Model-Driven Simulation-Based Analysis for Multi-Robot Systems

James Harbin, Simos Gerasimou, Nicholas Matragkas, Athanasios Zolotas and Radu Calinescu
Department of Computer Science, University of York, UK
Email: {james.harbin,simos.gerasimou,nicholas.matragkas,thanos.zolotas,radu.calinescu}@york.ac.uk

***Abstract*—Multi-robot systems are increasingly deployed to provide services and accomplish missions whose complexity or cost is too high for a single robot to achieve on its own. Although multi-robot systems offer increased reliability via redundancy and enable the execution of more challenging missions, engineering these systems is very complex. This complexity affects not only the architecture modelling of the robotic team but also the modelling and analysis of the collaborative intelligence enabling the team to complete its mission. Existing approaches for the development of multi-robot applications do not provide a systematic mechanism for capturing these aspects and assessing the robustness of multi-robot systems. We address this gap by introducing ATLAS, a novel model-driven approach supporting the systematic robustness analysis of multi-robot systems in simulation. The ATLAS domain-specific language enables modelling the architecture of the robotic team and its mission, and facilitates the specification of the team's intelligence. We evaluate ATLAS and demonstrate its effectiveness on two oceanic exploration missions performed by a team of unmanned underwater vehicles developed using the MOOS-IvP robotic simulator.**

***Index Terms*—model-driven engineering, robotics, simulation**

I. Introduction

Multi-robot systems (MRS) are distributed and interconnected robotic teams deployed to carry out missions that are beyond the competency of a single robot [1]. MRS are particularly useful in missions that require: robust behaviour and fault tolerance since the system can utilise redundancy to cover for a failed robot by distributing its responsibilities between the healthy team members; long-term autonomy since robots can execute tasks in a round-robin manner and replenish their batteries when idle; and improved scalability and performance since tasks can be performed more efficiently through parallelism if they are decomposable. Example missions that would benefit from MRS capabilities include environmental data collection of a large marine area using a team of unmanned underwater vehicles (UUVs) [2], power plant inspection using aerial vehicles [3], and order fulfilment and restock within a warehouse using robotic arms and ground vehicles [4].

MRS can execute these missions through *collective intelligence (CI)* algorithms that encapsulate communication policies within the closed-loop control of individual robots (e.g., MAPE-K [5]) and adaptation strategies for delegating responsibilities within the team. The use of CI enables capitalising on the unique benefits offered by MRS in these business- and safety-critical application domains [6]. Consider, for instance, a team of UUVs, each equipped with sonar sensors, deployed to discover hazardous objects by scanning a large marine area. A CI instance may partition this area based on the UUV sensor capabilities (e.g., reliability, energy consumption) while also specifying how the team will respond and redistribute pending tasks when a team member fails or experiences difficulties (e.g., when a UUV enters an area where the water salinity or temperature is outside the operating envelope of its sensors).

Selecting a suitable CI is a very important problem that directly affects the MRS performance and resilience [1]. An effective CI would empower MRS to cope with uncertain environments (e.g., sudden changes in environmental conditions), evolving mission objectives and unpredictable degradation of robotic components (e.g., sensor failure of multiple robots) [7]. Unavoidably, this problem is non-trivial. Engineers have a plethora of different options both in terms of adaptation strategies and communication policies that complicate the process of designing, implementing and assessing candidate CI algorithms. The large design space that comprises robotics teams of different sizes and individual robots with a wide range of performance and functionality characteristics (e.g., sensor width, energy capacity) only exacerbates the task of choosing the most suitable robotic team and desired CI instantiation.

Despite recent advances in the specification and analysis of MRS [8], [9], existing approaches either focus on providing specialised robotic functionality (e.g., perception, control) [10]–[12] or software for specific robotic platforms (e.g., ROS [13] or MOOS-IvP [14]). This limits their applicability to MRS missions characterised by simplistic CI behaviour resulting in reduced MRS resilience or bespoke algorithms that intertwine the CI logic with low-level platform-specific code resulting in added maintenance cost [15]. These important limitations of existing approaches increase significantly the effort to explore the tradeoffs between candidate MRS designs [16] and adaptation strategies of different CI algorithms [17].

We introduce ATLAS, a model-driven, tool-supported framework for the systematic engineering of MRS that facilitates the exploration and tradeoff analysis of candidate MRS designs and CI algorithms. Driven by insights derived from recent robotics surveys [16], [18], ATLAS is underpinned by the following principles. First, the ATLAS domain-specific language (DSL) enables the specification of (i) the MRS mission, including both functional and non-functional requirements; and (ii) the characteristics of individual robots comprising the MRS, including architecture, internal behaviours, and capabilities (e.g., use of energy efficient or reliable sensors). Second, the ATLAS code generation engine consumes the

MRS mission and system specifications, and produces the necessary infrastructure (i.e., ATLAS middleware, CI templates, target simulator logical interface) that enables the communication of the ATLAS components with the target robotic simulator (e.g., MOOS-IvP [14], ROS [13]). Finally, the low coupling between the ATLAS components supported by the CI templates improves system extensibility while also supporting tradeoff analysis between different CI algorithms.

The main contributions of our paper are:

- The ATLAS tool-supported framework for the systematic engineering of MRS enabling tradeoff analysis of different MRS architectures and CI algorithms from the early stages of the MRS development process;
- An extensive ATLAS evaluation of two MRS case studies built with MOOS-IvP [14], a widely-used platform for the implementation of autonomous applications on UUVs;
- A prototype open-source ATLAS tool and case study repository, both available on our project webpage at https://www.github.com/jrharbin-york/atlas-middleware.

We structure the paper as follows. Section II presents a motivating example that we use to illustrate ATLAS, which is detailed in Section III. Sections IV and V describe our ATLAS implementation and evaluation, respectively. Finally, Section VI discusses related work, and Section VII summarises our results and suggests directions for future research.

II. Motivating Example

We will illustrate ATLAS using a UUV team deployed on an object detection mission within a large marine area that contains both benign and hazardous objects. Each UUV is equipped with a sonar sensor that can detect objects when they are in close proximity, localisation hardware and a radio transceiver to interface with a centralised control computer (shoreside) which runs the CI that coordinates the activities of the UUV team. Figure 1 shows the three UUVs executing lawnmower-style sweeps (horizontally back and forwards followed by vertical steps) over subdivided regions of the area, and three objects to be located (where green and red triangles indicate benign and malicious objects, respectively).

The shoreside uses a CI algorithm to coordinate the UUVs behaviour and fulfil the requirements shown in Table I. Given the safety-critical nature of this mission, all objects should be detected (R1). Depending on the object's type, one or two verifications by peer UUVs should be performed (R2), thus reducing the risk that an incorrect type has been assigned to the detected object. Since UUVs have limited battery capacity, it is important to partition the area effectively and complete the mission at the least possible time (R3) and before the maximum mission execution time given by $T_{max} = 2400s$.

TABLE I: UUV requirements for the object detection mission.

ID	Description
R1	All environmental objects within the area should be detected.
R2	When a UUV detects a malicious (benign) object, the detection must be verified by two (one) peer UUVs.
R3	Subject to satisfying R1 and R2, the CI should coordinate the UUVs so that the mission execution time is minimised.

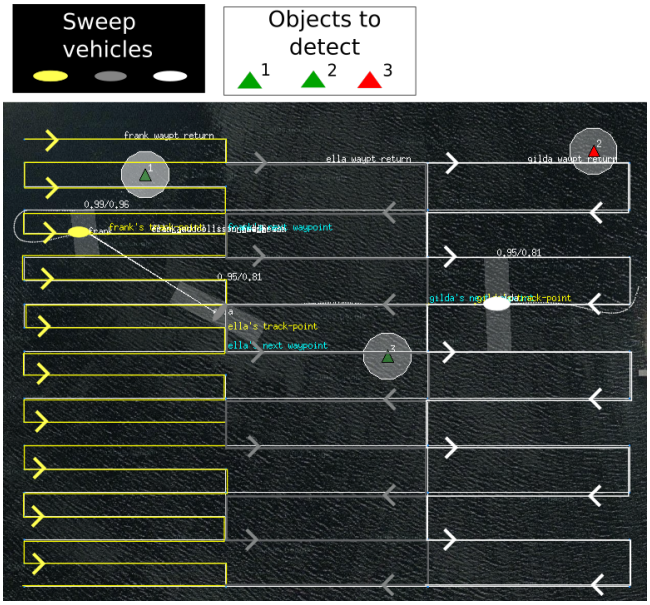


Fig. 1: UUV team deployed on a hazardous object detection mission, developed using the MOOS-IvP simulator [14].

When designing the UUV team, engineers want to investigate how effectively teams comprising three or four UUVs would accomplish the mission. Furthermore, each UUV can be equipped with a wide or a narrow sensor whose scanning area is 10m or 20m wide, respectively. Consequently, this results in 48 possible configurations (i.e., designs) for the UUV team.

Beyond the UUV team configurations, several CI algorithms could be used to support the execution of the object detection mission. We assume that robotic engineers are interested in evaluating the following standard and advanced CI algorithms.
- *Standard CI*: This CI partitions the area between the UUVs equally, based on a lawnmower pattern with a constant vertical separation, independent of each UUV's sonar sensor range. When performing verifications of detected objects (R2), the CI selects the UUV that is the closest to the object. Verifications are performed for a constant fixed time of 600 seconds, scanning the area around the detection. When completing the verification, the UUV is commanded to resume its original sweep region from the beginning.
- *Advanced CI*: This CI partitions the area between the UUVs proportionally to the strength of their sonar sensors, i.e., a UUV equipped with a more capable sensor is assigned a larger area than a UUV equipped with a narrow sensor. The CI also monitors the status of the verification, returning the dispatched UUV back to its originally assigned area as soon as the final waypoint of the verification task is reached. When this happens, the CI instructs the UUV to resume its task from the point at which it was interrupted.

Evidently, the advanced CI is more efficient and aims at reducing the overall mission execution time but incurs significant communication cost due to the frequent communication between the dispatched UUV and the shoreside. On the contrary, the standard CI, albeit slower, is more meticulous and may help the UUV to discover previously missed objects.

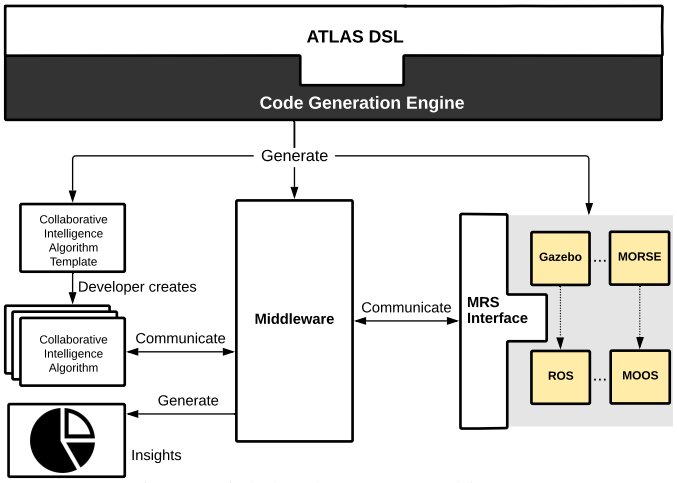


Fig. 2: High-level ATLAS architecture

III. ATLAS

A. Overview

Figure 2 shows the high-level ATLAS architecture. The core of ATLAS comprises the ATLAS DSL (Section III-B) for the specification of the team structure and mission objectives, and a model-driven code generation engine for the generation of a lightweight middleware (Section III-D), a simulator-specific logical interface, and CI templates that facilitate tradeoff analysis of CI algorithms (Section III-E). The middleware is responsible for moderating the communication between team members (team level) and between the components within an individual robot (robot level). To achieve this, ATLAS exploits the modular structure of robots and the publish-subscribe protocol underpinning widely-used robotic platforms such as MOOS-IvP [14] and ROS [13]. A robot is a hierarchical composition of software and hardware components that interacts with the environment and communicates with its peers via exchanging messages using input and output interfaces [7]. The middleware also reinforces the separation of concerns between the MRS and the underlying CI algorithms which are responsible for steering the team to achieve its mission. This is a unique characteristic of ATLAS that facilitates the investigation of different CI algorithms [17] (e.g., decentralised collective learning, leader-election algorithms) for carrying out the specified mission and comparing non-functional attributes such as scalability and performance. The simulator-specific logical interface enables the direct communication between the middleware and the running simulator, thus supporting data exchange and monitoring of the MRS state during simulation. This interface is also a key enabler of ATLAS that not only reduces the coupling between framework components and improves extensibility, but it also enables to connect and interchange different robotic simulators easily for experimentation and analysis. These have been among the key challenges identified in recent robotics surveys [16], [18] and ATLAS contributes in addressing them.

In the following sections, we detail the fully-fledged ATLAS instance for the MOOS-IvP robotic simulator, and also describe our preliminary implementation for the model-driven

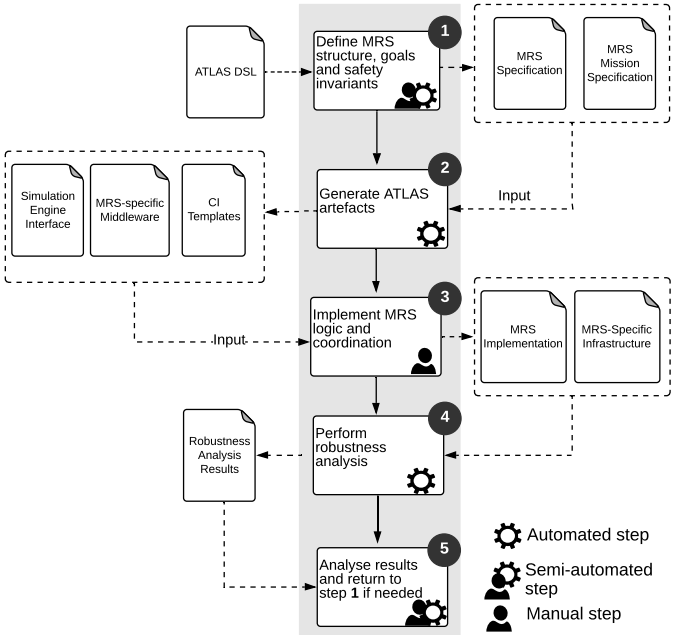


Fig. 3: Stepwise model-driven ATLAS methodology

generation of the corresponding interfaces for Gazebo/ROS (Section III-D), thus demonstrating the generality of ATLAS.

The high-level workflow of the ATLAS methodology is shown in Figure 3. Through the use of the purpose-designed ATLAS DSL, domain experts can define models of the MRS mission (i.e., requirements, goals and safety invariants) and team composition (Step 1). The ATLAS code generation engine consumes these models and automatically generates in Step 2 (i) a middleware that enables the communication of the various system components; (ii) CI templates which will be used by the middleware to coordinate the robotic team; (iii) interface code that enables the middleware to communicate directly with the target robotic simulator; and (iv) the necessary configuration files for the target robotic simulator. During Step 3, engineers implement the logic and coordination of the system. To this end, they populate the CI templates generated in Step 2 with suitable code that realises their chosen CI algorithms. This separation of concerns between the implementation of the high-level behaviour (exhibited by the algorithm) and the low-level functionality of a robot has two major benefits. First, it reduces the effort to analyse multiple CI algorithms without changing the underlying low-level system behaviour. Second, it enables to reuse already available low-level code developed for individual robotic team members that has been developed independently. When the necessary artifacts are implemented, the MRS simulation analysis is automatically executed in Step 4. ATLAS records simulation results related to the MRS mission requirements in the form of logs comprising both messages exchanged between system components and events occurred during the simulation. The analysis of these results enables the selection of the most effective MRS designs and the identification of the most suitable CI algorithm for the target mission.

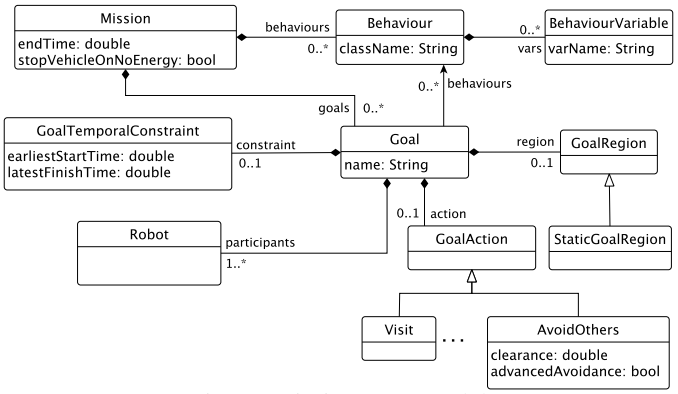


Fig. 4: Mission Metamodel

B. DSL Core Concepts

At the core of the ATLAS DSL is the concept of a *Mission* (Figure 4) which includes the *Goals* that should be fulfilled. Each goal is attached to the *Region* where the goal takes place. Goals also include a reference to the specific *GoalAction* that should be executed as part of the goal. Example actions include *Patrolling* an area, *Avoiding* an obstacle, etc. Although the DSL covers the vast majority of actions available in the current MOOS robotic environment, this is an extensibility point of the DSL; interested users can extend the *GoalAction* type to introduce new actions and the appropriate model-to-text (M2T) transformations to generate the corresponding MOOS implementations. Since goals may need to start after or end before a specific time, each goal is linked to a *GoalTemporalConstraint* that defines the earliest starting and latest finish time of the goal. Also, since each goal is mission-specific, implementing the actions for each goal is left to the user. For example, the GoalAction named TrackDistances will track the relative distances of robots to each other and check for possible intersection with environmental objects. This information is recorded allowing the production of related metrics for further analysis.

Robotic simulators provide bespoke implementations of various behaviours such as navigating to specific coordinates (e.g., using the *WayPoint* and *MoveBaseGoal* behaviours in MOOS-IvP and ROS, respectively), and avoiding other robots and obstacles. These behaviours can be employed by low-level simulator code to facilitate the specification and execution of a robotic mission. This information is also important for the CI. The DSL enables linking simulator-specific behaviour, represented in the model as elements having the name of the behaviour, to high-level mission goals through the *bevaviours* reference. The *Behaviour* class enables specifying multiple behaviour variables (topics in ROS) from the MRS simulator that capture the status of a robot. The intent of including these behaviour variables in the DSL is to make the middleware aware and instrument the communication of any value updates to the CI. When an update occurs, the CI can use the updated information and respond appropriately to a particular low-level MRS event, e.g., terminate the mission when notified about the successful traversal of a set of waypoints.

Another important element of the ATLAS DSL is the set

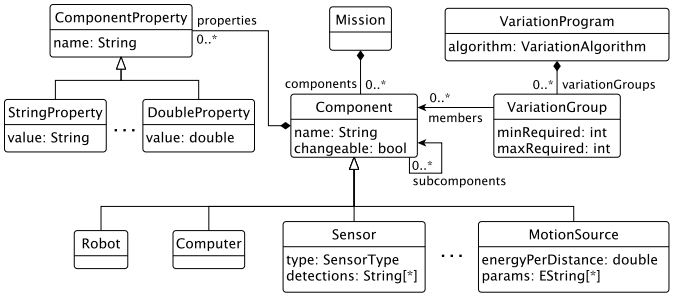


Fig. 5: Components Metamodel

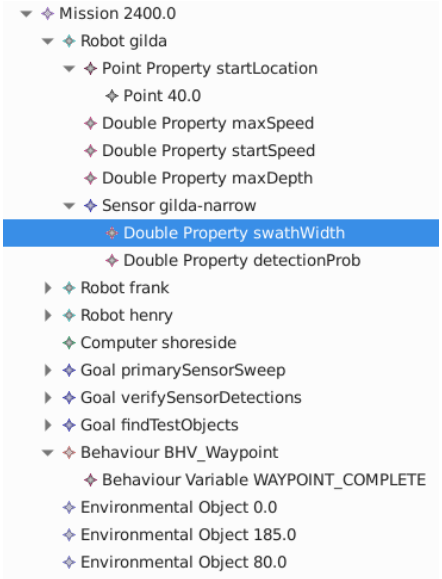


Fig. 6: Model of the UUV mission from Section II.

of *Robot*s employed to satisfy a specific goal. Conforming to the hierarchical representation of many robotic systems [12], ATLAS enables the specification of the MRS architecture in a compositional manner (Figure 5). A *Component* represents the top level element of this hierarchical representation and can either be a *Robot* (e.g., the UUVs in the motivating example) or a *Computer* (e.g., a shoreside computer that is responsible for the execution of the CI algorithm and the coordination of the robots). Robots and Computers consist of a number of *subcomponents* (e.g., *Sensor*s, *Actuators*, *MotionSource*s, etc.). Also, each component contains *ComponentProperties* of different datatypes that enable the specification of different characteristics of the associated component (e.g., nominal operating rate of a sensor or expected energy consumption).

Example 1: Figure 6 shows the model instance for the UUV mission from Section II. The model comprises the specifications of the three UUVs (gilda, frank, and henry), the shoreside computer which executes the CI strategy, the various mission goals and the coordinates of the three environmental objects that must be detected by the UUV team.

C. Searching for Optimal Robot Configurations

In addition to the specification of concrete MRS instances, the ATLAS DSL supports also the analysis of candidate

MRS designs through design space exploration. To this end, we leverage concepts from the domain of software product line engineering [19]–[21] and represent alternative candidate designs as a *VariationGroup* with a defined cardinality (cf. Figure 5). The members of a variation group are components, at the same level of the MRS hierarchy, that can participate in the mission; this includes robots and subscomponents. The set of all variation groups forms a *VariationProgram*. Given such a program, ATLAS uses model-to-model (M2M) transformation to automatically generate all possible mission models that conform to the mission metamodel in Figure 4 and meet the *min/maxRequired* properties of each group. For example, consider the UUV team from Section II and assume we want to assess if the mission can be fulfilled using a subset of the available UUVs. The UUVs are added in the *Mission* and the *VariationGroup* referring to these robots is created. The *minRequired* and *maxRequired* properties are set to three and four, respectively. Subcomponents can also form variation groups. For example, if there are different types of *Sensor*s (or sensors of the same type but with different properties) that a robot can use for a mission, those subcomponents can form another variation group. ATLAS consumes the variation groups and through M2M transformation automatically generates the possible mission model *configurations* (designs).

Example 2: Figure 7 shows the variation model (irrelevant details are omitted for reasons of brevity) for the variation scenario of the motivating example. The execution of the M2M using as input this variation model will generate the 48 possible configurations that conform to the constraints set in the variation groups, i.e., each robot should use either the wide or narrow sonar sensor and the mission should use at least three out of four available UUVs. The configurations comprise 16 models with four robots and 32 models with three robots all with different narrow (10m) or wide (20m) sensor instantiations.

D. ATLAS Middleware

The middleware is a key component of ATLAS that enhances *separation of concerns* between the CI instances and the target robotic simulator. The CI receives information about the MRS status via the middleware, executes its logic, and relays back its decisions to the MRS via the middleware. Also, the middleware uses runtime monitoring to assess the status of the goals defined in the DSL. This internal state, including goal events, is stored in log files that allow the computation of relevant mission metrics and post hoc analysis. The low coupling between the CI and the MRS simulator, mediated by the middleware, enables not only to experiment easily with several candidate CI algorithms but also reinforces maintainability and extensibility (e.g., the components can be computing platform and programming language independent).

Irrespective of the target robotic simulator, the ATLAS middlerware comprises (i) a simulator-specific interface that enables ATLAS to connect to the simulator, and subscribe and publish messages/topics; (ii) a highly-efficient message broker (e.g., ActiveMQ [22]) to facilitate fast inter-robot communication and interaction with the CI algorithm; and (iii) a CI mapping software module that converts the high-level CI commands into message/topic changes for the target simulator. ATLAS uses the mission model (e.g., Figure 6) and automatically generates these components through a series of M2T transformations. Currently, we fully support the MOOS-IvP simulator and have a prototype implementation for Gazebo/ROS. Clearly, for each target simulator the M2T transformation and required components are developed only once and can be reused thereafter. Due to space constraints, we briefly discuss below the concrete instantiations of these two simulators. The full details of the M2T transformations are available on our project webpage.

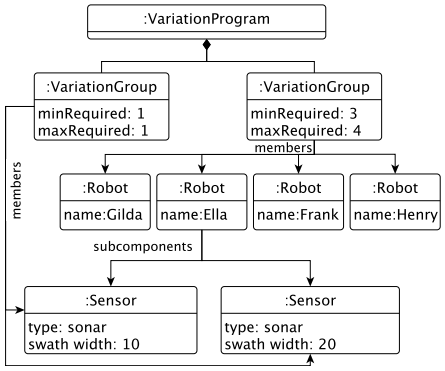


Fig. 7: Variation model for the UUV mission from Section II

ATLAS middleware for MOOS-IvP. The M2T transformation uses the mission model and configures appropriately the ATLAS middleware to enable monitoring the MRS status for the specified goals. Also, MOOS-specific configuration files are produced to represent the robot configurations, properties and behaviours necessary to run the simulation. Within MOOS-IvP, each robot is represented as a community comprising a set of C++ software modules that provides the functionality of robot components. Each community is served by an individual publish-subscribe database (MOOSDB) which contains key-value pairs. When robots communicate, these key-value pairs form a communication channel through which one robot will publish a message and subscribed robots will receive the update and act accordingly. Example messages that can be sent/received include sensor detection events, speed values and location information.

The communication between the ATLAS middleware and the simulator occurs through MOOSDBInterface, a generic MOOS software component that enables interfacing directly with robot communities. The simulator did not support this functionality. Thus, we developed this reusable software component which can now be instrumented through the middleware to publish/subscribe to messages within MOOSDB databases, e.g., receiving updated robot coordinates, activating the return home behaviour upon mission completion.

Messages received by the MOOS-IvP simulator are kept in the message broker until they are automatically processed and translated into update nodes mapped to the initial mission goals. These updates are transmitted to the CI algorithm

enabling revision of its internal state and informing subsequent decision-making. The middleware also supports translation of generic requests to low-level MRS messages (e.g., return to home or go to this location commands).

ATLAS middleware for Gazebo/ROS. This simulator realises also the publish-subscribe architecture with a topic subscription graph being the primary communication mechanism between robots and their components. Accordingly, the M2T transformation produces the necessary ROS launch scripts, which set out the simulation component task graphs, and the necessary ROS configuration to launch the ROS processes and implement the simulation functionality.

The middleware communicates with the ROS simulator using *rosbridge* (http://wiki.ros.org/rosbridge_suite), a widely-used and stable ROS component that serves as the logical interface module between ATLAS and Gazebo. Through rosbridge, ATLAS can subscribe to ROS topic updates, triggering notifications when these subscribed topics are updated, and publish new or edit existing topics. Dynamic topic subscriptions can be added/removed during simulation, thus enabling to reconfigure the interface during mission execution. Similarly to MOOS-IvP, the message broker sets up topic subscriptions to the simulator (e.g., to obtain the pose and velocity information) and stores updates on ongoing sensor events published to topics for further processing by the CI. Finally, CI actuation commands (e.g., UAV take off/land) are translated into specific low-level ROS topic changes, in order to implement the behaviour changes specified by these commands.

As already reported, the ATLAS middleware for Gazebo/ROS is still an early prototype. Hence, evaluating ATLAS using this simulator is outside the scope of this paper. We report this early work here to highlight the generality of our ATLAS framework, its support for multiple robotic simulators as well as the ability to add new simulators with modest effort.

E. Collective Intelligence

Through collective intelligence (CI), engineers can encode the high-level logic to coordinate robot decisions and manage the behaviour of the overall MRS. In addition to supporting the analysis of different robotic team configurations using variation groups (cf. Section III-C), ATLAS enables the automated evaluation of alternative CI algorithms. Users can inspect the mission metrics produced by each CI instance and select the best for hardening the MRS. This is another key feature of ATLAS that reduces further the coupling between the target robotic simulator and the high-level decision-making process.

To facilitate the development of CI algorithms, ATLAS performs an M2T transformation using the mission model to generate CI templates. These templates have empty placeholder methods that reflect the mission goals, behaviours and events mapped to robot components defined within the mission model (Figure 6). Engineers should specialise those templates with the appropriate logic to develop the target CI algorithm (cf. Step 3 in Figure 3).

The communication of fully-fledged CI instances with the ATLAS middleware is underpinned by the *inversion of control* programming paradigm [23]. To achieve this, the middleware becomes aware of the methods within the CI template and the messages associated with each method during the M2T transformation. When the middleware receives an update to a subscribed message/topic from the robotic simulator, it maps the message to the appropriate CI method and proceeds with its invocation. The CI executes its logic and informs the middleware for its decision (e.g., instruct a robot to take over a failed peer) so that the latter can send the appropriate message updates to the simulator via the logical interface. Since the CI instance runs as an independent Java process, both the CI and the middleware run independently and operate in a non-blocking mode using JSON messages.

The current ATLAS version supports the delegation of CI control to a single MRS component. This component can be a centralised control station or a single robot that acts as the leader with full knowledge and total control over its peers. Supporting other CI variants like hierarchical or decentralised [6] is out of scope and is left for future work.

Example 3: Listing 1 shows an except of the advanced CI for the UUV team from Section II. The method `init` is invoked when the simulation begins to set up the required CI data structures (e.g., the robotic team), to split the monitored region between the team based on the sensor capabilities of each robot and send the initial commands to activate the patrolling behaviour for the robots (lines 1–5). When a new sensor detection occurs, the middleware invokes the `SONARDetection` method providing also the robot that performed the detection and the information of the object. This method first checks whether the object has not been detected before by the UUV team (line 11). If this holds, and depending on the object type, the required number of UUVs to verify the detection are

Listing 1: Advanced CI excerpt of the UUV team (Section II)

```
1  public static void init () {
2    List<Robot> robotTeam = setupRobotTeam();
3    Map<Robot, Region> regionAssignments = splitRegion(fullRegion, robotTeam);
4    ...
5  }
6
7  public static void SONARDetection (SensorDetection det, Robot rbt) {
8    int objID = (int)det.getField("objectID");
9    String detType = det.getField("type");
10   \*check when object was checked before, benign or malicious*\
11   if (isObjectNew(objID)) {
12     if (isObjectBenign(detType))
13       for (int i=0; i<BENIGN_VERIFICATIONS; i++)
14         verifyDetection (objID, robotTeam, rbt);
15     else
16       for (int i=0; i<MALICIOUS_VERIFICATIONS; i++)
17         verifyDetection (objID, robotTeam, rbt);
18 }
19
20 public static void verifyDetection (SensorDetection det, List<Robot>
       robotTeam, Robot rbt) {
21   Point loc = (Point) detection.getField("location");
22   String rbtVer = chooseRobot(loc, robotTeam, rbt);
23   if (isRobotValid(rbtVer)) {
24     API.setSweepAroundPoint(rbtVer, loc, SWEEP_RADIUS,
       VERTICAL_STEP_SIZE_CONFIRM_SWEEP,
25         ("UUV_COORDINATE_UPDATE_VERIFY_" + rbtVer));
26     CILog.logCI("Setting robot " + rbtVer + " to verify detection");
27   }
28   else
29     CILog.logCI("ERROR: No robots available to confirm the detection");
30 }
```

selected using the `verifyDetection` method (lines 12–17). Every call to `verifyDetection` selects a UUV that is not currently involved in another verification task (line 22) and uses an API method to communicate with the middleware and instruct the UUV to execute the verification of an area around the detection zone (line 24). This information is logged for post hoc analysis by the user (lines 26 and 29). A UUV executing a verification returns to its original sweep patterns using a behaviour variable (i.e., `WAYPOINT_COMPLETE`) which is set to true when the navigation around the detection zone is completed (the code is not shown here due to space constraints). The full CI implementation is available at https://tinyurl.com/ATLAS-ExampleAdvancedCI.

IV. ATLAS Prototype

The prototype ATLAS model-driven tool uses the Epsilon family of languages [24] to perform the MDE tasks. In particular, DSL is implemented with EMF [25]. We use Epsilon's EMF Model Java API to check the variation groups and produce the candidate MRS designs (Section III-C) and the Epsilon Generation Language to generate the ATLAS middleware (Section III-D), the configuration files needed by the target simulator and the CI template (Section III-E). We also use ActiveMQ [22] to link the MRS simulator, CI algorithm and the middleware. The open-source ATLAS source code, the full experimental results summarised next and the case studies used for its evaluation are available at https://www.github.com/jrharbin-york/atlas-middleware. Finally, a video showing the execution of ATLAS for the case study described in Section II is available at https://tinyurl.com/ATLAS-ExampleVideo.

V. Experimental Evaluation

A. Research Questions

RQ1 (Configuration Analysis). Can ATLAS help with finding the optimal configuration for a robotic team? We use this research question to analyse if ATLAS can support the specification and analysis of different variation points in an MRS and enable the selection of the optimal configuration.

RQ2 (Collective Intelligence Analysis). Can ATLAS support tradeoff analysis between different collective intelligence algorithms? We use this research question to analyse if ATLAS can assess the situations in which different CI algorithms perform well on specific metrics, thus enabling robotic developers to select optimal CIs for a given mission.

RQ3 (Reproducibility). How does the non-determinism of robot simulations affect the reproducibility of the ATLAS results? We analysed if ATLAS can enable the identification of non-reproducible behaviours of MRS configurations and the discovery of outliers, thus providing evidence that the MRS, CI or the overall system may produce sub-optimal behaviour.

B. Evaluation Methodology

Case Studies. Following the standard practice in empirical software engineering [26], [27], we evaluate ATLAS using two distinct case studies from the domain of UUVs using the MOOS-IvP robotic simulator [14]: (1) the object detection UUV mission described in Section II and developed by the ATLAS team; and (2) a variant of the Bo-Alpha mission [28] developed by the MOOS-IvP community and included within the standard package of the simulator, which we describe next.

TABLE II: UUV requirements for the object detection mission.

ID	Description
R1	Each vehicle must complete more than one sweep of the left and right areas, alternating periodically once both completed
R2	Vehicles must avoid entry into the obstacle regions
R3	Vehicles must return to their starting points before their battery is depleted

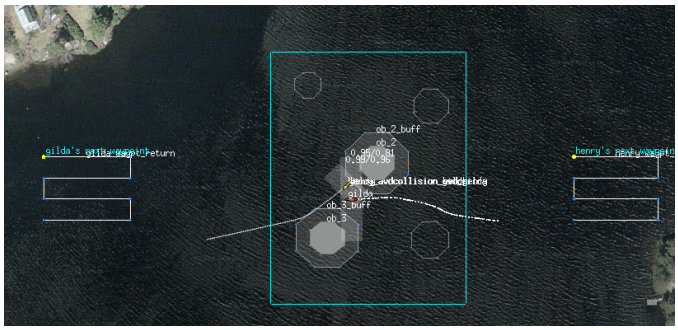


Fig. 8: Bo-Alpha mission for UUV monitor and avoidance

Bo-Alpha mission. UUVs make measurements (e.g., salinity, temperature) on the left and right sides of a topology, occasionally alternating sides. To avoid collisions, the UUVs must not come too close to each other and also avoid obstacle zones. The UUVs monitor their residual energy levels and communicate this information and positions to a control station running the CI. Figure 8 shows two robots crossing the central region containing five obstacles depicted as white octagons.

The mission is executed for $T_{max} = 1200s$ and the CI coordinates the team's behaviour to fulfil the requirements in Table II. The metrics used to assess the satisfaction of the requirements are the residual energy of the UUVs after mission completion and the total UUVs distance from their return locations. A competent CI would steer the robots back to their base with sufficient residual energy.

Engineers are interested in analysing the CI instances below.
- *Standard CI*: The UUVs alternate sides every 150s. This CI recalls the robots to return home 150s before T_{max}.
- *Energy-based CI*: This CI algorithm tracks the positions of the UUVs and alternates them when both have finished their assigned area. Also, this CI monitors the energy remaining on the UUV battery and sends the recall command to return to base when the critical energy threshold of 750mAh is met.

For this UUV mission, there are 16 possible configurations: two robots with a "fast" or "slow" version each, travelling at 1.5m/s and 3.0m/s, respectively. Each UUV also supports a standard (2800mAh) and a high capacity (5000mAh) battery.

Experimental Setup. We performed a wide range of experiments using the object detection and Bo-Alpha UUV missions. The setup involved running ATLAS and MOOS-IvP in both missions for the two CI variants and all possible configurations (48 and 16, respectively). All experiments were run on a *Ryzen 5 3600* machine with 16Gb of RAM using VirtualBox 6.1.

C. Results and Discussion

RQ1 (Configuration Analysis). Figure 9a shows the number of missed detections for the 48 possible MRS configurations of the object detection mission using the standard (top) and advanced CI (bottom). Under the standard CI, ATLAS identified eight optimal configurations in which the MRS completed its mission successfully with zero missed detections. Other configurations produce a wide range of possible detection failures, up to the worst case of seven missed detections.

We analysed the produced MRS configurations to discover factors that can affect the system performance and reliability. Our analysis showed that the optimal MRS designs for a team of four UUVs always equipped the UUV gilda with a wide sensor, while for a three UUV team all UUVs should be equipped with a wide sensor. Engineers can factor in the cost of robots and sensors and decide whether a three- or four-robot team is preferred for this mission. Further analysis revealed a particularly sensitive MRS design when UUV gilda uses a narrow sensor. These designs tend to produce three missed detections, due to the UUV gilda missing the initial detection of the rightmost object in its assigned area and two subsequent verifications. Accordingly, such designs fail to meet the mission requirements and should not be preferred.

Figure 9b shows the relation between the number of missed detections and mission completion time for the object detection mission. Some MRS configurations using the standard CI exhausted the available time signifying that at least one UUV did not complete its task on time. We identified a general pattern indicating a tradeoff with a decline in sweep completion time at the cost of partial mission completion, i.e., the mission can complete earlier when fewer detections are performed. MRS configurations that detected no object completed around 750-1600 seconds, where 750 seconds correspond to the shortest time needed to complete a sweep. Since no verifications were performed, this is expected. Subsequent analysis showed that most of these configurations comprise three UUVs all equipped with a narrow sensor. This configuration produces missed detections due to the width of the sweep patterns used in the standard CI, especially when objects are in the middle of the assigned UUV regions.

Considering the Bo-Alpha mission, we found one MRS configuration comprising two fast robots that completed two full sweeps using the standard CI (Figure 10a). However, this configuration did not meet the other mission requirement as it failed to steer the UUV team back to its base with sufficient residual energy. Further analysis of the configurations that completed a single sweep yielded a configuration that reported some residual energy and a mean vehicle distance of less than 50 meters. This configuration consists of a slow vehicle (*henry*) with a large battery, that enables the UUV to return to base with some residual energy, and a fast vehicle (*gilda*) with a smaller battery that can complete its sweep sooner.

These findings clearly demonstrate that ATLAS can support the analysis of different MRS configurations via its variation groups. The outcome of this analysis enables the selection of an optimal configuration for a given mission and requirements.

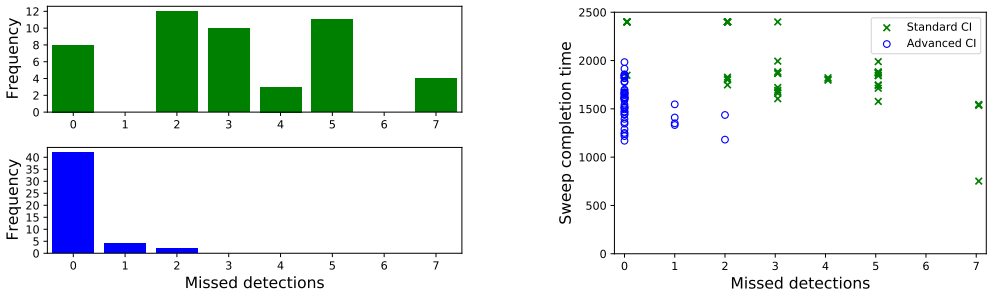
RQ2 (Collective Intelligence Analysis): To answer this research question, we compared the standard and advanced CI algorithms in both case studies. For the object detection mission, Figure 9a shows that the advanced CI produces a better worst-case scenario than the standard CI, with two missed detections at most. In fact, over 40 MRS designs succeed without missing an object failure. This behaviour is mainly caused by the CI adapting to the sensor range and using a smaller vertical sweep step size for narrow sensors. Also, many MRS configurations using the advanced CI completed the mission faster and had fewer missed detections than the corresponding configurations using the standard CI (Figure 9b). This behaviour occurs because the advanced CI instructs the UUV, after completing its verification task, to resume its original task from the point at which it was interrupted.

The analysis of the Bo-Alpha mission showed no optimal MRS configurations with the standard CI, since every configuration fails to complete more than one sweep of the left and right areas (R1 violation) or fails to return to base (R3 violation). The low number of completed sweeps using the standard CI is due to the improperly short timing that alternates the UUVs between different areas before they have completed their sweeps (Figure 10a). In contrast, the energy-based CI (with the low energy return threshold of 750mAh) produces a larger number of completed sweeps with a modal value of six. Most of the energy-based MRS configurations return the vehicles back to base (being at most 50 metres from the base).

The standard CI experiences many cases with very low residual energy (Figure 10c). The overall better performance of the energy-based CI occurs because this CI uses a waypoint completion feedback to alternate the sweep sides between the vehicles, rather than a time limit as in the standard CI. The better performance is also due to using the energy feedback for sending the return command sufficiently early, rather than waiting until getting close to the simulation end time.

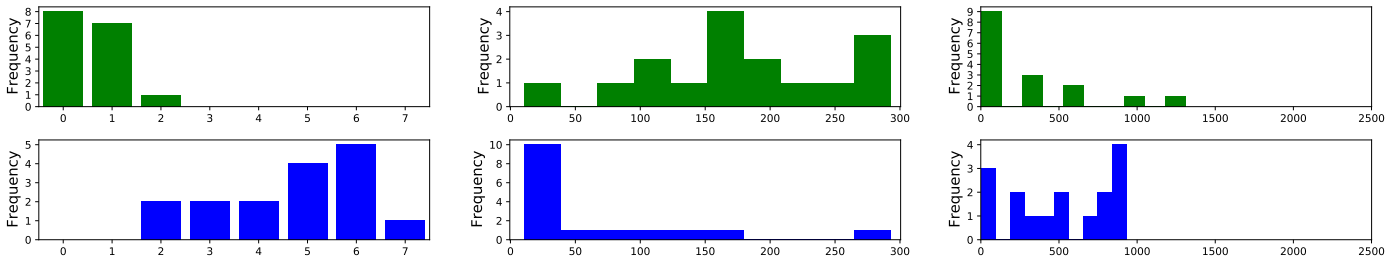
These results provide sufficient empirical evidence that ATLAS can support the comparison of different CI algorithms given a set of mission-specific metrics. Engineers can use these results to select optimal CIs for specific combinations of missions and robots. We note that the manual crafting and analysis of designs and CI algorithms for robotic teams was the status quo before ATLAS. Our experience with developing robotic missions using a non-ATLAS-based solution was the primary motivation for devising ATLAS.

RQ3 (Reproducibility). We assessed the impact of non-determinism in the reproducibility of simulation-based evaluation of MRS components by analysing the best MRS configuration and CI algorithm pair for each case study over 30 independent runs. Non-determinism can be due to robot behaviour (e.g., path planning), components of the simulation engine, or the operating system [29]. Non-deterministic simulations can produce behaviour that is not representative of real systems. Regardless of the source of non-determinism, we can assess the behaviour of a system via repeated executions of a fixed



(a) Missed detections with standard CI (top) and advanced CI (bottom) (b) Missed detections vs Mission completion time

Fig. 9: Results for the object detection UUV mission over the 48 MRS configurations for each CI instance



(a) Number of completed sweeps (b) Total distance from base (c) Residual energy

Fig. 10: Bo-Alpha mission results over the 16 MRS configurations for the standard CI (top) and energy-based CI (bottom)

system configuration. This provides a distribution of metrics over multiple runs, enabling the identification of outliers in which non-determinism may produce sub-optimal behaviour.

In the object detection mission, we used a configuration that produced the lowest timing and zero missed detections in research question RQ2. Figure 11 shows the relationship between missed detections and completion time for both CIs over 30 independent runs for this configuration. In all runs, there is a longer completion time under the standard CI than the advanced CI. This is not surprising since with the standard CI the UUV restarts its sweep pattern from the start after each verification. Moreover, the completion time for the advanced CI with no missed detections is generally clustered around the 1100-1300 second range, with some advanced CI executions producing a single missed detection and correspondingly a slightly shorter sweep time. This behaviour occurs because when there is a missed detection, at least one robot will not be interrupted for verifications, thus allowing the robot to complete its task slightly faster.

In the Bo-Alpha mission, we used a configuration that obtained the maximum number of sweeps in research question RQ2 with large batteries on both vehicles. The results over 30 independent runs show that there is little variability (Figure 12). Generally, the energy tracking CI performs well and better than the standard CI. The final distance from the base in Figure 12b shows that generally the metric is always constant, with a large mean distance with the standard CI and a small value with the energy tracking CI. Given that the robots are equipped with large batteries, the results show a considerable amount of final energy left upon the vehicle which is moving slower. Interestingly, there are a couple of outlier cases, which

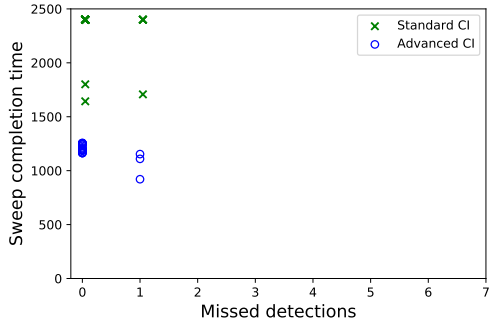
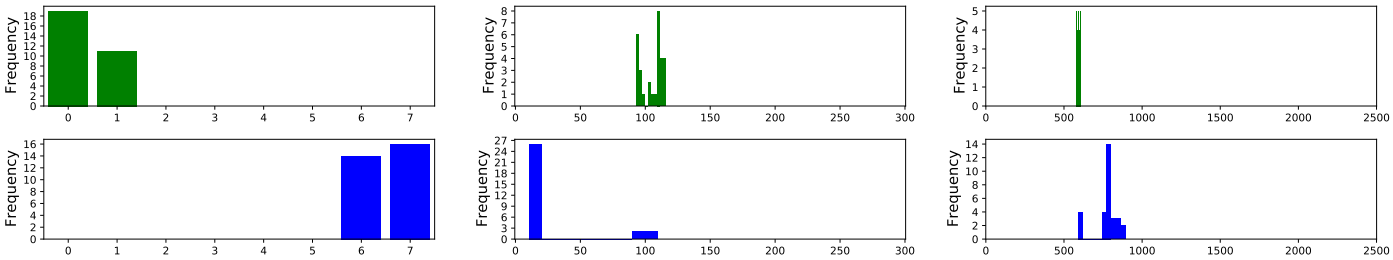


Fig. 11: Results over 30 independent runs for the object detection mission using the best configuration and both CIs

report a higher total final distance for the energy tracking CI. One possible explanation may be that given the variations in position, the faster vehicle was too far away from base when the return command was sent and therefore was unable to return home in time. Another explanation is that in rare cases, when returning home, robot collision avoidance strategies may incorrectly navigate the robots a considerable distance away from their intended home points. These results show the ability of ATLAS to identify potential instability of configurations in rare cases. However, it is up to the user to analyse the logs and determine precisely what happened in each case.

D. Threats to Validity

We limit **construct validity** threats that could be due to assumptions and simplifications when deciding the experimental methodology using case studies that represent common UUV missions. We developed the object detection mission. The Bo-



(a) Number of completed sweeps (b) Final distance from base (c) Residual energy

Fig. 12: Bo-Alpha mission results over 30 independent runs for the standard CI (top) and energy-based CI (bottom)

Alpha mission [28] has been developed by the MOOS-IvP community and is part of the standard version of the simulator.

We reduce **internal validity** threats that could produce incorrect analysis data and lead to deriving incorrect insights by assessing ATLAS using independent research questions and analysing the simulator-produced logs automatically. We support reproducibility of our findings by making our analysis scripts available in the project's repository.

We mitigate **external validity** threats that could reduce the generalisation of ATLAS using established MDE practices and tools like EMF [25], Epsilon [24] and ActiveMQ [22]. The experimental evaluation involved two case studies (one provided by the MOOS-IvP community), reduces further this threat. The early prototype ATLAS middleware for Gazebo/ROS demonstrates the generality of our framework and its capability to support multiple robotic simulators.

VI. Related Work

The work presented lies at the intersection of model-driven engineering and robotics. Developing model-driven solutions for the robotics domain is an established area, which has produced several results over the years [30]–[32]. The majority of the proposed domain-specific modelling languages deals only with specific robot functions such as perception or control, while there are some model-driven toolchains like RobotML [10], BRICS [12], SmartSoft [9], and Robochart [11] which provide multiple modelling notations to be used together when developing a robotic system. For a detailed description of different approaches to model-driven engineering of robots, the reader is referred to [18] and [33].

Despite the available literature on the application of MDE to robotics, the engineering of MRS is still inadequately investigated. Cattivera and Casalaro [34] conducted a systematic mapping study on the application of MDE to the engineering of mobile robots and they found that out of all the studies reviewed, only 19% (i.e. 13 studies out of 69) deal with MRS. The most commonly formalism used for modelling multi-robot behaviour is finite state machines and statecharts (e.g. [35]–[37]). Other approaches include Ciccozzi et al. [31], who propose the FLYAQ family of graphical domain-specific languages to model the structure and behaviour of multi-robot aerial systems, and Pinciroli and Beltrame [38] who propose a textual DSL for specifying the behaviour of robot swarms. Instead of developing a language for specifying the behaviour of multi-robot systems, Dragule et al. [39] extend FLYAQ with a specification language, which enables engineers to specify domain-specific constraints for robotic missions in a declarative manner. Finally, very few approaches propose solutions for modelling explicitly communication, task allocation, and coordination between robots with the exception of [40].

The aforementioned languages and tools focus on the specification of the behaviour and structure of multi-robot systems. To the best of our knowledge, very few approaches focus on the design-space exploration of robotic systems. Saeedi et al. [41] use simulation to tune the parameters of SLAM algorithms, while Christiansen et al. [42] use design space exploration to optimise the configuration of an animal feeding robot. Instead, ATLAS focuses on the exploration and evaluation of different algorithms for the collective intelligence of the robotic robot team. Also, our approach is middleware- and simulator-agnostic, since its flexible, message-based architecture allows it to be easily extended to accommodate experimentation with different robotic platforms.

VII. Conclusions and Future Work

This paper presents the ATLAS framework for design-space exploration and tradeoff analysis of MRS architectures and CI algorithms. It described the framework's architecture, the core concepts of the ATLAS domain-specific language, and the ATLAS middleware. The evaluation of the framework is based on two MRS case studies built with MOOS-IvP [14]. The evaluation indicated that ATLAS is capable of modelling MRS and their missions, and it enables the exploration and tradeoff analysis of different MRS configurations and CI algorithms.

In the future, we plan to evaluate the expressive power of the ATLAS DSL by applying it to more case studies, complete the support for Gazebo/ROS and extend its applicability to different robotic platforms [43]. Also, we would like to improve the variability modelling capabilities of ATLAS, enabling the analysis of more elaborate design alternatives [20] and investigate the incorporation of intelligent techniques to search for optimal MRS configurations [44]–[47].

Acknowledgements: This research was supported by the European Union's Horizon 2020 project SESAME (grant agreement No 101017258) and by the Lloyds Register Foundation under the Assuring Autonomy International Programme grant SAFEMUV.

REFERENCES

[1] L. E. Parker, *Reliability and fault tolerance in collective robot systems*. Pan Stanford Publishing, Singapore, 2012, ch. 6.

[2] S. Gerasimou, R. Calinescu, S. Shevtsov, and D. Weyns, "UNDERSEA: an exemplar for engineering self-adaptive unmanned underwater vehicles," in *SEAMS'17*, 2017, pp. 83–89.

[3] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013.

[4] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *IEEE Access*, vol. 6, pp. 28 573–28 593, 2018.

[5] J. O. Kephart, A. Diaconescu, H. Giese, A. Robertsson *et al.*, "Self-adaptation in collective self-aware computing systems," in *Self-Aware Computing Systems*. Springer, 2017, pp. 401–435.

[6] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Göschka, "On patterns for decentralized control in self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 76–107.

[7] R. De Lemos *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 1–32.

[8] A. Bucchiarone, A. Cicchetti, and M. De Sanctis, "Towards a domain specific language for engineering collective adaptive systems," in *2nd International Workshops on Foundations and Applications of Self* Systems*. IEEE, 2017, pp. 19–26.

[9] C. Schlegel, T. Haßler, A. Lotz, and A. Steck, "Robotic software systems: From code-driven to model-driven designs," in *2009 International Conference on Advanced Robotics*. IEEE, 2009, pp. 1–8.

[10] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi, and M. Ziane, "RobotML, a domain-specific language to design, simulate and deploy robotic applications," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2012, pp. 149–160.

[11] A. Miyazawa, P. Ribeiro, W. Li, A. Cavalcanti, J. Timmis, and J. Woodcock, "RoboChart: modelling and verification of the functional behaviour of robotic applications," *Software & Systems Modeling*, vol. 18, no. 5, pp. 3097–3149, 2019.

[12] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, "The BRICS component model: a model-based development paradigm for complex robotics software systems," in *28th Annual ACM Symposium on Applied Computing*, 2013, pp. 1758–1764.

[13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, vol. 3, 01 2009.

[14] M. R. Benjamin, H. Schmidt, P. M. Newman, and J. J. Leonard, "Nested autonomy for unmanned marine vehicles with MOOS-IvP," *Journal of Field Robotics*, vol. 27, no. 6, pp. 834–875, 2010.

[15] S. Gerasimou, N. Matragkas, and R. Calinescu, "Towards systematic engineering of collaborative heterogeneous robotic systems," in *2019 IEEE/ACM 2nd International Workshop on Robotics Software Engineering*. IEEE, 2019, pp. 25–28.

[16] L. E. Parker, "Multiple mobile robot systems," in *Handbook of Robotics*. Springer, 2008, pp. 921–941.

[17] F. Rossi, S. Bandyopadhyay, M. Wolf, and M. Pavone, "Review of multi-agent algorithms for collective behavior: a structural taxonomy," *IFAC-PapersOnLine*, vol. 51, no. 12, pp. 112–117, 2018.

[18] A. Nordmann, N. Hochgeschwender, D. Wigand, and S. Wrede, "A survey on domain-specific modeling and languages in robotics," *Journal of Software Engineering in Robotics*, vol. 7, no. 1, pp. 75–99, 2016.

[19] K. Pohl, G. Böckle, and F. J. van Der Linden, *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[20] S. Apel, D. Batory, C. Kästner, and G. Saake, "Software product lines," in *Feature-Oriented Software Product Lines*. Springer, 2013, pp. 3–15.

[21] J.-M. Jézéquel, "Model-driven engineering for software product lines," *International Scholarly Research Notices*, vol. 2012, 2012.

[22] ActiveMQ, https://activemq.apache.org/components/classic/.

[23] R. C. Martin, "Design principles and design patterns," *Object Mentor*, vol. 1, no. 34, p. 597, 2000.

[24] D. Kolovos, L. Rose, R. Paige, and A. Garcıa-Domınguez, "The Epsilon book," *Structure*, vol. 178, pp. 1–10, 2010.

[25] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework*. Pearson Education, 2008.

[26] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons, 2012.

[27] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to advanced empirical software engineering*, 2008, pp. 285–311.

[28] MOOS-IvP, "Bo Alpha UUV Mission," https://oceanai.mit.edu/ivpman/pmwiki/pmwiki.php?n=IvPTools.UFldObstacleSim.

[29] A. Afzal, C. Le Goues, M. Hilton, and C. S. Timperley, "A study on challenges of testing robotic systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification*. IEEE, 2020, pp. 96–107.

[30] C. Schlegel, A. Steck, D. Brugali, and A. Knoll, "Design abstraction and processes in robotics: From code-driven to model-driven engineering," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010, pp. 324–335.

[31] F. Ciccozzi, D. Di Ruscio, I. Malavolta, and P. Pelliccione, "Adopting mde for specifying and executing civilian missions of mobile multi-robot systems," *IEEE Access*, vol. 4, pp. 6451–6466, 2016.

[32] F. Ciccozzi, D. Di Ruscio, I. Malavolta, P. Pelliccione, and J. Tumova, "Engineering the software of robotic systems," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 507–508.

[33] E. de Araújo Silva, E. Valentin, J. R. H. Carvalho, and R. da Silva Barreto, "A survey of model driven engineering in robotics," *Journal of Computer Languages*, p. 101021, 2021.

[34] G. Cattivera and G. Casalaro, "Model-driven engineering for mobile robot systems: A systematic mapping study," *Malardalen University*, 2015.

[35] H. Skubch, M. Wagner, R. Reichle, and K. Geihs, "A modelling language for cooperative plans in highly dynamic domains," *Mechatronics*, vol. 21, no. 2, pp. 423–433, 2011.

[36] A. Paraschos, N. I. Spanoudakis, and M. G. Lagoudakis, "Model-driven behavior specification for robotic teams," in *AAMAS*, 2012, pp. 171–178.

[37] J. M. Gascuena, E. Navarro, and A. Fernández-Caballero, "Model-driven engineering techniques for the development of multi-agent systems," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 1, pp. 159–173, 2012.

[38] C. Pinciroli and G. Beltrame, "Buzz: An extensible programming language for heterogeneous swarm robotics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016, pp. 3794–3800.

[39] S. Dragule, B. Meyers, and P. Pelliccione, "A generated property specification language for resilient multirobot missions," in *International Workshop on Software Engineering for Resilient Systems*. Springer, 2017, pp. 45–61.

[40] P. A. Baer, R. Reichle, and K. Geihs, "The spica development framework–model-driven software development for autonomous mobile robots," in *Proceedings of the 10th international conference on intelligent autonomous systems (IAS-10'08)*, 2008, pp. 211–220.

[41] S. Saeedi, L. Nardi, E. Johns, B. Bodin, P. H. Kelly, and A. J. Davison, "Application-oriented design space exploration for slam algorithms," in *2017 IEEE International Conference on Robotics and Automation*. IEEE, 2017, pp. 5716–5723.

[42] M. P. Christiansen, P. G. Larsen, and R. N. Jørgensen, "Robotic design choice overview using co-simulation and design space exploration," *Robotics*, vol. 4, no. 4, pp. 398–420, 2015.

[43] A. Elkady and T. Sobh, "Robotics middleware: A comprehensive literature survey and attribute-based bibliography," *Journal of Robotics*, vol. 2012, 2012.

[44] S. Gerasimou, R. Calinescu, and G. Tamburrelli, "Synthesis of probabilistic models for quality-of-service software engineering," *Automated Software Engineering*, vol. 25, no. 4, pp. 785–831, 2018.

[45] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering," in *30th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2015, pp. 319–330.

[46] C. Henard, M. Papadakis, M. Harman, and Y. Le Traon, "Combining multi-objective search and constraint solving for configuring large software product lines," in *37th IEEE/ACM International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 517–528.

[47] S. Gerasimou, J. Camara Moreno, R. Calinescu, N. Alasmari, F. Alhwikem, and X. Fang, "Evolutionary-guided synthesis of verified pareto-optimal MDP policies," in *36th IEEE/ACM International Conference on Automated Software Engineering*, 2021, in press.