate a rich variety of elements related to the operational context or the characteristics of the robots. However, it is a tedious process to manually specify all elements of a mission to be performed by a robot swarm without following any predefined process. This might create situations where designers use environmental elements that are important to obtain a desired collective behaviour for one application scenario, while omitting the same elements if they impede the desired collective behaviour. This *ad-hoc* mission specification process might create confusion between designers that are working on a same set of missions. Moreover, if requirements are not defined explicitly, it is impossible to check the consistency of mission models. It is also impossible to tell whether a robot swarm eventually performs the mission successfully or not. To simplify the communication between designers and to check for possible inconsistencies, all these aspects must be formally defined and automatized.

2.2 System and software engineering for robotics

System and software engineers have made their contribution to robotics by providing tools and standardized methodologies for the specification and the definition of robot systems[36]. In system and software engineering, researchers have addressed a variety of emerging challenges in the design and development of complex systems by providing generic solutions, often disregarding their specific nature. One of the main challenges is collecting requirements[37]. To simplify the requirements' elicitation for complex system, researchers have focused on defining standard processes and methods that can be fully or partially automatized.

In requirements engineering, there is a basic assumption that underlines most of the approaches: if all requirements are known, it is always possible to decompose any high-level goal into a sequence of operations that allow the system to attain it. For example, goal orientation[38] is a widely recognized process for eliciting, modeling, specifying and analyzing system requirements. Goals are statements of intent organized in AND/OR structures that can range from high-level strategic concerns to low-level technical requirements and assumptions on the system and the environment where it operates. It is generally accepted that robotic systems are too complex for engineers to obtain complete requirements on the system and the environment[37]. Hence, it is typically assumed that robotic systems are highly uncertain due to incomplete requirements. However, this is not a valid assumption to be made in swarm robotics. The uncertainty of a robot swarm is not only the result of incomplete requirements, but it mostly emerges from the complex interactions between the robots and between the robots and the

environment. This means that the gap between the high-level swarm goals and the low-level robot behaviours is inevitable.

In a number of works[10–12], system and software engineers have synthesized low-level robot behaviours from high-level mission descriptions. For example, FLYAQ[11] is a tool that allows defining missions for teams of multi-copters. Starting from a high-level description of the mission, FLYAQ automatically generates a detailed flight plan for a team of autonomous multicopters that can perform the specified mission, while preventing collisions between multicopters and obstacles. FLYAQ was developed based on model-driven engineering principles. It uses a family of domain-specific languages for specifying civilian missions for multi-robot systems[10]. Each language focuses on a certain aspect of the system:

- *Monitoring modeling language:* a language that enables the specification of the mission goals complemented by the definition of the context in which the mission will be realized.
- *Robot language (RL):* a language to specify the type and the configuration of the robots that will be in charge of realizing the specified mission.
- *Behaviour language (BL):* a language that specifies robot atomic movements and actions.

However, FLYAQ cannot be used in swarm robotics because the nature of robot swarms does not support the synthesis of the individual robot behaviour from the collective swarm behaviour.

To the best of our knowledge, requirements specification for swarm robotics has not been properly addressed as a research question. The highest level of abstraction that has been extensively discussed in swarm robotics is the development process. A work in this direction is Buzz, a scripting language for programming heterogeneous robot swarms[39,40]. The language offers primitives to define swarm behaviors, both in a bottom-up and in a top-down fashion. The formal specification of requirements for robot swarms was partially discussed by Brambilla *et al.*[41] in a work devoted to a top-down design approach based on prescriptive modeling and model checking. The approach of Brambilla *et al.* consists of four phases to specify, design, realize, and validate a robot swarm. In the first phase, the developer specifies the requirements using temporal logic. However, the approach does not provide a precise process definitionof requirements specification but rather a set of examples on how designers can use probabilistic computation tree logic to specify swarm-level requirements.

We believe that model-driven engineering can provide support in gathering explicit and clear requirements for robot swarms. Model-driven engineering has been explored in the design of complex systems being an essential factor in reducing costs and development time. It has been successfully used in various domains including avionics, automotive, and telecommunications[16]. For example, domain-specific modelling (DSM)[42] is a powerful methodology in model-driven engineering, which enables users to model systems using concepts close to the problem definition.

3 Integrated automatic design process for robot swarms

We present here the main phases and key activities to design control software for a robot swarm in a systematic way (Figure 1). In an automatic off-line design process, we identify three phases: requirements specification, design by optimization, and deployment of the control software on the robots.

In *requirements specification*, the designer identifies and declares all the characteristics of the robot swarm, the target environment in which it will operate, the mission that it should accomplish, the objective that it should fulfil, the possible constraints, etc. In the current state of the art, no standard process has been defined for collecting requirements. Typically, designers specify missions informally and in an *ad-hoc* manner, which makes specifications vague and eventually hinders a final verification of whether the swarm developed satisfies the requirements or not[43]. Starting from the requirements, the designer defines an objective function to be then optimized in the second phase. As requirements are specified informally, this step must be performed manually and is discretionary, non-repeatable, and error prone. In the second phase, *design by optimization*, the control software of the individual robots comprised in the swarm is produced by an automatic design method[14,15]. An automatic design method is defined through: (i) a reference model of the robotic platform for which it can design control software; (ii) an optimization algorithm; and (iii) the space of control software it can possibly produce. The reference model is an abstraction of the robotic platform that specifies in formal terms the characteristics and capabilities of the robots; the optimization algorithm is the algorithm that drives the optimization process; and the space of the control software that can be produced is typically expressed by a parametric architecture and by the set of the possible values of its parameters.

The last phase is the *deployment on the robots*. It consists of all the activities related to the transfer of the control software produced to the robots in the target environment. Some

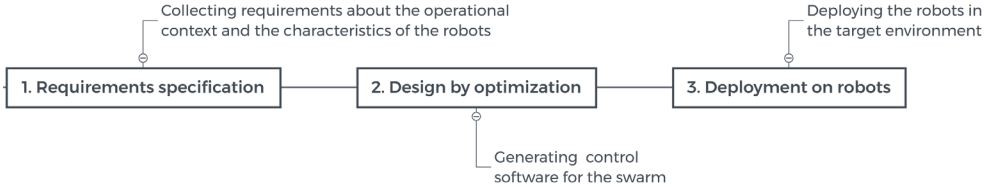


Figure 1. Automatic off-line design of robot swarms.

tools exist that are able to generate code that can be directly ported to the robots. For example, ARGoS[20], a multi-engine simulator for robot swarms, can currently generate code for a number of platforms including marXbot[44], e-puck[21], Thymio[45], Kilobot[46], and Khepera IV[47]. Due to its modular nature, ARGoS can be extended to generate code for a variety of robotic platforms.

3.1 Requirements specification for robot swarms

We present an approach that enables experts (control software designers) and non-expert users (non-technical end users) to specify requirements for robot swarms in a standard and consistent way. Figure 2 depicts the workflow of the automatic design process. The novelty of our approach is the requirements specification phase, outlined by the dash-dotted line in Figure 2.

The user has an informal picture of what the robot swarm should do and of the environment in which it should operate. From these informal requirements, a formal model of the mission goals and the target environment should be defined. We developed swarm mission language (SML) to allow users to specify requirements. The output of this phase is an objective function that will be subsequently optimized by the optimization process, and a model of the target environment to be used in the simulations performed within the automatic design process. The model of the target environment is specific to the automatic design method and the tools used. In our work, we used model-to-model transformation techniques to translate the missions specified in SML into configuration files for `Chocolate`.

3.2 Design by optimization

In the subsequent automatic design phase, we use `Chocolate` with a design budget of 200K simulation runs as an automatic design method to generate control software for the robot swarm. `Chocolate` operates on a set of six low-level behaviors and six conditions[1]. In this context, a low-level behavior defines how the robot operates its actuators in response to the readings of its sensors. On the other hand, a condition is an event that the robot perceives via its sensors and that determines whether the robot should transition from one behavior to another. Conditions contribute to determine which behavior is executed at any moment in time.

The low-level behaviors on which `Chocolate` operates are the following.

Exploration: the robot moves straight forward, if the front of the robot is clear of obstacles. If an obstacle is perceived via the front proximity sensors, the robot turns in-place for a random number of control cycles drawn in $\{0, ..., \tau\}$, where τ is an integer parameter $\in \{0, ..., 100\}$.

Stop: the robot stops its movement.

Phototaxis: the robot moves towards a light source. The robot moves forward while avoiding obstacles, if it does not perceive any light source.

Anti-phototaxis: the robot moves away from a light source. The robot moves forward while avoiding obstacles, if no light source is perceived.

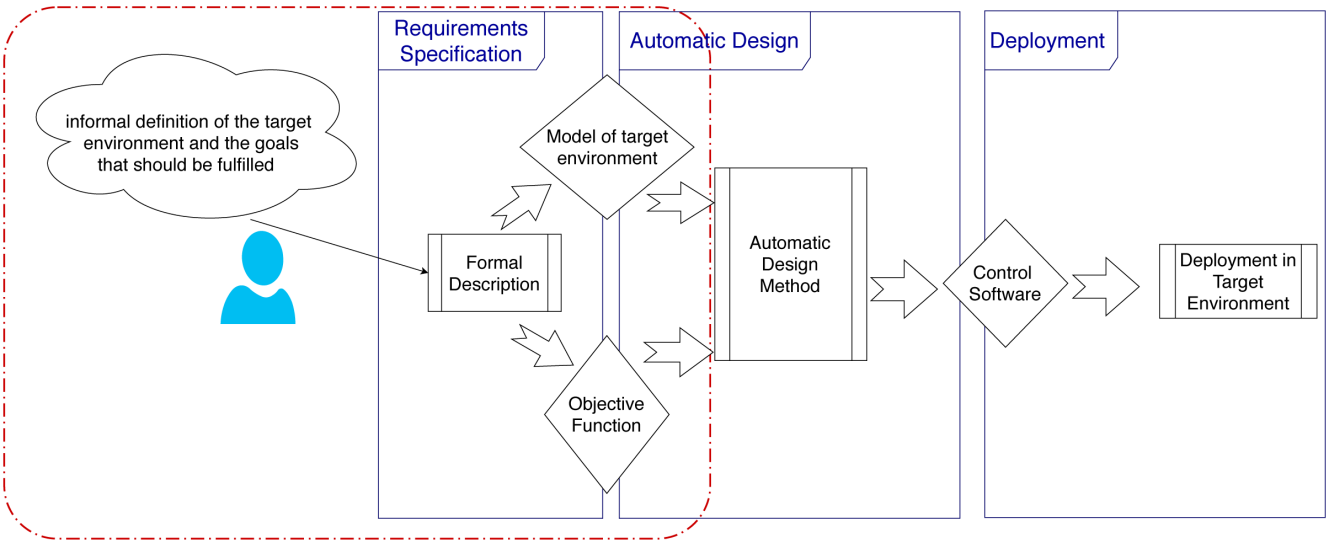


Figure 2. Integrated automatic design process for robot swarms.

Attraction: the robot moves towards its neighboring peers, following αV_d, where $\alpha \in [1, 5]$ controls the speed of convergence towards the peers. The robot moves straight forward while avoiding obstacles, if it does not perceive any peer.

Repulsion: the robot moves away from its neighboring peers, following $-\alpha V_d$, where $\alpha \in [1, 5]$ controls the speed of divergence. The robot moves straight forward while avoiding obstacles, if it does not perceive any peer in its neighborhood.

The conditions under which a robot switches from a behavior to another are the following.

Black-floor: true with probability β, if the ground situated below the robot is perceived as black.

Gray-floor: true with probability β, if the ground situated below the robot is perceived as gray.

White-floor: true with probability β, if the ground situated below the robot is perceived as white.

Neighbor-count: true with probability $z\ (n) = (1 + e^{\eta(\xi - n)})^{-1}$, where n is number of detected peers. The parameters $\eta \in [0, 20]$ and $\xi \in \{0, ..., 10\}$ control the steepness and the inflection point of the function, respectively.

Inverted-neighbor-count: true with probability $1-z\ (n)$.

Fixed-probability: true with probability β.

For more details on the low-level behaviours and conditions of `Chocolate`, we refer the reader to their original description[7].

4 An approach to specifying swarm missions

In this section, we present an approach to specifying swarm missions. To be able to specify missions, we need to understand the nature of the requirements in swarm robotics. First, we discuss a classification framework for swarm missions, then we develop a metamodel that defines the semantics of the Swarm Modeling Language (SML).

4.1 Classification framework for swarm missions

A classification framework of the main missions studied in the literature has already been proposed[4]. Missions have been classified in different categories: spatially organizing missions, navigation missions, collective decision-making, and other swarm missions.

Spatially organizing missions focuses on organizing and distributing robots and objects in the environment. This category consists of missions like aggregation (robots group in a region of the environment), pattern formation (robots position themselves on a regular lattice), chain formation (robots position themselves so as to connect two points in the environment), self-assembly and morphogenesis (robots physically connect to each other following a particular pattern), and object clustering and assembling (robots position objects in the environment).

Navigation missions focus on coordinating the movements of a swarm of robots. The following missions are part of this category: collective exploration (robots explore an unknown environment), coordinated motion, also known as flocking (robots move in formation similarly to schools of fish or flocks of birds), collective transport (robots cooperate to transport an object).

Collective decision-making is a set of missions where the focus is on how robots influence each other when making choices. Here, we can find missions like consensus achievement (robots reach a consensus on one choice among different alternatives) and task allocation (robots dynamically choose the task to execute in order to maximize performance). The last category is for missions that are outside the scope of the previous classes. Here, we can find missions like collective fault detection (robots autonomously detect failures and faulty behaviors) and human-swarm interaction. This classification framework is interesting to understand the different types of collective behaviours robots can perform. However, one important aspect that has not been discussed in this framework is how do we tell whether the swarm accomplishes its mission and how we quantify the degree to which it is successful. In this work, we classify missions in terms of the objective function that describes them. That being said, we propose a different classification framework based on a measure of success for the mission. In many of the missions mentioned above, the typical way to measure success is through the concept of a region. Rewards and penalties can be naturally given according to whether robots (and/or relevant objects) are in a specific region at a certain moment in time, or not.

4.2 SML metamodel

In this section, we describe the SML metamodel on which we base the Swarm Mission Language. In Figure 3, Figure 4, and Figure 5, we present the concepts used to design the language. The proposed abstractions are tailored to the literature in swarm robotics, as discussed in the previous section. With them, we intend to provide a way to define missions in a standard and consistent way. We formally define a mission as follows.

Definition 4.1 (Mission). A mission is a triplet $M = (E, S, O)$ where:

- E is the environment where the mission is performed;
- S is the robot swarm that should perform it;
- O is the objective function to be optimized.

The SML metamodel provides modeling constructs that enable the specification of the three aspects: the environment, the robot swarm and the objective function.

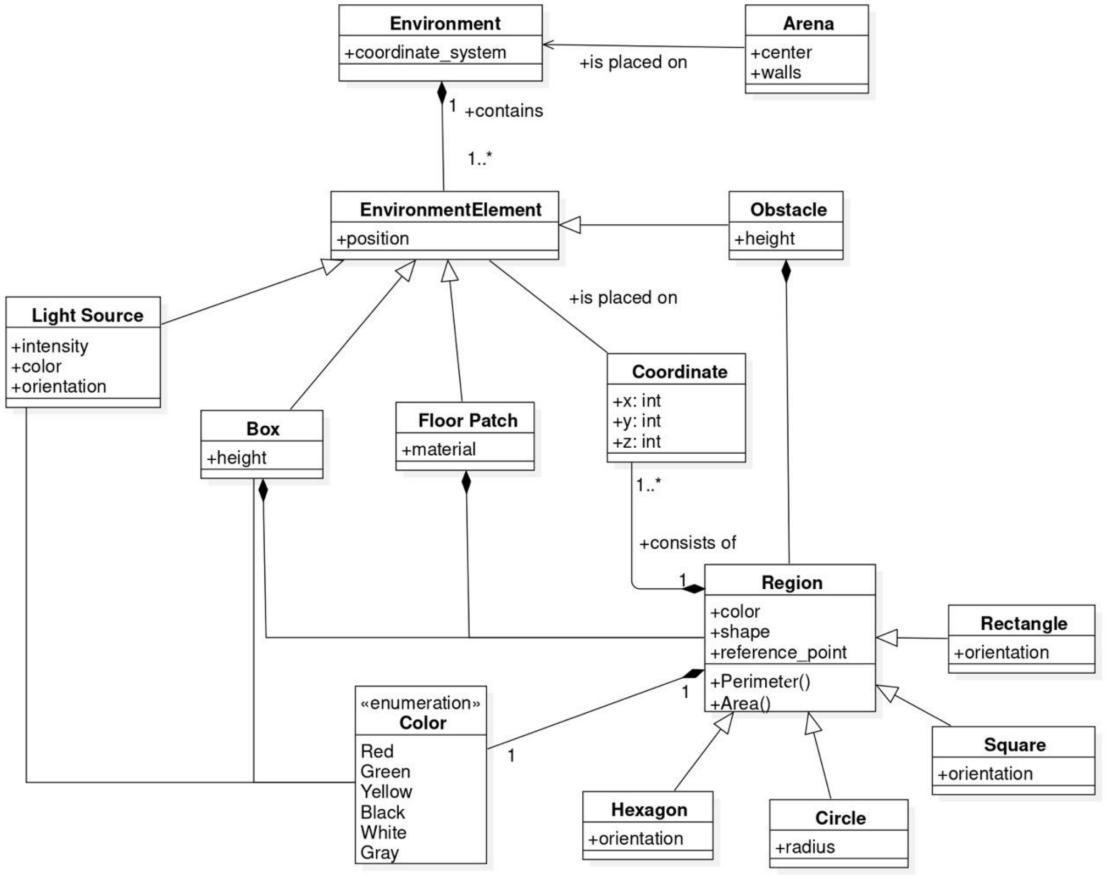


Figure 3. **Constructs to specify the environment.**

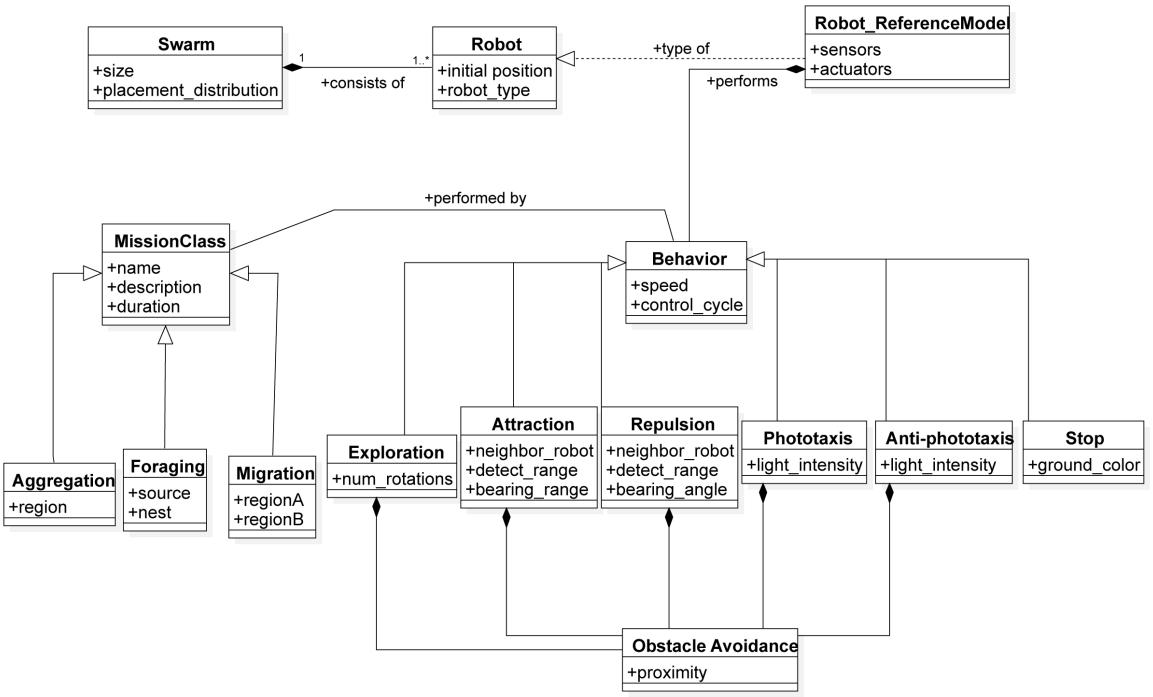


Figure 4. **Constructs to specify the overall robot swarm, individual robot behaviour, and the mission class.**

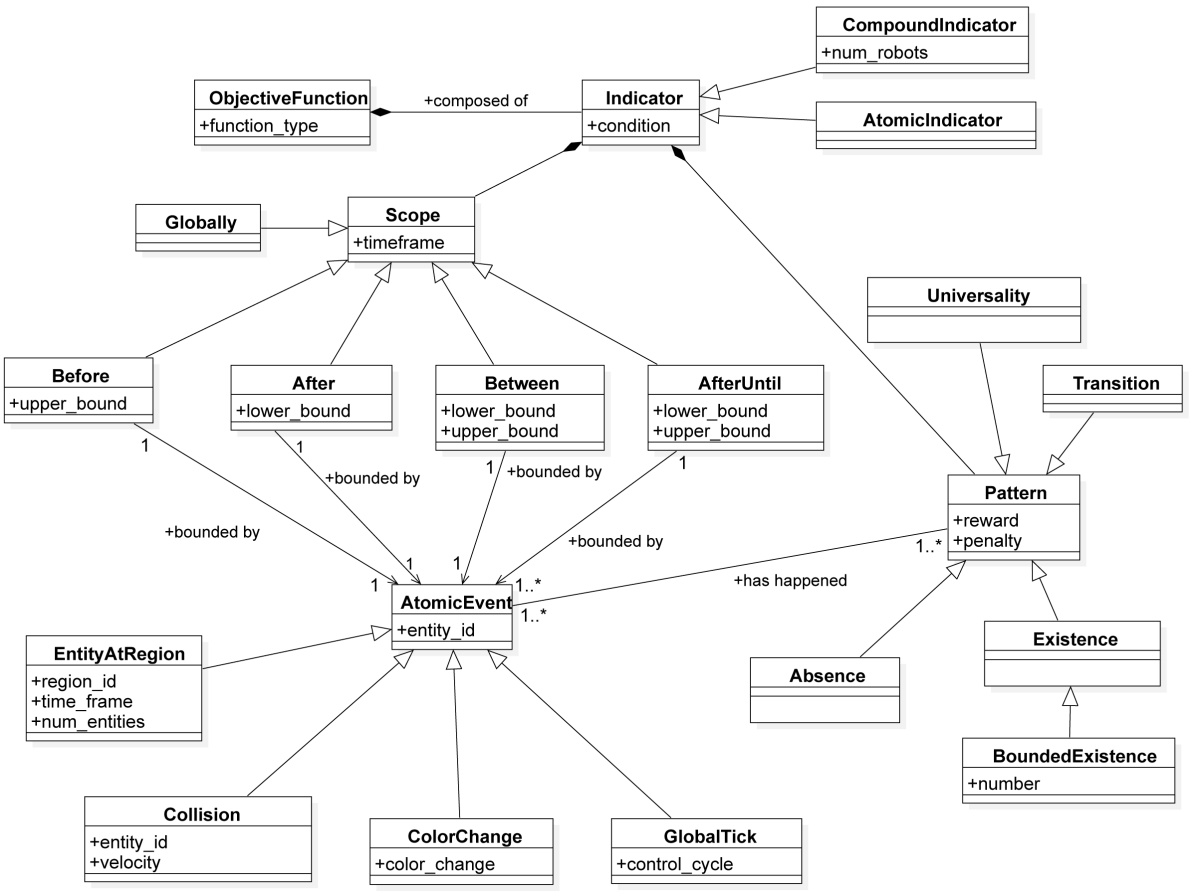


Figure 5. Constructs to specify the assessment of the mission objectives.

The environment where the mission is performed is defined through the *Environment* construct. It consists of an *Arena* that gives the context of the mission. The *EnvironmentElement* is an abstract metaclass that can be implemented through the classes of the various elements placed in the environment. The number and type of elements that can be used across missions might vary. The central concept in SML is the one of *Region*. The *Region* is an abstract class that can be instantiated as a specific geometric shape referring to coordinates in the environment. All environment constructs can be found in Figure 3.

The *Swarm* is a construct that is developed to specify the number and type of robots performing the mission, their initial position, and the set of low-level behaviours they can perform. The *Robot* construct represents a specific instance of the reference model of a robot.

Definition 4.2 (Reference Model). A reference model of a robot is a tuple $R = (T, A, P, V, Z)$ where:

- T is a set of actuators and sensors;
- A is a set of attributes;
- $P : T \rightarrow A$ is an assignment function that maps sensors/actuators T to the corresponding attributes A;
- V is a set of values that can be given to the attributes;
- $Z : A \rightarrow V$ is an assignment function for the attributes.

An example of a reference model is shown in Table 1.

The *Behaviour* is an abstract class that represents a low-level action that an individual robot can perform. A behaviour can be instantiated on a different level of abstraction and can be modular, as in our implementation. The modular structure of the Behavior construct of the language allows composition of atomic sub-behaviors into complex behaviours. For example, in `Chocolate`[1] we implemented an Obstacle Avoidance behavior as a sub-behaviour into five low-level behaviors. *Behavior* is associated with the concept of *MissionClass*, an abstract class that presents a set of missions. The association between the behavior and mission class relates to the corresponding mapping that defines which behaviors are suitable for a specific mission class to be performed. The *MissionClass* is a construct that provides a template for a set of missions that share similar behavior patterns. In this work, all implemented behaviors were used to create a swarm controller for each instantiated mission class. We instantiated three mission classes: aggregation, foraging and migration.

Table 1. Reference model RM 1.1[48]. Sensors and actuators of the extended version of the e-puck robot.

sensor/actuator	variables	values
proximity	$prox_i$, with $i \in \{0, ..., 7\}$	[0, 1]
light	$light_i$, with $i \in \{0, ..., 7\}$	[0, 1]
ground	$ground_i$, with $i \in \{0, ..., 2\}$	{*black, gray, white*}
range-and-bearing	n	{0, ..., 19}
	V_d	([0, 0.7]m,[0, 2π] radian)
wheels	v_l,v_r	[–0.12, 0.12]ms^{-1}

The third aspect of the language is the representation of the objective function that is used to measure the success of the mission execution (Figure 5). The *ObjectiveFunction* is a construct that consists of a set of *Indicators*. An *Indicator* is an abstract class that represents the smallest measurable unit of performance in a mission. It can take one of the two forms:

- *Atomic Indicator*: a construct that represents the smallest measurable unit of performance for an individual robot;
- *Compound Indicator*: a construct that represents the smallest measurable unit of performance for a set of robots.

The idea of the indicator is taken from Dwyer *et al.*[49] and Autili *et al.*[50]. Indicators are defined as a pattern in a scope—i.e., each indicator is represented through two constructs:

- *Pattern*: a construct used to measure the degree of success in a mission;
- *Scope*: a timeframe within the duration of the mission during which the pattern is quantified.

We identify the following scopes: *GLOBALLY*, *AFTER* the occurrence of an event, *BEFORE* the occurrence of an event, *BETWEEN* the occurrence of two events, or *AFTER* the occurrence of one event, and *UNTIL* the occurrence of an another one (*AFTER UNTIL*). *Atomic Event* is an abstract class that specifies the possible events that can happen during a mission. In our work, we identified four abstract events:

- *Color Change*: an event that is triggered when there is a color change in the mission entities;
- *Entity at Region*: an event that is triggered when a set of entities are in a specific region (e.g., robots stay in a certain region);
- *Collision*: an event that is triggered when two entities in the mission collide. For example, it might be a collision between two robots, a collision between a robot and an obstacle, a collision between a robot and a wall in the arena etc.;
- *Global Tick*: an event that is triggered at every step of the mission execution.

As mentioned before, patterns are mission-agnostic concepts that are used to quantify the success of a mission. We identify a set of patterns that quantifies the appearance of a specific mission concept during execution. In the following, we describe three patterns of SML:

- *Absence*: a pattern that quantifies the absence of an event—e.g., a robot is not in a specific region, a robot does not perceive light, etc.;
- *Existence*: a pattern that quantifies the existence of an event—e.g., a robot is in a specific region, a robot broadcasts a message, etc.;
- *Universality*: a pattern that quantifies the universality of an event—something that should always occur.
- *Transition*: a pattern that quantifies the transition between two events—e.g., a robot moves from one region to another.

Bounded existence is a sub-pattern of an existence which quantifies the existence of an event only in certain bounds (something should occur at most *n* times).

These constructs are abstract and need to be instantiated to be realized in SML. More details about the instantiation of these constructs is given in Section 5.

5 Implementation of SML as a textual domain specific language (DSL)

The SML Language conforms to the SML metamodel discussed in Section 4.2. We implemented SML as a textual domain specific language (DSL) to enable non-technical end users

(users that do not necessarily have technical knowledge in swarm robotics) and swarm designers to specify missions using structured English grammar. The development of SML is based on the following concepts:

- **Extensibility**:Control software designers should be able to add new constructs for new classes of missions. One of the most important goals in the development of SML was to provide an easy way for designers to add new language constructs. This enables reusability of the language across projects, missions, and research groups.
- **Variability**: Control software designers should be able to define missions in a variety of ways. Variability plays an important role in the definition of swarm missions. There are many examples in the swarm robotics literature where the position of the robots and of other objects in the environment is defined in a probabilistic way. There are many other examples where the position of the objects is provided in a deterministic way. Providing a rich interface using variability points was the second most important goal in the design of SML.
- **Usability**: Non-technical end users (users that do not necessarily have technical knowledge in swarm robotics) should be able to specify missions.
- **Generality**: Control software designers should be able to specify classes of missions that are non-trivial. Having constructs that are generic and independent from the functional behaviour of the robots is extremely relevant for managing the complexity of the missions to be performed.

5.1 SML syntax

To implement the language we used Xtext[51]. Xtext is a framework for developing domain-specific languages. It provides a full infrastructure, including parser, linker, typechecker, and compiler. The current implementation of SML is focused on realizing abstract concepts through a set of elements that are necessary to define a mission. We will extend the set of these elements in a future work, which will increase the application domain of SML. It is important to note that the current implementation of SML in Xtext includes variable name resolution, parse error visualisation, and syntax highlighting. A screenshot of the SML editor is shown in Figure 6.

Snippet code of the grammar that enables mission description and swarm configuration is presented in Listing 1. In the implementation, the model of the language is realised through three high level concepts: *Environment* specification, *Swarm configuration*, and *Mission objective* specification. The current implementation of the language supports three different types of mission: *aggregation*, *foraging*, and *migration*. In aggregation, the robot swarm must group (Figure 8). In foraging, the swarm must collect items from the environment and brings them to the nest (Figure 9); while in migration, the swarm must move from one initial location to another one (Figure 10). The current implementation of the language is extensible in the following directions: (i) it allows new classes of missions to be defined through the *Task* construct and (ii) it allows new types of robots to be defined through the *Robot* construct. The current implementation includes support for e-puck[21] and s-bot[52]. Each of these robots has different sensors and actuators with different attributes and values.

Snippet code of the grammar that enables the specification of mission objectives is presented in Listing 2. In the current implementation of the language, *Occurence* is the only pattern that we used to quantify the success of the mission. Through this pattern, we can specify a variety of missions that are non-trivial. We satisfy one of the main aspects of our language mentioned in Section 5—generality. Patterns enable generality by separating the concepts on how the success of the

mission1.sml

```
The arena is a Circle C with center at point 0.0, 0.0 and radius 3.25 m
The arena contains the following elements:
An obstacle as a Circle obs with  center at point 0.3, 0.5 and radius 0.7 m
A light source emitting a red light with intensity 3.0 is placed at point 1.3, 1.3, 0.4
A Circle c1 with center at point 0.5, 0.5 and radius 0.8 m
A Circle c2 with center at point 1.7, 1.7 and radius 0.3 m
A Rectangle d with reference point 2.4, 3.3 and length 0.25 m ,width 0.8 m ,height 0.3 m
There are 4 obstacles distributed with a Gaussian distribution in c1
The robot swarm consists of less than 12 e-puck distributed with a Uniform distribution in C
The mission of the robots is to aggregate on c2
The total time of the mission is less than 300 seconds
The performance measure is defined in terms of an objective function to be maximized.
The objective function is computed as follows:
At any point in time, each robot gets a penalty 0.05 if it is on c1
At the end of the mission, each robot gets a reward 30.0 if it is on d
```

Figure 6. Screenshot of the Swarm Mission Language (SML) editor in Xtext.

```
grammar org.xtext.example.sml.Sml with org.eclipse.xtext.common.Terminals
generate sml "http://www.xtext.org/example/sml/Sml"

Model:
   (env=Environment) (sw=Swarmconf) (ms=MissionSpecification);

Swarmconf:
  'The' 'robot' 'swarm' 'consists' 'of' x=Range r=Robot pr=ProbabilisticDecription;

Robot:
  'e-puck' | 'foot-boot';

MissionSpecification: m=MissionDefinition mt=MissionTime ob=MissionObjective;

MissionTime: 'The' 'total' 'time' 'of' 'the' 'mission' 'is' T=Range m=Metric;

MissionDefinition: 'The' 'mission' 'of' 'the' 'robots' 'is' 'to' t=MissionClass;

MissionClass:
    Aggregate | Migration | Foraging;

Aggregate:
  'aggregate' 'on' r+=[Region] (',' r+=[Region])*;

Migration:
  'aggregate' 'on' r+=[Region] (',' r+=[Region])* 'while' 'avoiding' t+=[Region] (',' t+=[
    ↪ Region])*;

Foraging:
  'collect' 'food' 'from' source+=[Region] (',' source+=[Region])* 'and' 'bring' 'it' '
    ↪ back' 'to' nest=[Region] (',' source+=[Region])*;

Time:
  x=Range | StartMission | EndMission;

Metric:
  'seconds' | 'steps' | 'm';
```

Listing 1. **Language grammar for defining the mission and the swarm configuration.**

```
MissionObjective:
  'The' 'performance' 'measure' 'is' 'defined' 'in' 'terms' 'of' 'an' 'objective' '
    ↪ function' 'to' 'be' Type=('maximized.' | 'minimized.')
  'The' 'objective' 'function' 'is' 'computed' 'as' 'follows:'  (in+=Indicator)*;

Indicator:
  sp=Scope  oc= (AtomicIndicator | CompoundIndicator) ;

AtomicIndicator:
  'each' 'robot' oc=Pattern ;

CompoundIndicator:
  'the' 'swarm' oc=Pattern ;

Scope:
  sp = AnytimePoint | StartMission | EndMission | Condition ;

AtomicEvent:
  'Walls change color' ;

Pattern:
  Reward | Penalty ;

Penalty:
  'gets' 'a' 'penalty'  k=Double c=Condition;

Reward:
  'gets' 'a' 'reward' k=Double c=Condition;

Condition returns String:
  {existence_absence1}('if' | 'If') 'it' 'is' inside=('on' | 'outside')  r=[Region] |
  {existence_absence2}('if' | 'If')'there' 'are' 'more' 'than' n=INT 'robots' inside=('on'
    ↪  | 'outside')   r=[Region] (",")*  |
  {transition} ('if' | 'If') 'it' 'goes' 'from' r=[Region] 'into' nest=[Region] |
  {universality} 'always';

StartMission: ('Beginning'|'beginning') 'of' 'the' 'mission' (',')*;

EndMission: ('at'|'At') 'the' 'end' 'of' 'the' 'mission' (',')*;

AnyTimePoint: ('at'|'At') 'any' 'point' 'in' 'time' (',')*;
```

Listing 2. **Language grammar for defining mission objectives.**

mission is measured from the functional behavior of the robots. As we identify additional patterns in robot swarm missions, we plan to add them as additional constructs in SML. SML is highly extensible, which allows new pattern constructs to be added at a later stage of the development.

Patterns need to be instantiated. *Occurence* is instantiated through the concepts of *Reward* and *Penalty*. *Condition* is another construct that plays an important role in the definition of the language. We use it to determine the situations under which a score is assigned to the swarm that executes the mission. This construct can be extended by identifying other situations that are relevant in swarm robotics. In the current implementation, a score is assigned only through the concept of a region: whether a robot, a set of robots, or objects are in a particular part of the environment.

Snippet code of the grammar that enables the specification of the environment elements is presented in Listing 3. We implemented a set of environmental elements that can be directly used in missions. At the moment of writing, we have implemented four environmental elements: *Wall*, *Light Source*, *Floor Patch* and *Obstacle*. Each of these elements has been defined through different attributes. The extensibility of SML allows designers to easily add new environmental constructs if they need them.

To demonstrate the variability of SML, we defined a set of variability points for the different constructs in SML. These variability points allow designers to formulate mission concepts in a variety of ways. For example, the position of robots, objects, and obstacles in the environment can be specified either in a deterministic or a stochastic way. In Figure 6, we present a simple mission specification. An obstacle is defined in a deterministic way with the following statement:

The arena contains the following elements:
An obstacle as a Circle obs with center at point 0.3, 0.5 and radius 0.7 m.

An example of obstacle description in a stochastic way is presented through the following statement:

```
Environment:
  a=Arena ('The' 'arena' 'contains' 'the' 'following' 'elements' ':')? (environment+=
    ↪ EnvironmentElements)*;

Arena:
  'The' 'arena' 'is' 'a' s=Region 'surrounded' 'by' 'walls' '.';

EnvironmentElements:
  EnvironmentElement | ElementDescription ;

EnvironmentElement:
  Obstacle | Light | Object | Patch  | 'A' Region ;

ElementDescription:
  el='There' 'are'  x=Range obj=Element r=ProbabilisticDecription;

Element:
  'objects' | 'obstacles' | 'light' 'sources' | 'patch';

ProbabilisticDecription:
   'distributed' 'with' 'a' dis=Distribution 'distribution' 'in' k=[Region] ;

Distribution:
  'Uniform' |'Gaussian' | 'Constant' ;

Region:
  (colors=Color)? (shape2d=Shape2D) (name=ID)  region=RegionDefinition;

RegionDefinition:
  Definition1 | Definition2 | Definition3;

Definition1:
  ('with' referencepoint=Position)? ('and' dimensions=Dimension)? ;

Definition2:
  'defined' 'through' 'the' 'following' 'Vertices' ':' (point+=Position) (';' point+=
    ↪ Position)* ;

Definition3:
  'where' ax=Axis 'is' r=Range ;

Dimension:
  'radius' r=Double m1=Metric  ('and' 'height' h=Double m2=Metric)? |  'length' l=Double
    ↪ m3=Metric ',width' w=Double m4=Metric ('and' 'height' h=Double m5=Metric)?  | 'side'
    ↪  s=Double m6=Metric ;

Range:
  Upperbound | Lowerbound | Interval | LowerorEqualbound | UpperorEqualbound |
    ↪ ConstantSize;
```

Listing 3. Language grammar for defining the environment.

The arena contains the following elements:

A Circle c1 with center at point 0.5, 0.5 and radius 0.8 m.

There are 4 obstacles distributed with a Gaussian distribution in c1.

This aspect of the language provides a rich platform for the definition of the various elements in the environment. *Region* is the basic language construct we use to define the various elements. In the current implementation of SML, we have three variability points on how we specify regions:

- through one reference point and set of dimensions (Definition1: Figure 9);
- only through a set of reference points (Definition2: Figure 9);
- through the global coordinate system (Definition3: Figure 9).

We made the assumption that a global coordinate system exists and its origin is positioned in the center of the arena. This coordinate system is only used to specify the position of elements in the environment. It is important to note that robots are unable to utilize this coordinate system to position themselves or place objects in the arena because usage of a global coordinate system to create a certain robot behaviour goes against the main principles of swarm robotics.

5.2 Model-to-model transformation from SML to ARGoS XML files

ARGoS is a multi-engine simulator for swarm robotics. ARGoS has two main components that need to be defined in order to specify a mission: the XML configuration file and the so-called loop functions. Using the ARGoS XML file, users can specify the simulated space. ARGoS provides a way to specify several entity types. Each entity type stores information about a specific aspect of the simulation. It includes the position and the orientation of each object in the environment such as obstacles, light sources, boxes, and robots. The file is highly customizable and extendable—new entity types can be easily added and new features of the entities can be easily adapted and adjusted.

The loop functions are user-defined functions that are executed in strategic points of the simulation loop. Developers can customize the initialization and the end of an experiment, and add custom functions to be executed before and/or after each simulation step. Loop functions allow one to access and modify the entire state of the simulation. In particular, loop functions are a convenient way for computing relevant performance metrics used to measure the success of a mission.

Using model-to-model transformation techniques from the mission specification defined in SML, we automatically generate the XML file that is used by ARGoS to describe the simulation space and the loop functions that are used by ARGoS to run experiments. The mission models that are used by ARGoS are represented in Figure 7.

The Simulated Environment and the Controller are specified in the .xml file. A fragment of the generated XML configuration file is shown in Listing 4. It consists of five parts: experiment configuration details, definition of loop functions, specification of controllers, specification of the environment (the arena) and e-puck specification and distribution in the environment.

The loop functions are defined in a C++ file. We describe the main loop functions that are generated from the SML specification (fragment of the generated C++ file is shown in Listing 5). `Init` is a function that is used to instantiate all mission elements. We use it to create all mission entities, including the robots and the environment elements. `PostStep` is a function that is executed after each simulation steps. In this function, we iterate through the robot swarms and perform calculations, based on the definition of the objective function. In our scenarios, we used the concepts of reward and penalty to increase or decrease the value of the variable score which is an information on how well the robot swarm is performing the mission. `GetSwarmPosition()`

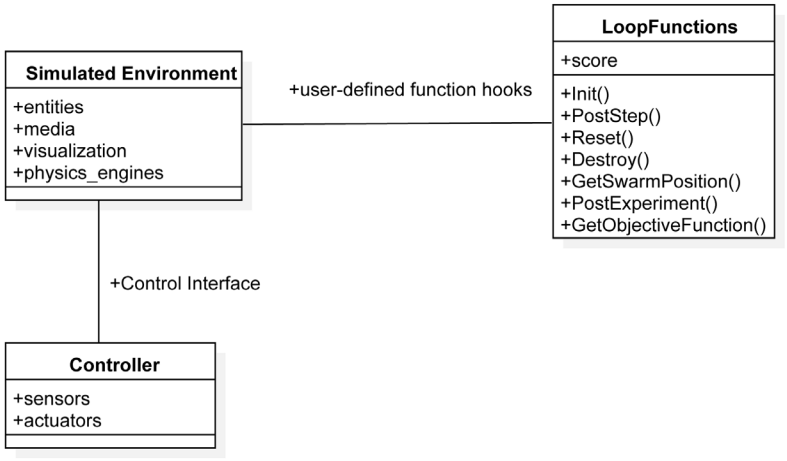


Figure 7. ARGoS architectural model.

```
<?xml version="1.0" ?>
<argos-configuration>
  <framework>
    <!--EXPERIMENT DETAILS-->
    <experiment length="360"
                ticks_per_second="10" random_seed="0"/>
  </framework>

  <!-- * Loop functions * -->
  <loop_functions library="mission1/libchocolate_mission1_loopfunc.dylib" label="
    ↪ chocolate_mission1_loop_functions">
   <params dist_radius="1.2" number_robots="15" m_min_robots="15"/>
  </loop_functions>

  <!-- * Controllers * -->
  <controllers>
    <automode_controller id="automode"
                         library="libautomode.dylib">
      <actuators>
        ...
      </actuators>
      <sensors>
        ...
      </sensors>
    </automode_controller>
  </controllers>

  <!-- * Arena * -->
  <arena size="10, 10, 1" center="0,0,0">
    <physics_engines>
      <dynamics2d id="dyn2d" />
    </physics_engines>

    ...
  <!-- * E-Puck * -->
 <e-puck id="epuck_01_30">
    <body orientation="0,0,0" position="0,-0.4,0" />
    <controller config="automode" />
  </e-puck>
    ...
  <e-puck id="epuck_15_32">
    <body orientation="0,0,0" position="0.4,-0.6,0" />
    <controller config="automode" />
  </e-puck>


   <distribute>
      <position method="uniform" min="-0.6,-0.6,0" max="0.6,0.6,0" />
      <orientation method="uniform" min="0,0,0" max="90,0,0" />
      <entity quantity="5" max_trials="100">
        <box id="bsd" size="0.28,0.046,0.059"  mass="1.8" movable="true" />
      </entity>
  </distribute>

  </arena>
 ...
</argos-configuration>
```

Listing 4. A fragment of the generated XML configuration file for Aggregation.

is a function that defines the region where the robots are placed at the start of the mission. Here, we position each of the robots in the environment using some of the preexisting algorithms for placement. We only specify the boundaries of the region that is used for initial placement. `Reset()` is a function that is used by `Chocolate` to reset all variables before initializing a new run. `PostExperiment()` is a function that is executed after the mission is over. In our case, this function reports only the value of the ObjectiveFunction. `GetObjectiveFunction()` is a function that is used to return the score of an experimental

```
#include "mission.h"

ChocolateMission1LoopFunction::ChocolateMission1LoopFunction() {
  m_cCoordBlackSpot = CVector2(0.0,0.0);
  m_fObjectiveFunction = 0;
  m_unNumberEdges=6;
  ... }

ChocolateMission1LoopFunction::ChocolateMission1LoopFunction(const
    ↪ ChocolateMission1LoopFunction& orig) {}

ChocolateMission1LoopFunction::~ChocolateMission1LoopFunction() {}

void ChocolateMission1LoopFunction::Destroy() {}

Real ChocolateMission1LoopFunction::GetObjectiveFunction() {return m_fObjectiveFunction;}

argos::CColor ChocolateMission1LoopFunction::GetFloorColor(const argos::CVector2&
    ↪ c_position_on_plane){
  CVector2 vCurrentPoint(c_position_on_plane.GetX(), c_position_on_plane.GetY());
  Real d = (m_cCoordBlackSpot - vCurrentPoint).Length();
  if (d <= m_fRadius) {
    return CColor::BLACK;
  }
  return CColor::GRAY50;
}

void ChocolateMission1LoopFunction::PostStep() {
  CSpace::TMapPerType& tEpuckMap = GetSpace().GetEntitiesByType("epuck");
  ...
  for (CSpace::TMapPerType::iterator it = tEpuckMap.begin(); it != tEpuckMap.end(); ++it)
    ↪ {
    CEPuckEntity* pcEpuck = any_cast<CEPuckEntity*>(it->second);
    Real fDistanceSpot = (m_cCoordBlackSpot - cEpuckPosition).Length();
    if (fDistanceSpot <= m_fRadius) {
      ...
    }}}

void ChocolateMission1LoopFunction::PostExperiment() { LOG << "Score_objective:" <<
    ↪ m_fObjectiveFunction << std::endl; }

void ChocolateMission1LoopFunction::Init(TConfigurationNode& t_tree) {
  CoreLoopFunctions::Init(t_tree);
  CRadians fAngle=(2*CRadians::PI)/m_unNumberEdges;
  for(UInt32 i = 0; i < m_unNumberEdges; ++i) {
      ...
      pcWall = new CBoxEntity(id.str().c_str(), CVector3((fRadius * Cos(fAngle * i)),(
    ↪ fRadius * Sin(fAngle * i)), 0), CQuaternion().FromEulerAngles(-CRadians::PI +(fAngle
    ↪  * i),CRadians::ZERO,CRadians::ZERO), false, CVector3(0.01,sidesize,0.08), 0.2);
      AddEntity(*pcWall);
    }
     MoveObstacles();
     ...
   }

void ChocolateMission1LoopFunction::Reset(){
  m_fObjectiveFunction = 0;
  CoreLoopFunctions::Reset();
}

CVector3 ChocolateMission1LoopFunction::GetRandomPosition() {
...
       a = m_pcRng->Uniform(CRange<Real>(0.0f, 1.0f));  b = m_pcRng->Uniform(CRange<Real
    ↪ >(0.0f, 1.0f));
       ...
      fPosX = b * InnerRadious() * cos(2 * CRadians::PI.GetValue() * (a/b));  fPosY = b *
    ↪ InnerRadious() * sin(2 * CRadians::PI.GetValue() * (a/b));
...
  return CVector3(fPosX, fPosY, 0);  //position the robot in the environment
}

REGISTER_LOOP_FUNCTIONS(ChocolateMission1LoopFunction, "mission1_loop_functions");
```

Listing 5. A fragment of the generated C++ file.

run for a mission. If the score is high, `Chocolate` considers that the robot swarm performs the mission well.

5.3 Validation of the SML implementation

The internal consistency of SML is critical for a fully integrated design process. To validate the internal consistency of the language we followed these steps:

1. **Instantiate a mission that contains all language constructs.** We created an instance of a mission that contains (almost) all language constructs. Furthermore, we specified smaller instances that instantiate the elements that could not be included in a single large instance due to exclusion constraints in the language.

2. **Validate the language in Xtext**. We created a set of validation rules that each mission instance should comply to. First, we created validation rules that ensure the structure of the mission instance. Each instance of a mission specification must contain a specification of the arena, specification of the environment, specification of the robot swarm and the objective that is measured. If any of these elements are missing, an error to the user is reported. Second, each physical element must be fully specified in terms of its location. The robot swarm, obstacles, lights, patches and all other environment elements must be located in a region of the arena. Third, all environment elements in the environment must be unambiguously specified in terms of their dimensions. As SML supports different type of specification variations (through a set of vertices, through side size, through a combination of a side size and a size of a diagonal), we created rules that verify that each specification is unambiguous representation of a physical element i.e. does not have any missing information (e.g., a designer that specifies a circle with a radius, must also specify the center of the circle, otherwise the information is incomplete.) These rules help the designer to double check semantic inconsistencies between different environmental elements.

3. **Check consistency of the generated model files.** To validate the consistency of the generated files, we performed the following analysis:

   - XML configuration consistency: For each system specification, we loaded the generated configuration file in ARGoS to confirm their internal consistency. Each configuration was successfully loaded by ARGoS. If there is an error in the configuration file, ARGoS is not able to load the file or if it loads it, some elements in the visualization will be missing.

   - Use the `Chocolate` design process to check for coherence and consistency of the loop functions. For each generated loop function file, we run `Chocolate` to confirm the validity. `Chocolate` is not able to run if the XML configuration is not properly specified or if the required functions Init(), PostStep(), GetSwarmPosition(), PostExperiment() are not instantiated (Figure 7). If the loop functions contain a syntactical error, `Chocolate` throws an exception.

6 Demonstration

Setup

To demonstrate the generality of SML, we present three different missions for which code was automatically generated and ported to real robots (see *Underlying data*[53]). In the demonstration, we use e-puck robots[21] equipped with several extension boards[54], including the range-and-bearing board[55]. We specify three missions in the SML editor. When we save a mission specification in the editor, the code generation process starts and generates an ARGoS .xml file and a .cpp file that contains the loop-functions that measure the performances of the swarm on the mission at hand. These files are used as artifacts for `Chocolate`[1]. `Chocolate` generates control software for the reference model of the e-puck reported in Table 1[48].

Results

The following examples demonstrate the applicability of SML. For each of the three missions, `Chocolate` is executed 10 times to obtain 10 instances of control software. Each design process relies on a maximum of 200000 simulated runs. The simulator adopted in the study is ARGoS3, beta 48. We evaluated each instance of control software obtained by `Chocolate` in simulation. The best one was chosen and evaluated on real robots, as well.

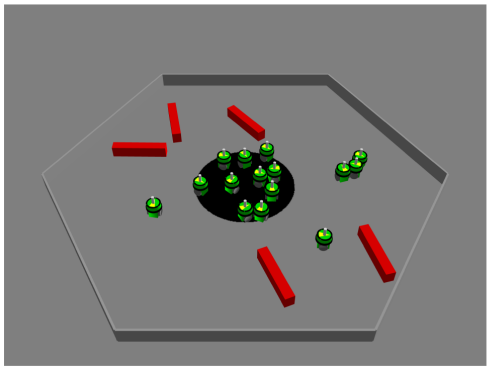
In Listing 6, we specify *Aggregation on one spot*. First, we identify the shape of the arena, which is a *Hexagon* with sides of 1 m. Then, we declare various environmental elements and their location in a global coordinate system with an origin at the center of the arena. In this context, at line 4, we specify a circular black patch placed in the center of the arena with radius of 25 cm. At line 5, we specify the number and type of robots that will be used in the mission. We use 15 e-pucks distributed across the whole arena. At line 6, we define the details of the mission. Here, we explicitly state the mission goal and connect it to environmental elements. In this mission, the robots must aggregate on the black patch $c1$. At line 7, we specify that the mission completion time is 360 seconds.

At the end, we finish our specification with the definition of the objective function (line 9–10)—we state which concepts of the mission should be measured: we reward each robot that position itself on the black patch $c1$. Using the aforementioned mission specification, we generate the control software shown in Figure 8(c). The performance of the control software is evaluated in simulation—Figure 8(a) and on real robots—Figure 8(b).

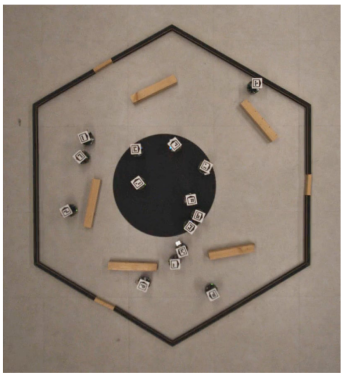
In Listing 7, we specify a *Foraging* mission with two sources and one nest. The arena is a dodecagon with a side of 0.66 m (line 1). We declare 4 environmental elements (line 2 – 6). For each environmental element, we specify its position in a global coordinate system with origin at the center of the arena. We declare a light source with intensity 5.0, two food sources

```
The arena is a gray Hexagon h1 with center at point 0.0, 0.0 and side 1 m surrounded by
    ↪ walls.
The arena contains the following elements:
There are 5 obstacles distributed with a Gaussian distribution in h1
A black patch as a Circle c1 with center at point 0.0, 0.0 and radius 0.25 m
The robot swarm consists of 15 e-puck distributed with a Constant distribution in h1
The mission of the robots is to aggregate on c1
The total time of the mission is 360 seconds
The performance measure is defined in terms of an objective function to be maximized.
The objective function is computed as follows:
At any point in time, each robot gets a reward 3 if it is on c1
```

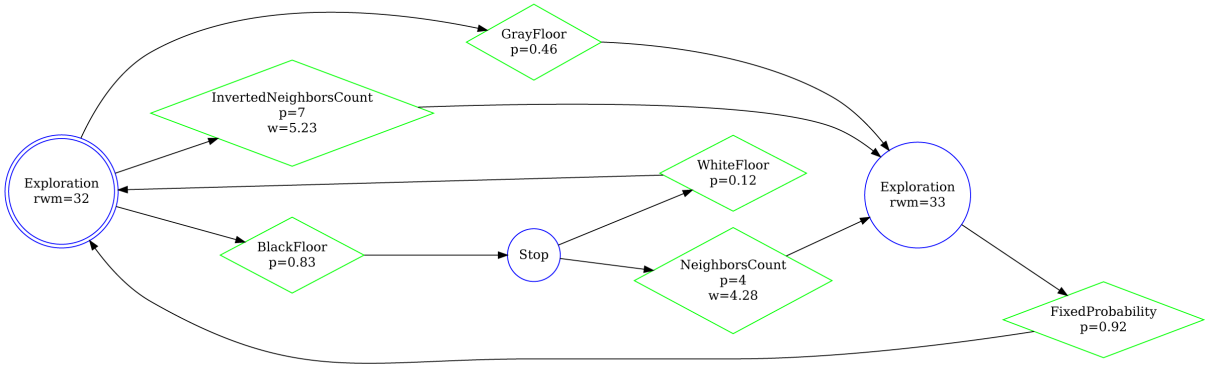
Listing 6. Mission specification.



(a) Simulated arena.



(b) Real arena.



(c) Control software evaluated in simulation and reality.

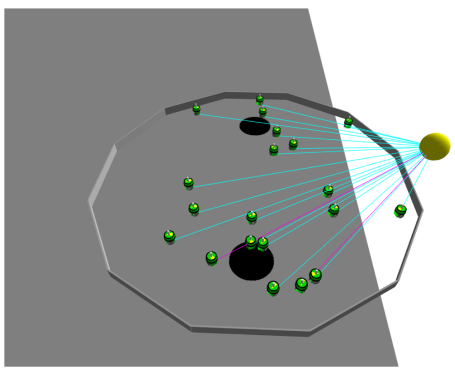
Figure 8. Aggregation on one spot.

located on $b1$ and $b2$ and a nest $w1$. At line 7, we state that 20 e-puck robots will be used in the mission. At line 8, we state the explicit details about the mission—connecting the mission goals (collect food, bring food) with their corresponding environmental elements (Circle b1, Circle b2, Region w1). At line 9, we define that the mission completion time is 270 seconds. At the end, we declare the specific elements of the objective function—we quantify when an individual robot arrives in one of the black patches $b1$ or $b2$ and gets back to its nest $w1$. Using the aforementioned mission specification, we generate the control software shown in Figure 9(c). The performance of the control software is evaluated in simulation—Figure 9(a) and on real robots—Figure 9(b).

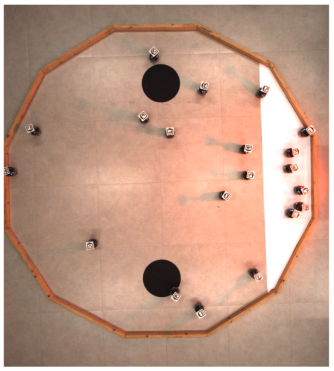
In Listing 8, we specify *Migration with an obstacle*. The arena of the this mission is a square with a side of 1.5 m (line 1). We declare four environmental elements (line 2 - 6). For each environmental element, we specify its position in a global coordinate system with origin at the center of the arena. We declare a light source with intensity 3.0, a triangular white patch $t1$—an area to which the swarm must move, a large obstacle r_0 b in the center of the arena and initial location of the swarm

```
The arena is a gray Dodecagon d1 with center at point 0.0, 0.0 and side 0.66 m surrounded
    ↪ by walls.
The arena contains the following elements:
A light source emitting a red light with intensity 5.0 is placed at point 0.0, -1.3, 0.4
A black patch as a Circle b1 with center at point 0.75, 0.0 and radius 0.15 m
A black patch as a Circle b2 with center at point -0.75, 0.0 and radius 0.15 m
A white patch as a Region w1 where y is less than -0.8
The robot swarm consists of 20 e-puck distributed with a Uniform distribution in w1
The mission of the robots is to collect food from b1, b2 and bring it back to w1
The total time of the mission is 270 seconds
The performance measure is defined in terms of an objective function to be maximized.
The objective function is computed as follows:
At any point in time, each robot gets a reward 1.0 if it goes from b1 into w1
At any point in time, each robot gets a reward 1.0 if it goes from b2 into w1
```

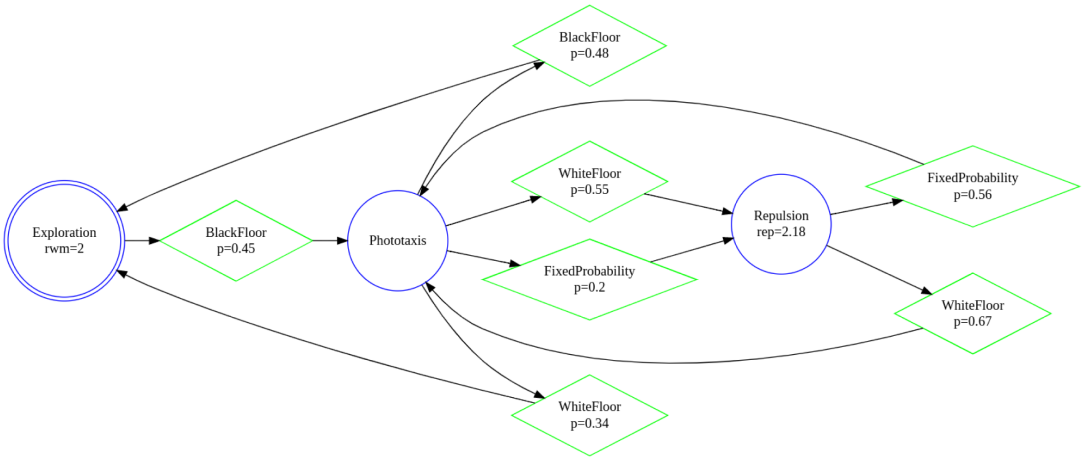
Listing 7. Mission specification.



(a) Simulated arena.



(b) Real arena.



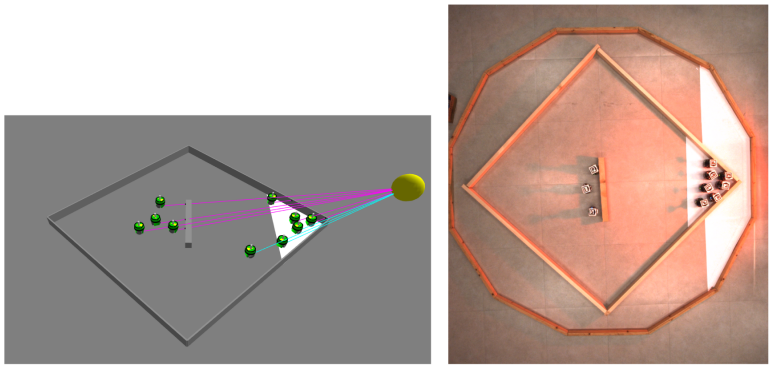
(c) Control software evaluated in simulation and reality.

Figure 9. Foraging.

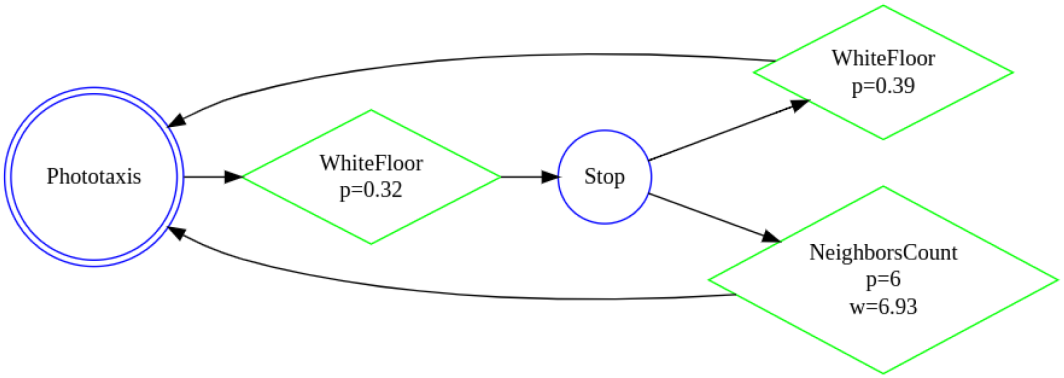
r t. At line 7, we state that 10 e-puck robots will be used in the mission. At line 8, we state the explicit details about the mission—connecting the mission goals (migration to an area) with their corresponding environmental elements (Triangle t1). At line 9, we define that the mission completion time is 300 seconds. In the end, we declare the specific elements of the objective function (line 9–13): we reward the swarm if more than 5 robots are at the target location. Moreover, every time the swarm receives a reward, we add a small penalty for each robot outside the target area. Using the aforementioned mission specification, we generate the control software shown in Figure 10(c). The performance of the control software is evaluated in simulation—Figure 10(a) and on real robots—Figure 10(b).

```
The arena is a gray Square sq1 with center at point 0.0, 0.0 and side 1.5 m surrounded by
    ↪ walls.
The arena contains the following elements:
A light source emitting a red light with intensity 3.0 is placed at point 0.0, -1.3, 0.4
An obstacle as a Rectangle r_ob with center at point 0.0, 0.0 and length 0.046 m ,width
    ↪ 0.5 m, height 0.059 m
A white patch as a Triangle t1 defined through the following Vertices:  point 0.4125,
    ↪ -0.66; point -0.4125, -0.66; point 0.0, -1.05
A Square rt defined through the following Vertices: point -0.3, 0.4; point 0.3, 0.4; point
    ↪  -0.3, 1.0; point 0.3, 1.0
The robot swarm consists of 10 e-puck distributed with a Uniform distribution in rt
The mission of the robots is to aggregate on t1 while avoiding r_ob
The total time of the mission is less than 300 seconds
The performance measure is defined in terms of an objective function to be maximized.
The objective function is computed as follows:
At any point in time, the swarm gets a reward 1.0 if there are more than 5 robots on t1
If there are more than 5 robots on t1, each robot  gets a penalty 0.05 if it is outside t1
```

Listing 8. Mission specification.



(a) Simulated arena. (b) Real arena.



(c) Control software evaluated in simulation and in reality.

Figure 10. Migration with an obstacle.

The generated code, the data collected in simulation and in reality, and the videos of the behavior by the swarm of physical robots are available online as supplementary material[53,56].

7 Conclusions

We presented a first instance of a fully automatic and integrated process for the design of collective behaviors for robot swarms. The novel contribution in this paper is the definition of a specification language and an automatic approach that transforms a formal mission specification into a configuration setup needed to run the design by optimization of control software using `Chocolate`. We introduced SML, a textual language to specify a mission that can be accomplished by a robot swarm. From the mission specification, we

automatically generated code and deployed it on real robots. We demonstrated the applicability of SML on three missions.

The current implementation of SML supports the specification of missions in which rewards and penalties can be expressed with reference to regions: depending on whether robots or objects are in a particular part of the environment at a given moment in time. Missions like aggregation, foraging and collective exploration can be modeled through the concept of a region. However, collective decision making or coordinate navigation missions discussed in Section 4.1 (e.g., flocking, consensus achievement, and task allocation) cannot be modeled using the current implementation of SML.

Future work will develop along three lines. First, we will work on the language extensibility. We plan to extend the set of constructs to support the specification of new classes of missions. We will introduce new indicators and patterns to measure success of new mission types. Moreover, we plan to enrich the language by introducing quality attributes as part of the mission specification. Safety, performance, and energy-efficiency are just a few important quality attributes that should be modeled as separate language constructs in the definition of missions for robot swarms. Introducing quality attributes in the mission specification process will contribute towards a better definition of mission objectives. Moreover, we will work on a formal language validation. After employing new constructs that should support new classes of missions, we plan to perform model validation to understand if there are constructs that are not covered by the current implementation of SML. We plan to create a mission generator and generate a set of mission instances that can help us to analyze the coverage of the language. This will help us to increase the number of mission classes that can be captured by the language implementation.

Second, we plan on developing our contribution towards a systematic methodology for designing robot swarms. We are focused on a full-fledged linear automatic design process as we were making our first steps in closing all the gaps in the fully automatic design process: a swarm designer is able to specify robot swarm requirements, but also include details about the design setup to be used to obtain the desired robot swarm. This includes information on which automatic design method use, how many simulation runs to perform, and information on the target environment. In this contribution, these details were manually specified as part of the automatic design method (we considered only Chocolate and a fixed design budget of 200K simulation runs). Moreover, to obtain a fully systematic integrated design process, swarm designers need to be able to select a set of predefined individual behaviours that can be used by the automatic design method to generate the control software. In this work, six behaviours were considered and they were selected as part of the automatic design method. Swarm designers need to be able to specify these details in the early phases of the design process, together with the definition of the system requirements. After obtaining a full-fledged linear automatic design process, we plan to move to an iterative design process where there is a feedback loop between design and testing before deployment of the control software in the target environment.

Third, we will focus on the usability of our integrated design process by investigating two research directions: i) We will perform an extensive user study to investigate SML's usability. To obtain deeper insights on how easily swarm designers can use SML for swarm missions, we will define a rigour protocol for evaluation based on objective qualitative and quantitative metrics that should demonstrate the usefulness of our domain-specific language; ii) We will develop a graphical user interface for non-expert users. The aim of the graphical interface is defining a process for non-expert users that is user friendly and less prone to input errors.

Data availability

Underlying data

Zenodo: Integrated automatic design process for robot swarms. https://zenodo.org/record/5184720[53].

This project contains the following underlying data:

- SML (Swarm Mission Language) related files: (i) specification files used to create missions; (ii) generated files to be used by an optimization method
- AutoMoDe related files: (i) the log files running AutoMoDe - an optimization method that generates control software for different missions; (ii) generated control software
- Demonstration: (i) snapshots and videos of running the missions on real robots

Data are available under the terms of the Creative Commons Attribution 4.0 International license (CC-BY 4.0).

Software availability

ARGoS3-AutoMoDe for the implementation of AutoMoDe-Chocolate

- Source code available from: https://github.com/demiurge-project/ARGoS3-AutoMoDe
- Archived source code at time of publication: https://doi.org/10.5281/zenodo.4849541;
- License: MIT license.

Demiurge-epuck-dao for the reference model of the robots used by the AutoMoDe design method

- Source code available from: https://github.com/demiurge-project/demiurge-epuck-dao
- Archived source code at time of publication: https://doi.org/10.5281/zenodo.4849535
- License: MIT license.

Argos3-epuck for the e-puck robot ARGoS3 plugin

- Source code available from: https://github.com/demiurge-project/argos3-epuck
- Archived source code at time of publication: https://doi.org/10.5281/zenodo.4882714
- License: MIT license.

ARGoS3 for the ARGoS3 simulator

- Source code available from: https://github.com/ilpincy/argos3
- Archived source code at time of publication: https://doi.org/10.5281/zenodo.4889111
- License: Creative Commons Attribution 4.0 International license (CC-BY 4.0).

Irace for the Iterated F-race algorithm

- Software available from: https://cran.r-project.org/package=irace
- Archived source code at time of publication: https://doi.org/10.5281/zenodo.4888996
- License: Creative Commons Attribution 4.0 International license (CC-BY 4.0).

References

1. Francesca G, Brambilla M, Brutschy A, *et al.*: **AutoMoDe-Chocolate: automatic design of control software for robot swarms.** *Swarm Intell.* 2015; **9**(2/3): 125–152.
Publisher Full Text
2. Dorigo M, Birattari M, Brambilla M: **Swarm robotics.** *Scholarpedia.* 2014; **9**(1): 1463.
Publisher Full Text
3. Bozhinoski D, Birattari M: **Designing control software for robot swarms: Software engineering for the development of automatic design methods**. In: *ACM/IEEE 1st International Workshop on Robotics Software Engineering RoSE*. New York, ACM, 2018; 33–35.
Reference Source
4. Brambilla M, Ferrante E, Birattari M, *et al.*: **Swarm robotics: a review from the swarm engineering perspective.** *Swarm Intell.* 2013; **7**(1): 1–41.
Publisher Full Text
5. Quinn M, Smith L, Mayley G, *et al.*: **Evolving controllers for a homogeneous system of physical robots: structured cooperation with minimal sensors.** *Philos Trans A Math Phys Eng Sci.* 2003; **361**(1811): 2321–43.
PubMed Abstract | Publisher Full Text
6. Trianni V: **Evolutionary swarm robotics: evolving self-organising behaviours in groups of autonomous robots**. Springer, 2008; **108**.
Publisher Full Text
7. Francesca G, Brambilla M, Brutschy A, *et al.*: **AutoMoDe: a novel approach to the automatic design of control software for robot swarms.** *Swarm Intell.* 2014; **8**(2): 89–112.
Publisher Full Text
8. Birattari M, Ligot A, Hasselmann K: **Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms.** *Nat Mach Intell.* 2020; **2**(9): 494–499.
Publisher Full Text
9. Brugali D, Prassler E: **Software engineering for robotics.** *IEEE Robot Autom Mag.* 2009; **16**(1): 9–15.
Publisher Full Text
10. Di Ruscio D, Malavolta I, Pelliccione P: **A family of domain-specific languages for specifying civilian missions of multi-robot systems**. In: *First Workshop on Model-Driven Robot Software Engineering-MORSE*. York, UK, 2014; 13–26.
Reference Source
11. Bozhinoski D, Di Ruscio D, Malavolta I, *et al.*: **Flyaq: Enabling non-expert users to specify and generate missions of autonomous multicopters**. In: *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. San Diego, CA, USA, IEEE, 2015; 801–806.
Publisher Full Text
12. Bozhinoski D, Garlan D, Malavolta I, *et al.*: **Managing safety and mission completion via collective run-time adaptation.** *J Syst Archit.* 2019; **95**: 19–35.
Publisher Full Text
13. Hoos HH: **Programming by optimization.** *Commun ACM.* 2012; **55**(2): 70–80.
Publisher Full Text
14. Francesca G, Birattari M: **Automatic design of robot swarms: achievements and challenges.** *Front Robot AI.* 2016; **3**(29): 1–9.
Publisher Full Text
15. Birattari M, Ligot A, Bozhinoski D, *et al.*: **Automatic off-line design of robot swarms: a manifesto.** *Front Robot AI.* 2019; **6**: 59.
PubMed Abstract | Publisher Full Text | Free Full Text
16. Schmidt DC: **Model-driven engineering.** *IEEE Computer.* 2006; **39**(2): 25.
Reference Source
17. Schlegel C, Lotz A, Lutz M, *et al.*: **Model-driven software systems engineering in robotics: covering the complete life-cycle of a robot.** *Info Technol.* 2015; **57**(2): 85–98.
Publisher Full Text
18. Schlegel C, Haßler T, Lotz A, *et al.*: **Robotic software systems: From code-driven to model-driven designs**. In: *2009 International Conference on Advanced Robotics*. Munich, Germany, IEEE, 2009; 1–8.
Reference Source
19. Hasselmann K, Ligot A, Ruddick J, *et al.*: **Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms.** *Nat Commun.* 2021; **12**(1): 4345.
PubMed Abstract | Publisher Full Text | Free Full Text
20. Pinciroli C, Trianni V, O'Grady R, *et al.*: **ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems.** *Swarm Intell.* 2012; **6**(4): 271–295.
Publisher Full Text
21. Mondada F, Bonani M, Raemy X, *et al.*: **The e-puck, a robot designed for education in engineering**. In P. Gonçalves, P. Torres, and C. Alves, editors, *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*. Portugal, Instituto Politécnico de Castelo Branco, 2009; 59–65.
Reference Source
22. Nolfi S, Floreano D, Floreano DD: **Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines**. MIT press, 2000.
Reference Source
23. Nolfi S: **Behavioral and Cognitive Robotics: An Adaptive Perspective**. Institute of Cognitive Sciences and Technologies, National Research Council, CNR-ISTC, Roma, Italy, 2021.
Reference Source
24. Hauert S, Zufferey JC, Floreano D: **Reverse-engineering of artificially evolved controllers for swarms of robots**. In: *2009 IEEE Congress on Evolutionary Computation.* IEEE, 2009; 55–61.
Publisher Full Text
25. Ligot A, Birattari M: **Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms.** *Swarm Intell.* 2020; **14**(1): 1–24.
Publisher Full Text
26. Koos S, Mouret JB, Doncieux S: **The transferability approach: crossing the reality gap in evolutionary robotics.** *IEEE Trans Evol Comput.* 2013; **17**(1): 122–145.
Publisher Full Text
27. Haasdijk E, Bredeche N, Nolfi S, *et al.*: **Evolutionary robotics.** *Evol Intell.* 2014; **7**: 69–70.
Publisher Full Text
28. Birattari M, Ligot A, Francesca G: **AutoMoDe: a modular approach to the automatic off-line design and fine-tuning of control software for robot swarms**. In: Nelishia Pillay and Rong Qu, editors, *Automated Design of Machine Learning and Search Algorithms*. Nature, Cham, Switzerland, 2021; 73–90.
Publisher Full Text
29. Geman S, Bienenstock E, Doursat R: **Neural networks and the bias/variance dilemma.** *Neural Comput.* 1992; **4**: 1–58.
Reference Source
30. Birattari M, Stützle T, Paquete L, *et al.*: **A racing algorithm for configuring metaheuristics**. In: W.B. Langdon and *et al.*, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO*. San Francisco CA Morgan Kaufmann. 2002; 11–18.
Reference Source
31. Birattari M: **Tuning Metaheuristics: A Machine Learning Perspective**. Springer, Berlin, Germany, 2009.
Publisher Full Text
32. Birattari M, Yuan Z, Balaprakash P, *et al.*: **F-race and iterated f-race: An overview**. In: Thomas Bartz- Beielstein, Marco Chiarandini, Luís Paquete, and Mike Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, Berlin, Germany, 2010; 311–336.
Publisher Full Text

33. Franzin A, Stützle T: **Revisiting simulated annealing: A component-based analysis.** *Comput Oper Res.* 2019; **104**: 191–206.
Publisher Full Text

34. Hasselmann K, Birattari M: **Modular automatic design of collective behaviors for robots endowed with local communication capabilities.** *PeerJ Comput Sci.* 2020; **6**: e291.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

35. Ligot A, Kuckling J, Bozhinoski D, *et al.*: **Automatic modular design of robot swarms using behavior trees as a control architecture.** *PeerJ Comput Sci.* 2020; **6**: e314.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

36. Nordmann A, Hochgeschwender N, Wigand D, *et al.*: **A survey on domain-specific modeling and languages in robotics**. 2016.
Reference Source

37. Neill CJ, Laplante PA: **Requirements engineering: the state of the practice.** *IEEE Softw.* 2003; **20**(6): 40–45.
Publisher Full Text

38. Van Lamsweerde A: **Goal-oriented requirements enginering: a roundtrip from research to practice**. In: *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004*. Washington, DC USA, IEEE. 2004; 4–7.
Publisher Full Text

39. Pinciroli C, Beltrame G: **Buzz: An extensible programming language for heterogeneous swarm robotics**. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016; 3794–3800.
Publisher Full Text

40. Beltrame G, Merlo E, Panerati J, *et al.*: **Engineering safety in swarm robotics**. In: *Proceedings of the 1st International Workshop on Robotics Software Engineering*. 2018; 36–39.
Publisher Full Text

41. Brambilla M, Brutschy A, Dorigo M, *et al.*: **Property-driven design for swarm robotics: A design method based on prescriptive modeling and model checking.** *ACM Transactions on Autonomous and Adaptive Systems.* 2015; **9**(4): 17.
Publisher Full Text

42. Kelly S, Tolvanen JP: **Domain-specific modeling: enabling full code generation**. John Wiley & Sons, 2008.
Publisher Full Text

43. Dixon C, Winfield AFT, Fisher M, *et al.*: **Towards temporal verification of swarm robotic systems.** *Rob Auton Syst.* 2012; **60**(11): 1429–1441.
Publisher Full Text

44. Bonani M, Longchamp V, Magnenat S, *et al.*: **The marxbot, a miniature mobile robot opening new perspectives for the collective-robotic research**. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Piscataway, NJ, IEEE., 2010; 4187–4193.
Publisher Full Text

45. Riedo F, Chevalier M, Magnenat S, *et al.*: **Thymio II a robot that grows wiser with children**. In: *2013 IEEE workshop on advanced robotics and its social impacts*. Tokyo, Japan, IEEE. 2013; 187–193.
Publisher Full Text

46. Rubenstein M, Ahler C, Nagpal R: **Kilobot: A low cost scalable robot system for collective behaviors**. In: *2012 IEEE Int Conf Robot Autom*. St Paul, MNUSA, IEEE., 2012; 3293–3298.
Publisher Full Text

47. Soares JM, Navarro I, Martinoli A: **The Khepera IV mobile robot: performance evaluation, sensory data and software toolbox**. In *Robot 2015: Second Iberian Robotics Conference*. Lisbon, Portugal, Springer. 2016; 767–781.
Publisher Full Text

48. Hasselmann K, Ligot A, Francesca G, *et al.*: **Reference models for AutoMoDe**. Technical Report TR/IRIDIA/2018-002, IRIDIA, Université libre de Bruxelles, Belgium, 2018.
Reference Source

49. Dwyer MB, Avrunin GS, Corbett JC: **Patterns in property specifications for finite-state verification**. In: *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No. 99CB37002)*. Los Angeles, CA, USA, IEEE., 1999; 411–420.
Publisher Full Text

50. Autili M, Grunske L, Lumpe M, *et al.*: **Aligning qualitative, real-time, and Probabilistic property specification patterns using a structured english grammar.** *IEEE Trans Softw Eng.* 2015; **41**(7): 620–638.
Publisher Full Text

51. Eysholdt M, Behrens H: **Xtext: implement your language faster than the quick and dirty way**. In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*. Nevada, USA, ACM. 2010; 307–309.
Publisher Full Text

52. Mondada F, Guignard A, Bonani M, *et al.*: **Swarm-bot: from concept to implementation**. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Piscataway, NJ, IEEE. 2003; **2**: 1626–1631.
Publisher Full Text

53. Bozhinoski D, Birattari M: **Integrated automatic design process for robot swarms.** 2021.
http://www.doi.org/10.5281/zenodo.5184720

54. Garattoni L, Francesca G, Brutschy A, *et al.*: **Software infrastructure for e-puck (and TAM)**. Technical Report TR/IRIDIA/2015-004, IRIDIA, Université libre de Bruxelles, Belgium, 2015.
Reference Source

55. Gutiérrez I, Campo A, Dorigo M, *et al.*: **Open e-puck range & bearing miniaturized board for local communication in swarm robotics**. In: Kinugawa Kosuge, editor, *IEEE Int Conf Robot Autom. ICRA*. Piscataway, NJ, IEEE. 2009; 3111–3116.
Publisher Full Text

56. Bozhinoski D, Birattari M: **Requirements specification for swarm robotics: Supplementary material**. 2020.
Reference Source

Open Peer Review

Current Peer Review Status: ✔ ✔ ✔

Version 2

Reviewer Report 04 November 2022

https://doi.org/10.21956/openreseurope.16243.r30379

© **2022 Hunt E et al.** This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

✔ **Edmund Hunt**
University of Bristol, Bristol, UK
James Ward
University of Bristol, Bristol, UK

I have no further comments to make.

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Swarm robotics

We confirm that we have read this submission and believe that we have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Version 1

Reviewer Report 07 June 2022

https://doi.org/10.21956/openreseurope.15113.r29339

© **2022 Hunt E et al.** This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

✔ **Edmund Hunt**
[1] University of Bristol, Bristol, UK
[2] University of Bristol, Bristol, UK
James Ward

[1] University of Bristol, Bristol, UK
[2] University of Bristol, Bristol, UK

This paper presents work on a textual language ('SML') for specification of swarm robot missions, and an engine to transform this SML into a form that can be used by the previously published 'AutoMoDe-Chocolate' automatic controller designer. The pipeline from mission specification to deployment is demonstrated in three example missions.

The ambitious goal to show a 'fully integrated design process' for swarm deployment, using off-line optimization methods, is met with an initial proof concept. Obviously, much work remains to be done on showing that this approach will be successful in 'real world' environments, both in relation to mission success (relative to alternative approaches, e.g. hand-design of controllers) and usability for non-expert users. But it is a useful step in that direction, and the paper therefore is a meaningful contribution to the field.

Specific points arising:

- Given the focus on swarms and emergent behaviour – e.g. the Introduction "Hence, the collective behavior of a robot swarm is a result of the local interactions between the individual robot and its neighbors and its environment" – the three missions tested do not especially rely on neighbour interactions/emergence, and could equally be tested on a single robot? And so the real-world validation is arguably on the boundary of what could properly be called swarm robotics, and I look forward to the promised future work on e.g. collective decision-making missions.

- In the Abstract, I would contest the claim that 'swarm designers need to follow standardised automatic design processes…' I suggest 'may benefit from following' rather than 'need to'.

- In section 3.1, it is claimed that the 'approach…enables…non-expert users (non-technical end-users) to specify requirements for robot swarms…'. Realistically though, how accessible is SML to a non-expert? Would some kind of graphical interface be easier to use and less prone to input errors?

- In section 3.2, the design budget is 200k simulation runs. This seems like a lot. What is the run time on what hardware? Why is such a large number necessary? What would be possible with e.g. 2k runs?

- 'Chocolate' operates on only six low-level behaviours and six conditions. This is fine for lab-based swarm concepts, but how extensible is this framework into the 'real world', e.g. with ROS-based robots running with sensors such as LiDARs, depth cameras, etc – robots with greater 'spatial intelligence' to understand context, performing more complex missions? I wonder if BDRML is a relevant approach.[1]

- It would be interesting to test missions where the simulation is not quite matching the deployed environment; e.g. with the aggregation mission, would it make much difference to specify 3 obstacles rather than the actual 5?

- On Fig. 1, realistically for real-world deployments, will there be a feedback loop between step 2 and 3? Take driverless cars as an example, development requires thousands of hours

of trials and controller refinement. Are you focused on missions where iterative testing and design will not be possible? Do you have specific contexts in mind?

- I find the large 'figures' showing code to be better suited to supplementary information, it interrupts the paper flow and makes the page count overlong.

- Minor points: p.3 'alea' – I am not familiar with this term, please clarify your meaning. p.3 'through the concept of region' – awkward phrasing.

References

1. Pitonakova L, Crowder R, Bullock S: Behaviour-Data Relations Modelling Language For Multi-Robot Control Algorithms. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017. 727-732 Publisher Full Text

Is the work clearly and accurately presented and does it cite the current literature?
Partly

Is the study design appropriate and does the work have academic merit?
Yes

Are sufficient details of methods and analysis provided to allow replication by others?
Yes

If applicable, is the statistical analysis and its interpretation appropriate?
Not applicable

Are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions drawn adequately supported by the results?
Partly

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Swarm robotics

We confirm that we have read this submission and believe that we have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 04 Aug 2022

Darko Bozhinoski

We are glad that the reviewer appreciated the work we have performed. In the following, we address his comments point by point.

- Concerning the idea of creating a graphical interface to facilitate the usage of SML,

we extended the Conclusions by adding this research direction as a future contribution.

- Concerning the large design budget of 200k simulation runs, we would like to stress that the decision of adopting such a large design budget is outside of the specific scope of the contribution we are making with this paper. We decided to adopt a large design budget, so that the design process has sufficient resources to obtain a controller that performs well. The focus here is on the specification of the mission and on the automatic process that transforms specifications into the input to be fed to Chocolate.

- Concerning the fact that the automatic design method Chocolate operates on only six low-level behaviours and 6 conditions, we would like to stress that the focus in this paper is not Chocolate per se, but rather how to define a fully automatic design process (from specifications to the actual execution of the mission). As we already mentioned in answer to the other reviewers, this work is only a first step towards an integrated automatic design process for robot swarms: a proof of concept implementation. Extending the framework from a lab-based environment to a real-world environment is definitely an important issue that will be addressed in future research work. In the Behaviour-Data Relations Modelling Language (BDRML) [1] approach, the authors propose a methodology to represent robot behaviours, data, and a set of conditional relations between the different primitives. In contrast, the main focus in our work is on establishing an end-to-end automatic approach where from a mission specification in natural language, swarm control software can be obtained without focusing on the specificities of data structures and behaviours.

- Concerning the idea of testing the approach on missions where the simulation environment does not fully match the deployment environment, we would like to point out that our current approach already provides support for it. Many environmental features can be described in a probabilistic manner, meaning that the automatic design process generates control software that is trained on a representative set of environments that are different from the one into which the swarm is eventually deployed. We refer to a class of missions (environments) and we only make the working hypothesis that all the environments experienced in simulation and the real one into which the swarm is deployed are part of the same mission class.

- We agree with the reviewer's comment that having a feedback loop in the automatic design process is an important part of the development of robust and reliable robot swarms. Yet, the setting we considered in the paper—that is, the linear life-cycle model—is the basic building block to achieve any more complex life-cycle. We feel that considering a lifecycle with a feedback loop will be an important future step. In this work, we focused on the simplest development model as we are making our first steps in closing the gaps in the fully automatic design process: starting from the specification of a mission, obtaining a swarm controller in simulation and finally deploying it in a real environment. We strongly believe that before we move to an iterative design process, we must focus on having a full-fledged linear automatic

design process and fully master it.
We thank the reviewer for his valuable feedback.

[1] Pitonakova L, Crowder R, Bullock S: Behaviour-Data Relations Modelling Language For Multi-Robot Control Algorithms. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2017; 727-732 |

Competing Interests: No competing interests were disclosed.

Reviewer Report 30 May 2022

https://doi.org/10.21956/openreseurope.15113.r28563

© 2022 Millard A. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Alan Millard
[1] Department of Computer Science, University of York, York, UK
[2] Department of Computer Science, University of York, York, UK

This article presents a novel textual language for the specification of robot swarm missions called Swarm Mission Language (SML). Model-driven engineering techniques are used to automatically transform missions specified in SML into a format compatible with the previously published AutoMoDe-Chocolate optimisation-based design method. The authors demonstrate the potential of this integrated design process on three different example missions, with validation on physical robots.

This work is a valuable contribution to the field, bridging the gap between the high-level specification of swarm behaviours and automated design methods that determine the behaviour of individual robots. Integration with an established simulation tool (ARGoS) and robot platform (e-puck) makes the approach accessible to the research community, and the source code and data have been made available to aid reproducibility.

The current implementation of SML is relatively limited - for example, rewards/penalties can only be expressed based on the spatial locations of robots (or objects) at a particular time. However, the language is designed to be extensible, so this shortcoming can be addressed in future development (by the authors, or a third party).

To aid the design process (particularly for non-expert users), future work may wish to consider the development of a graphical interface that can automatically generate SML files (e.g. via interactive placement of spatial entities), to avoid laborious textual specification directly in Xtext.

Although the end-to-end design integration is validated in this article, there is no empirical evaluation of SML's usability, so it is unclear whether this new specification language actually

makes the process of designing swarm behaviours significantly easier. The contribution would be significantly strengthened by conducting a user study like that presented in "An Experiment in Automatic Design of Robot Swarms: AutoMoDe-Vanilla, EvoStick, and Human Experts", to demonstrate the utility of SML in comparison to other methods like Buzz.[1]

Minor comments:

The figure captions are too brief and not sufficiently descriptive - it should be possible to understand what each figure represents in isolation, based on context from the caption.

Figure 11 is excessively long - I suggest that you remove the XML comments, as they simply repeat the named tagged sections. Please also modify this example to use relative paths, instead of absolute paths specific to the author's personal filesystem.

The code listings are quite difficult to read (especially the C++ code in Figure 12). Please consider using a condensed monospace font, reformatting the code, or including additional syntax highlighting to improve clarity. They should also be captioned as listings, rather than figures (like "Listing 1. Mission specification").

Page 6 - "Phototaxis" should be italicised like the other items.
Page 8 - Figure 3: "Perimetar" > "Perimeter"
Page 10 - Under "Transition": "e.g.a" > "e.g., a"
Page 11 - "s-boot" > "s-bot". Should this even say s-bot? The grammar presented in Figure 7 mentions only the e-puck and foot-bot (marXbot).
Page 17 - "Fist" > "First"
Page 17 - "xml" > "XML"
Page 17 - "ARGOS" > "ARGoS"

References

1. Francesca G, Brambilla M, Brutschy A, Garattoni L, et al.: An Experiment in Automatic Design of Robot Swarms AutoMoDe-Vanilla, EvoStick, and Human Experts in ANTS 2014: Ninth International Conference on Swarm Intelligence. *Springer*. 2014. 25-37

Is the work clearly and accurately presented and does it cite the current literature?
Yes

Is the study design appropriate and does the work have academic merit?
Yes

Are sufficient details of methods and analysis provided to allow replication by others?
Yes

If applicable, is the statistical analysis and its interpretation appropriate?
Not applicable

Are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions drawn adequately supported by the results?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Swarm robotics

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 04 Aug 2022

Darko Bozhinoski

We are glad that the reviewer appreciated the work done. In the following, we address the comments point by point.

- We agree with the reviewer: the current implementation of SML is limited. It is indeed intended to be a proof of concept and opens the way to future developments. This is reflected also in the title of the article: "Towards an integrated automatic design process for robot swarms". As the reviewer points out, the language is extensible, and can be further developed, possibly by third parties. In this first, proof-of-concept implementation we decided to use spatial locations of robots for rewards/penalties due to the fact that in our analysis on a set of swarm missions, we noticed that most of the swarm missions measure success through an indicator that assesses the swarm being in a particular location at a specific time or by a mission objective that can be easily converted to this indicator. This definitely does not address all possible missions of interest, but covers a good share of them and, in our opinion, provides a clear idea of the potential of the proposal.

- Regarding the idea of developing a graphical interface to aid the design process and avoid laborious textual specification, we fully agree with the reviewer and we have plans to extend this contribution by introducing a graphical interface for swarm designers to design their missions as discussed in our conclusions. Yet, we would like to emphasise that the current textual specification of the swarm mission is already a significant step ahead with respect to what is described in the literature. Regarding the suggestion of conducting a user study to evaluate the SML usability, we would like to highlight that an evaluation of this kind is a huge endeavour and really rare in the literature. We feel that this deserves to be done as a separate study. We consider that an empirical evaluation of SML is an important future contribution that requires a definition of a protocol for evaluation based on objective qualitative and quantitative metrics that demonstrate the usefulness of our domain-specific language.

- We have reviewed and fixed all the other minor comments and typos highlighted by the reviewer.

We thank the reviewer for his valuable feedback.

Competing Interests: No competing interests were disclosed.

Reviewer Report 08 February 2022

https://doi.org/10.21956/openreseurope.15113.r28418

© 2022 Schroeder A. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Adam Schroeder
[1] University of Toledo, Toledo, OH, USA
[2] University of Toledo, Toledo, OH, USA

The main contribution of this work is the creation of a formal and standardized way to specify the requirements for a robot swarm mission and how the performance of a swarm can be quantitatively evaluated. Their tool is called swarm mission language (SML). After the requirements for a mission have been specified in SML, existing software tools are used to simulate swarm performance, optimize performance, and eventually write a controller for a physical robot. Overall, SML in combination with existing software enables an automatic design process for robot swarms.

The authors validated their tool and the use of an automatic design process with three example missions.

This work fills a current research gap and creates a useful tool for a swarm designer. It is adopting automatic design principles that are used in other disciplines to the specific discipline of robot swarm design. The authors acknowledge which scenarios are currently supported by their tool and that it could be extended to support more.

Communication within a swarm is one important design decision and it would be helpful for the authors to explicitly discuss this and how it is addressed with their tool.

In thinking of actual missions I would try to design a swarm for, it would be useful for the authors to discuss if and how their tool could handle (i) unknown or partially-known environments, where it would not be possible to specify their details in advance, (ii) dynamic objects in an environment, (iii) environments with non-discrete objects, e.g. a continuous distribution of food sources instead of discrete food sources, (iv) outdoor environments. Of course, even if these elements were supported by SML, they also would need to be supported in any simulator used.

Is the work clearly and accurately presented and does it cite the current literature?
Yes

Is the study design appropriate and does the work have academic merit?
Yes

Are sufficient details of methods and analysis provided to allow replication by others?
Yes

If applicable, is the statistical analysis and its interpretation appropriate?
Not applicable

Are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions drawn adequately supported by the results?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Swarm robotics

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 04 Aug 2022

Darko Bozhinoski

We are glad that the reviewer appreciated the work we have done so far. In the following, we will provide details and explanation about the concerns he raised.

Regarding the question on how our work addresses communication within a swarm, we would like to mention that AutoMoDe-Chocolate, the automatic design method used in this work - does not work with communication behaviours. In other works, we have developed automatic design methods that design control software for robots described by an extended reference model which include the ability to communicate via light and infra-red signals. These design methods (AutoMoDe-Gianduja [1] and AutoMoDe-TuttiFrutti [2]) are similar to AutoMoDe-Chocolate, but include modules that implement communication.The SML language can be extended to serve these other design methods. Concerning the question on how SML can handle: (i) unknown or partially-known environments, (ii) dynamic objects in an environment, (iii) environments with non-discrete objects and (iv) outdoor environments, we would like to clarify that SML supports the specification of a class of missions where a probabilistic description of the environment is provided. The robot swarm performs the mission in this range of possible environments. In our current setup, SML enables specification of semi-structured partially-known environments, environments with non-discrete objects (region of food source) and the execution of the mission can happen in an indoor or outdoor environment. We thank the reviewer for his valuable feedback.

[1] Hasselmann, Ken, and Mauro Birattari. "Modular automatic design of collective behaviors for robots endowed with local communication capabilities." PeerJ Computer Science 6 (2020): e291.

[2] Garzón Ramos, David, and Mauro Birattari. "Automatic design of collective behaviors for robots that can display and perceive colors." Applied Sciences 10, no. 13 (2020): 4654.

Competing Interests: No competing interests were disclosed.