Design, Verification and Validation of Distributed Artificial Intelligence from Simulations to Reality

James Edmondson[1], Christopher Tomaszewski[2], Cormac O'Meadhra[2], Jeffrey Hansen[1], and David Kyle[1]

***Abstract*— Researchers and practitioners of distributed artificial intelligence (AI) face a large number of challenges when building reliable systems for real world applications, especially in outdoor robotics. Distributed AI research presents additional issues beyond the software and hardware problems commonplace in robotics; researchers must manage emergent behaviors, complexity of distributed systems of systems, and verification and validation of distributed AI capabilities against mission, safety, and quality-of-service requirements in outdoor robotics systems. In this paper, we present a design process for mission-focused distributed AI capabilities that includes open source middleware offerings for taking simulated robotics to reality. We then describe a verification and validation process that uses statistical model checking to verify algorithms/AI within confidence intervals in simulated environments before transitioning distributed AI algorithms into real world autonomous surface vehicles in lakes.**

I. Introduction

Researchers and practitioners of distributed artificial intelligence (AI) and multi-agent systems (MAS) face a myriad of issues when developing distributed systems, especially for outdoor robotics applications. In addition to normal communication/localization/hardware problems caused by the environment, there is the more fundamental problem of designing, verifying and validating what is essentially a system of systems. It is difficult enough to design and validate a single robotic system and its interactions with the environment. Adding multiple agents and even swarms of robots that are constantly communicating with each other causes additional complications due to emergent behavior, i.e., properties and behaviors that emerge in the designed system that were not intended and are almost always detrimental towards the mission of the distributed system.

Understanding what a distributed system like a MAS is capable of and how it will operate in a real world environment is made more difficult by the types of primitives MAS developers are forced to use by popular Multi-Agent Software Frameworks (MASF). General purpose robotics software suffers from an algorithmic design problem that is directly tied to how traditional networking middlewares are being developed. Modern MASFs focus on paradigms such as message passing, publish/subscribe and remote procedure calls as the primary development primitives presented to developers. This is a direct result of roboticists and AI researchers harnessing software products from the middleware community in traditional distributed systems development from software packages like ACE, CORBA, MPI, ZeroMQ, DDS, etc.

Harnessing existing tools provides the MAS community with abundant tools for recording data as it passes between agents. Some toolkits, such as ROS or Stage/Player, use this logging to provide developers with the ability to replay events to figure out what went wrong. However, forensic replays are only valuable if researchers and developers can figure out how to use the tools to design a distributed AI in the first place. Additionally, replaying events between two agents is unlikely to be enough for MAS developers to pinpoint causes of emergent behavior or distributed mission failure amongst many agents. For an AI researcher, being an expert in message passing or pub/sub does not readily map to planning movement, implementing a collaborative algorithm or anything remotely AI-focused. The MAS research community would greatly benefit from a system that provides a more intuitive way of designing MASs and a more streamlined approach to verification from concept to simulation and reality, especially as a means of identifying emergent behaviors early and not during deployment.

In the next sections, we will discuss a different method for prototyping MASs for outdoor robotics that moves users away from focusing on message passing primitives and instead uses knowledge-based programming of a MAS. We use the BSD-licensed open source projects Multi-Agent Distributed Adaptive Resource Allocation (MADARA) [10], which provides a distributed knowledge base, and the Group Autonomy for Mobile Systems (GAMS) [8], [9] project to provide algorithm, simulation and hardware platforms for a MAS, built on top of MADARA. We then create an example mission-focused algorithm that can be evaluated in realistic simulations of quadcopters and boats and uses statistical model checking in a project called DEMETER [13] to verify correct operation within a certain confidence interval. Finally, after achieving confidence through large scale simulations, we deploy the GAMS algorithms into a MAS of unmanned surface vehicles deployed in lakes to validate the algorithms in a real world outdoor setting.

II. Related Work

A. MAS Development Tools

Multi-agent systems have benefited from a wide variety of middlewares, tools, and development kits [17], [25]. The more popular offerings available that feature support for simulation to reality transition are Player/Stage/Gazebo [12], [6], [2], JADE [1] and ROS [21], [7]. Player, JADE and ROS include simulation support, and Player provides both

[1]Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA

[2]Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA

a 2-d and 3-d simulation environment (Stage and Gazebo, respectively). Each offering creates TCP or HTTP networking interfaces between robotic agents that work in POSIX operating systems, and each has been used for real-world multi-agent systems. However, applications built with these toolkits are forced into networking models and settings that degrade performance, dependability and predictability in outdoor settings due to blocking behaviors that suffer during even brief communication outages. Because of the blocking communication, it is also very difficult if not impossible to formally verify ROS or Player applications.

In contrast, we are using the MADARA and GAMS middlewares which were built for outdoor, decentralized agents and focus on asynchronous protocols like UDP unicast, broadcast and multicast that recover quickly from network connectivity loss and reconnection because no queues are kept of old data. Unlike ROS, Player and other middleware offerings that focus on message queues, sockets, client/server interactions, pub/sub and explicitly setting up messaging between agents, the GAMS/MADARA paradigm allows multi-agent system developers to focus on knowledge interactions and the logical structure and flow of a finite-state machine or rule-based system, rather than trying to reason about higher-level interactions based on message flows.

B. MAS Validation Techniques

The main limitation of validation techniques for MAS is not that techniques do not exist, but that all work that we know of is being applied to temporal logics [11] or epistemic logics [19], [20] that do not adequately map to existing MAS development tools. Verification and validation are traditionally done in simulations and with unrealistic targets that do not feature noisy environments (e.g., GPS inaccuracy) or prolonged network drops. In fact, most formal methods techniques for distributed algorithms assume perfect communication and may even only model instantaneous shared memory to limit state space explosion and make verification more tractable. One of the more interesting things about using a distributed knowledge base approach like that in MADARA, is that verification tools that require a shared memory model may be applicable to MADARA since it creates a distributed knowledge base that resembles a shared memory model. Other simulation-based formal methods have been based on running large scale multi-agent simulations, analyzing logs, and applying statistical model checking to the logs to retroactively prove something about the system [14].

Probabilistic model checking [24] shows promise in dealing with the high variability and randomness of real-world robotics but in different ways to theoretical constructs. Probabilistic model checking models a system as a finite state probabilistic automaton, generally as a type of Markov Decision Process (MDP) or Discrete Time Markov Chain (DTMC), expressed as a formula in a temporal logic. This is in contrast to statistical model checking [28] and the usage of Monte-Carlo simulations, effectively treating a system like a black box that is being influenced by random variables.

For our work, we decided to use a statistical model checking technique that allows for incorporation of legacy/proprietary systems and modeling a system based on its variability instead of trying to perfectly represent timing and other characteristics as required in MDPs and formal method techniques like software model checking [15], which must exhaustively verify all properties of a system. Specifically, we use a tool called DEMETER that allows us to run massively parallel black-box simulations, aggregate statistical information, and provide certain types of input attribution to identify random variables in the distributed system and environment scenario that may result in mission failure. Applications using GAMS and MADARA have been verified before using software model checking [4], [5], but this is only applied to a synchronous models of computation (SMoC). A SMoC in distributed systems tends to require a barrier between each mission step to assure safety/collision-avoidance in movements throughout participating agents. Though useful in some situations, a SMoC requirement is very vulnerable to communication outages and also does not work in most adversarial conditions where an agent is not participating in the barrier (allowing the adversary to take advantage of slow barriers that prevent responsive movement in participants).

III. Solution Approach

A. Designing MAS Algorithms

To design AI with our approach, users need only think about how agents should make decisions and not on how data will be passed between agents. This is a significant change from the way that distributed AI has typically been developed with tools like ROS [21], [7]. We build on knowledge-based middlewares that handle messaging and communication seamlessly without much user involvement other than indicating unicast, broadcast, or multicast addresses and ports at startup for discovery and transport.

Our software stack includes the Multi-Agent Distributed Adaptive Resource Allocation (MADARA) [10] project for networking, threading and knowledge sharing and Group Autonomy for Mobile Systems (GAMS) [8], [9] for algorithms, simulation and hardware platforms. MAS algorithm developers map out the requirements of an algorithm and how agents need to interact or learn from each other. Specifically, developers identify high-level information that each agent is keying on to make important decisions.

Algorithms can be coded with C++ or Java as a finite state machine (FSM) in a Monitor, Analyze, Plan, Execute with Knowledge (MAPE-K) [16] controller within the GAMS framework of tools. GAMS algorithms contain three implementable function stubs: analyze, plan, and execute. Algorithm methods can either be executed sequentially, which we will focus on in this paper, or they can also be launched in a thread that operates outside of the MAPE-K controller, which we will not cover in this paper. If a user can write down how an agent analyzes, plans and executes in a logical control flow, they can readily adapt this program into a real MAS by filling in one or more of these functions with the

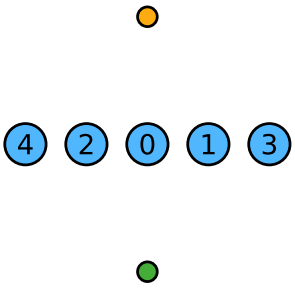


Fig. 1: Example line formation with 5 protectors

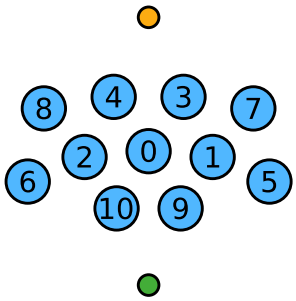


Fig. 2: Example onion formation with 11 protectors

appropriate logic. While any FSM can technically be directly placed into the execute method, breaking up the FSM into analyze and plan stages may help with code legibility and maintainability.

Algorithms in GAMS typically interact with a GAMS platform, which has standardized interfaces for movement and knowledge information like battery levels and general status information about movement. GAMS also includes a pose system that provides translations between arbitrary reference frames (e.g., from a Cartesian grid system in simulations to Latitude and Longitude in the real world). The GAMS code repository includes platforms for simulated quadcopters and boats in the Coppelia Robotics V-REP [22] platform, and there is also support for GAMS on the Platypus LLC Lutra [3], [27] boat platform for water-based sensing. This means that when we design movement-based algorithms for GAMS, we can deploy them on any supported simulation or real-world platform.

B. Motivating Scenario and Implementation

To highlight the development approach and explain later verification and validation stages, we focus on a distributed MAS algorithm that we have developed and deployed. Imagine a photographer trying to take a clear picture of a very important person (VIP). The photographer needs an unoccluded picture, a clear view, so the photo can be published in a media outlet for a fee. The VIP, on the other hand, wants privacy and does not want a sellable picture taken. We call this challenge the Paparazzi Problem.

A solution to this problem requires that the photographer is occluded by some external bodies from having a clear line-of-sight to the VIP. We go about solving this problem by creating a group of quadcopters or other MAS that executes a collaborative algorithm whose sole purpose is to block the line-of-sight of the photographer as the VIP moves around, ruining the photographer's shot and hopefully deterring future photograph attempts.

We implement this as a distributed finite state machine (FSM) in GAMS and MADARA that allows for an arbitrary group of agents to serve as protectors of the VIP. This protector group looks at the position of the VIP and the position of the photographer and arranges itself in a type of formation that is intended to occlude the current line-of-sight between the photographer and VIP and account for immediate movements by the VIP or photographer to flank the protector group. We decided to implement two different versions of protector defense schemes: simple line or arc formations (shown in Figure 1) and a layered defense that we call onion defense (shown in Figure 2).

Simple formations such as in Figure 1 essentially create zones of defense in a Cartesian grid between the VIP (V) and the photographer (A). If three protector agents are present (P0, P1, and P2), then P0 always attempts to move directly between A and V. P1 tries to block line-of-sight to the left of P0, with orientation assuming P0 looking at the photographer A. P2 tries to block line-of-sight to the right of P0. If V or A move left, again with orientation assuming P0 looking at A, then P0 shifts left and P1 and P2 both shift left as they key off the new position of P0. If V or A moves right, then P0 shifts right and P1 and P2 follow its lead. If V or A move forward or backward, up or down, the protector agents P0, P1, and P2 move into new positions accordingly.

Due to page restrictions of this paper, we cannot show the full implementation of this simple formation algorithm, but readers may find the full implementation in the GAMS repository [1]. We will instead describe how the algorithm is encoded. As mentioned in Section III-A, GAMS provides a MAPE-K structure for algorithm implementation where designers fill in an analyze, plan, and execute function where appropriate. Our implementation overloads all three of these functions. In analyze, an agent updates internal variables with current VIP and photographer positions by accessing the appropriate MADARA variables for other agents that are provided by the GAMS framework (e.g., 'agent.vip1.location' and 'agent.photographer1.location'). In plan, the agent uses simple geometry and information about the size of the protection group to determine where it should be in the formation using simple geometry functions like atan2, sin and cos to establish where the agent should move in the execute function. The next position the agent should move to is saved in a variable that is then accessed in the execute function. Any variables modified that are global (in MADARA, any variable that does not start with a period (.) is global), are then automatically sent to other agents in the mission. Note that this is distinctly different from the message micromanagement required in other MASFs like ROS, Player/Stage, etc.

The layered onion formation in Figure 2 is an alteration

[1] http://tinyurl.com/ZoneCoverageSource

of this defensive scheme where an arbitrary depth of simple formations may be layered between V and A. This is useful if the robotic agents P0, P1, and P2 are non-holonomic (i.e., can only move in limited directions), thus opening up line-of-sight if robot orientation is inappropriate for reacting to V and A movements. This is also potentially useful for situations where the environment is causing drifts, e.g., wind in quadcopters or currents in surface vehicles on water. Additionally, this scheme of layered defense can also be useful if A or V have higher velocities or accelerations than the protector group. Essentially, the onion formation adds redundant formations to prevent flanking or holes opening up in defenses due to low holonomicity or environmental factors.

As with the line formation, the layered onion formation is implemented in a GAMS algorithm using the analyze, plan and execute stubs. The layered onion formation uses the plan function to compute its next location based on the current position of the VIP and photographer (e.g., 'agent.vip1.location' and 'agent.photographer1.location'), in a Cartesian grid around an origin at the center of the line between the VIP and the photographer. The agent stores that next location in a MADARA variable, and in the execute function, the agent simply moves to that next location. The analyze function for onion formation is the same as with the simple formations.

Each algorithm created in GAMS can also be paired with a factory method that allows users to call the algorithm remotely by changing the "agent.#.algorithm" variable in MADARA along with "agent.#.algorithm.args" for any arguments needed for the algorithm to start. These algorithms can also be sent and processed by all agents via the "swarm.algorithm" variable. This factory is useful for later demonstrations in the real world and allows us to change algorithms being executed on robotic systems arbitrarily, at any time, but it is not necessarily required if all users need is a simulated MAS. We added a factory method so the code could be included in the GAMS code repository, to make it available to other developers.

In the next section, we evaluate the distributed algorithm in Coppelia Robotics V-REP with the default quadcopter model that is included in the GAMS code repository.

C. Pre-deployment Verification With Confidence

Applications using GAMS and MADARA have been verified before using software model checking [4], [5], but this is only appropriate for synchronous models of computation (distributed systems that barrier between each mission step to assure safety/collision-avoidance in movements). To handle more realistic outdoor environments that must deal with asynchrony in the environment and adversarial agents, we instead use a verification tool called DEMETER [13] based on statistical model checking that evaluates the distributed algorithm within a simulation environment. DEMETER has several features that favor MAS development. First, it integrates readily with distributed GAMS and MADARA applications and allows us to run real-time monitors of the multi-agent system within the network by checking locations of agents and whether or not line-of-sight was prevented with a simple line intersection conditional coded into a MADARA application. Essentially, DEMETER allows us to insert real-time probes into the distributed system to monitor for mission success or failure.

TABLE I: Validation Results for Defensive Algorithms

	Disperse	Detect Range	Failure	Trials
1	Loose	Long	0.11%	265,896
2	Loose	Short	0.35%	114,912
3	Tight	Short	0.28%	114,504
4	Tight	Long	0.00%	400,000+

Second, DEMETER allows us to launch hundreds of thousands of simulations of an arbitrary number of GAMS agents in parallel within isolated Linux containers in a computing cluster to gather statistical metrics with confidence, aggregated across all runs. Third, the tool can identify causes of algorithm weaknesses and strengths through a process called input attribution in regards to randomized environmental variable inputs into the mission and environment. This feature is based on logistic regression to indicate what variable interactions are likely to cause mission failure, so improvements can be made or new insights can be gained about the algorithm. A ROC area-under-the-curve (AUC) metric calculated using five-fold cross validation is used as an overall quality of fit score for the logistic regression model to the actual data. The AUC value ranges between 0.5 for a poor fit to 1.0 for a perfect fit. We consider values above 0.8 to indicate a good fit.

Since GAMS includes support for quadcopter and boat models in the Coppelia Robotics V-REP simulation environment, we used the built-in models for the quadcopter to test the reactivity of the simple formation defenses with 4 protectors in simulated scenarios with 1 mobile photographer and a VIP in five minute simulations. We provided random inputs concerning initial photographer and VIP location, as well as short and long detection ranges for the protectors, and loose and tight initial formations around the VIP to gauge the effectiveness of the algorithm in dealing with readiness before a photographer showed up. These random inputs were tweaked automatically by DEMETER in a type of fuzz testing and then fed into the logistic regression analyses that help identify weaknesses or strengths of algorithms, in regards to effect on mission success. The results of running nearly 1 million experiments targeting a relative error of 5% in DEMETER are shown in Table I.

From DEMETER, we learned that a holonomic robot like a quadcopter running the line or onion defense schemes protected the VIP over 99% of the time for the duration of the five minute simulation, regardless of whether initial detection range was short (a visual range around the VIP) or long (the starting location of the photographer), and whether the initial dispersement of protectors around the VIP was tight (always within 5m of the VIP) or loose (a random starting location between 5m from the VIP up to the starting location of the

TABLE II: Polynomial Logistic Analysis Results for Input Attribution

Name	β	$se(\beta)$	p-value
θ_1^2	0.0939	0.0170	0.0000
θ_0^2	0.0939	0.0173	0.0000
N_P^2	-0.0208	0.0044	0.0000

photographer). There is a stratification of confidence in Table I that shows a tight initial protector formation around the VIP combined with a long detection range has the best mission success rate, and this result makes sense.

Results for statistical correlation using polynomial logistic analysis are shown in Table II. The AUC for this experiment was 0.87 indicating a good fit of the model to the data. θ_0 represents the distance that P0 (the leader) is from V when A is detected. θ_1 represents the distance that P1 starts from V on detection. N_P is the number of protectors. The values are squared here by the automated system within DEMETER that pairs random input variables in the simulation to gauge the impact these had on mission failure. The squares are useful here because they correlate to distance calculations, but the automated system also tried every second order polynomial pairing of all random inputs into the system. These squared values were just the most promising of the options tried in the aggregated statistical results.

The lower the p-value, the more statistically significant the terms are toward mission failure. Out of several dozen pairings of random input variables into the mission scenario, including positional information for all protectors, VIP, and photographer, only the three in Table II are statistically significant (¡ 0.05). β tells us the factor by which the log of the odds for the predicate being satisfied increases with each unit increase, and $se(\beta)$ is the standard error of that factor. More details on this type of logical analysis can be found in [13], including how to replicate the results without DEMETER.

The results of the logistical regression analyses in Table II showed that algorithm success in quadcopters depended on three variables: the initial distance of the lead protector (P0) from V, the initial distance of the immediate neighbor in the direction that the photographer is moving (P1 or P2), and the overall number of protectors. Additionally, DEMETER was able to show a high statistical correlation to success rate being higher when P0 initially started at a location between V and A and not on the other side of V when A was first recognized as a photographer. This makes sense as P0, the agent that blocks current line of sight, being near line-of-sight blocking position should positively affect mission outcome. Essentially, P0 being lucky enough to already be in position means that success chance for the overall protective algorithm is higher. To find this latter result, we had to modify the starting conditions so the photographer always started due north and out of initial photograph range while the protectors were randomly placed around the VIP. By fixing the photographer (A) position, the DEMETER tool could better isolate the significance of protector positions in relation to the initial position of V and A.

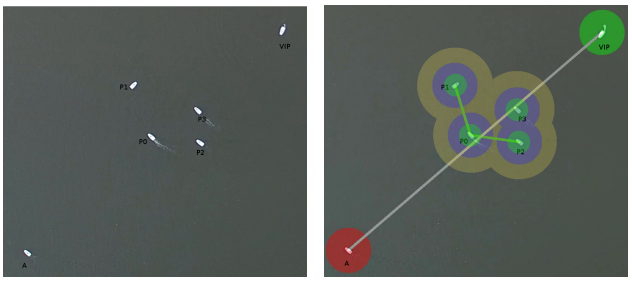
D. Real-World Deployment and Validation

Though the DEMETER results were promising, real-world validation is required to evaluate the performance of the MAS in outdoor robots. Due to difficulty obtaining clearance to fly a group of autonomous quadcopters near university airspace, we chose the Platypus Lutra autonomous surface vehicle (ASV) [23], [26], [18] for field experiments. The team of Lutra ASVs deployed were heterogeneous in computational units and associated components, including both Raspberry Pi 3 and Odroid-XU4 computing boards interfaced with Navio2 and Platypus control boards respectively. Because GAMS/MADARA are portable to ARM and Intel, this heterogeneity was not a problem. A single GAMS platform was implemented to interact with the PID controller on the ASVs via move-to-GPS commands, which map to the GAMS platform's reference-frame-agnostic interface for movement, and then compiled and deployed on both the Raspberry Pi- and the Odroid-based vehicles.

After testing the hardware and software controller stack for the Lutra ASV in the lab, the team of ASVs was moved to the water to replicate the scenarios tested in the DEMETER experiments. One robot was designated the VIP and four others were assigned as its protectors, running the GAMS defensive algorithms outlined in Section III-B. The remaining ASV played the role of the photographer and was given a waypoint following algorithm guiding its movement around the VIP. The VIP was simply allowed to hold its position and trust the protectors to occlude photographs. We deployed a downward-facing 4K camera aboard a DJI Phantom Professional 3 hovering over the lake to capture footage for analysis and algorithm performance evaluation. The experimental setup and sample trajectory of the photographer ASV are shown in Figure 3.

A total of three scenarios were tested: two variations of onion defense and one of line defense. In each scenario, the protectors attempt to position themselves between the photographer and VIP to successfully occlude photos. The photographer circumnavigates the VIP in a clockwise direction as the protectors adjust their position to maintain coverage. In order to evaluate the effectiveness of the protection algorithms and coverage provided by the team, the video footage collected from above was post-processed in OpenCV, using thresholding to segment the boats from the environment. For each experiment, one complete circumnavigation of the VIP by the photographer was considered. The number of video frames where the protectors successfully occluded the photographer's line of sight to the VIP was divided by the total number of frames in the trial to obtain the coverage success rate. Line of sight determination was made by computing the line between the photographer and VIP and then evaluating whether this line passed through the protection region of any protector agent. Using the known size of the ASV in both image and world frames, the desired protection radius around each defensive agent was scaled accordingly. A sample of an original and post-processed

Fig. 3: Experimental Setup



(a) Unprocessed Image (b) Processed Image

Fig. 4: Video Analysis Showing Line of Sight Computation. (a) shows an unprocessed frame with labeled ASVs. (b) shows a processed image with the three protection radii tested around the protectors [P0, P1, P2, P3].

video frame showing the protection radii considered (1.5m, 3m, 5m) and line of sight computation is given in Figure 4. The coverage success rate for all scenarios with stated protection radius settings is presented in Table III.

The results show a clear and unsurprising correlation between the coverage success rate and the size of the protection region. Since the DEMETER experiments were conducted with a protection radius of 5 meters, the corresponding real world results are of particular interest, especially since the real world experiments were using a less mobile, non-holonomic platform rather than the quadcopters in the simulation. Across all the scenarios, the lowest coverage success ratio achieved with this radius was 97 percent, which is in line with the verification results from DEMETER, especially when environmental factors such as localization error from GPS are considered.

The fact that line defense outperformed onion defense is not surprising given that the current implementation of

Scenario	1.5m Radius	3m Radius	5m Radius
Line Defense	0.595859	0.894343	0.995757
Onion Trial 1	0.594300	0.667714	0.973571
Onion Trial 2	0.572000	0.883238	0.981429
Onion Fast	0.827000	0.952000	1.000000

TABLE III: Video Analysis Results

the onion protection scheme requires ten or more agents to achieve proper layering. An interesting result came out of the final scenario tested, where the photographer was faster than the VIP and the protectors. Though we initially expected the algorithm to perform worse in this scenario, the results indicate it outperforms both onion defense and line defense schemes with agents of uniform locomotion capabilities. We suspect that due to the increased speed of the photographer, the protecting agents are typically moving into position rather than holding their assigned positions as is more often the case in the slower photographer scenario, allowing for better coverage of the VIP.

Because the boats are non-holonomic, when they drift out of their assigned formation positions they will often circle back, thereby opening up gaps in the coverage. This effect could be mitigated in the future by improving ASV station keeping with more intelligent maneuvers and behavior. Another factor that affected performance was the reactionary nature of the algorithms; the lead defender occasionally would not react fast enough to a moving photographer, allowing for a line of sight to open up to one side of the protector formation. This contingency can be better accounted for in future versions of the defensive algorithms by anticipating photographer movements based on past movements, but the current version is sufficient for occluding the VIP from the photographer agent at the 5 meter protection radius. This claim is supported with both statistical confidence through the DEMETER results and through the post-processing of real-world demo videos via OpenCV.

IV. Conclusion

In this paper, we have described a methodology based on open source middlewares and tools in GAMS and MADARA to design a mission-focused distributed artificial intelligence capability, the application of statistical model checking with input attribution in the DEMETER tool to large scale simulations of the group-based algorithm we designed, and a successful transition of the distributed algorithm to the real world in unmanned surface vehicles developed by Platypus LLC. We forensically analyzed real world performance of the algorithm with OpenCV in video feeds to isolate agents and evaluate their performance versus their objectives.

Future work may focus on scale of cooperating agents, additional photographers and VIPs, predictive algorithms that do not just react to adversaries, and better statistical model checking tools and techniques for evaluating algorithms and robotics platforms before deployment. Additionally, we believe more rigorous verification techniques like software model checking may be applicable to asynchronous models

of computation, though breakthroughs in reducing state space may be necessary for such tools to be useful.

REFERENCES

[1] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. Jade: A software framework for developing multi-agent applications. lessons learned. *Information and Software Technology*, 50(1):10–21, 2008.

[2] G. Biggs, R. Rusu, T. Collett, B. Gerkey, and R. Vaughan. All the robots merely players: History of player and stage software. 2013.

[3] J. Blum. Connecting run-time metrics to outcome performance of team attack and defense. Master's thesis, Carnegie Mellon University Pittsburgh, PA, 2016.

[4] S. Chaki and J. Edmondson. Model-driven verifying compilation of synchronous distributed applications. In *International Conference on Model Driven Engineering Languages and Systems*, pages 201–217. Springer, 2014.

[5] S. Chaki and J. Edmondson. Toward parameterized verification of synchronous distributed applications. In *Proceedings of the 2014 International SPIN Symposium on Model Checking of Software*, pages 109–112. ACM, 2014.

[6] T. H. Collett, B. A. MacDonald, and B. P. Gerkey. Player 2.0: Toward a practical robot programming framework. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005)*, page 145, 2005.

[7] S. Cousins and B. Gerkey. Milestones: First roscon and osrf [ros topics]. *Robotics & Automation Magazine, IEEE*, 19(3):14–15, 2012.

[8] A. Dukeman, J. A. Adams, and J. Edmondson. Extensible collaborative autonomy using gams. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 281–283. ACM, 2016.

[9] J. Edmondson, S. Chaki, J. Hansen, and D. Kyle. Software solutions for distributed autonomous multi-functional robotics in space. In *AIAA SPACE 2016*, page 5327, 2016.

[10] J. Edmondson and A. Gokhale. Design of a scalable reasoning engine for distributed, real-time and embedded systems. In *International Conference on Knowledge Science, Engineering and Management*, pages 221–232. Springer, 2011.

[11] M. Fisher and M. Wooldridge. On the formal specification and verification of multi-agent systems. *International Journal of Cooperative Information Systems*, 6(01):37–65, 1997.

[12] B. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, volume 1, pages 317–323, 2003.

[13] J. P. Hansen, S. Chaki, S. Hissam, J. Edmondson, G. A. Moreno, and D. Kyle. Input attribution for statistical model checking using logistic regression. In *International Conference on Runtime Verification*, pages 185–200. Springer, 2016.

[14] B. Herd. *Statistical runtime verification of agent-based simulations*. PhD thesis, KINGS COLLEGE LONDON, 2015.

[15] R. Jhala and R. Majumdar. Software model checking. *ACM Computing Surveys (CSUR)*, 41(4):21, 2009.

[16] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[17] J. Kramer and M. Scheutz. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, 22(2):101–132, 2007.

[18] P. LLC. Platypus lutra platform description, 2016.

[19] A. Lomuscio, H. Qu, and F. Raimondi. Mcmas: A model checker for the verification of multi-agent systems. In *International Conference on Computer Aided Verification*, pages 682–688. Springer, 2009.

[20] W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 209–216. ACM, 2003.

[21] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[22] E. Rohmer, S. P. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE, 2013.

[23] P. Scerri, P. Velagapudi, B. Kannan, A. Valada, C. Tomaszewski, J. Dolan, A. Scerri, K. S. Shankar, L. Bill, and G. Kantor. Real-world testing of a multi-robot team. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1213–1214. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[24] M. Stoelinga. *Alea jacta est: verification of probabilistic, real-time and parametric systems*. PhD thesis, University of Nijmegen, 2002.

[25] R. Trillo, S. Ilarri, and E. Mena. Comparison and performance evaluation of mobile agent platforms. In *Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on*, pages 41–41. IEEE, 2007.

[26] A. Valada, P. Velagapudi, B. Kannan, C. Tomaszewski, G. Kantor, and P. Scerri. Development of a low cost multi-robot autonomous marine surface platform. In *Field and Service Robotics*, pages 643–658. Springer, 2014.

[27] G. A. Wilde, R. R. Murphy, D. A. Shell, and C. M. Marianno. A man-packable unmanned surface vehicle for radiation localization and forensics. In *2015 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6. IEEE, 2015.

[28] H. L. S. Younes, M. Z. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking. *STTT*, 8(3), 2006.