Automatic Verification of Autonomous Robot Missions

Matthew O'Brien[1], Ronald Arkin[1] Dagan Harrington[2], Damian Lyons[2], and Shu Jiang[1]

[1] School of Interactive Computing, Georgia Tech, Atlanta, GA 30332
{mjobrien,arkin,sjiang}@gatech.edu
[2] Computer & Information Science, Fordham University, Bronx, NY 10458
{dharrington5,dlyons}@fordham.edu

Abstract. Before autonomous robotics can be used for dangerous or critical missions, performance guarantees should be made available. This paper overviews a software system for the verification of behavior-based controllers in context of chosen hardware and environmental models. Robotic controllers are automatically translated to a process algebra. The system comprising both the robot and the environment are then evaluated by VIPARS, a verification software module in development, and compared to specific performance criteria. The user is returned a probability that the performance criteria will hold in the uncertainty of real-world conditions. Experimental results demonstrate accurate verification for a mission related to the search for a biohazard.

Keywords: mobile robots, formal verification, performance guarantees, automatic translation

1 Introduction

Mission assurance by providing formal methods for assessing performance guarantees is a well identified need and crucial area of research for autonomy. This is essential in missions that must get the job done right the first time where there is no tolerance for failure. We have been focusing in particular on addressing search and remediation tasks for countering Weapons of Mass Destruction (C-WMD), e.g., biological, chemical, radiological or nuclear agents that might be posed by terrorist activities.

A variety of methods, historically based on model checking (e.g., [4],[9]) have been developed for robot performance guarantees and synthesizing provably correct controllers that have met with some success. But there remain problems associated with the scalability of these methods and their utility in continuous valued domains, typical of robotic sensing and actuation [10]. Our research, conducted for the Defense Threat Reduction Agency (DTRA), takes a different approach, utilizing process algebras as the basis for the representation as opposed to the temporal logics so often used in model checking. We feel this provides a better match for the requirements of real-time autonomy, and have had success in its application on a range of robotic missions: single robot waypoint [11], multi-robot bounding overwatch [14] and search and explore [7].

To accomplish this we have had to bridge the gap from automatically generated robot control software that is represented in the Configuration Network Language (CNL), a component of the MissionLab Mission Specification System used in our research [18]. This CNL code must then be processed by our verification module (VIPARS Verification in Process Algebra for Robot Schemas) to yield the performance guarantees and predictions necessary for informing the operator regarding the likelihood of success of her mission. Thus CNL must be translated to PARS (Process Algebra for Robot Schemas). Until now this translation has been performed manually, but as described in this paper this central task linking the control software to the verification module is now automatically translated, providing end-to-end operational capability. This paper describes how that transformation has been implemented and tested.

2 Related Work

Formal verification of systems is critical when failure creates a high cost, such as life or death scenarios. A variety of software validation techniques have been developed for applications from airplanes to medical devices. If the use of robots is to expand to similarly critical applications, verification techniques must be developed to meet this challenge. The embodied nature of robot software brings several additional complications. The real world is dynamic, unstructured, and continuous; making modeling difficult. In addition, information about the world provided by sensors is incomplete and noisy. The problem of adapting verification techniques for this domain has been approached in several ways.

One of the main methods of software verification is model checking [6],[19]. In model checking, the system is represented as a finite state automaton and formal specification in a modal logic. All states are explored and compared against the formal specifications or properties. The continuous nature of a robots workspace creates a state space far too large for traditional methods. This is commonly referred to as state explosion, and is a major focus of research in model checking. One technique applied to this problem is symmetry reduction. By determining symmetries, the number of states that must be checked can be reduced. Under ideal conditions, significant reductions can be made. Identifying symmetries may be a difficult task however, and is often dependent on the programming language used [19].

An alternative approach is to synthesize a valid controller given a robot model and a set of specifications. Linear temporal logic (LTL), or a restricted subset of LTL, GR(1), has been used to represent specifications in a way that allows for automatic generation of controllers [4],[9]. Effective motion planning has been demonstrated with this technique. Like model checking, these LTL based controller synthesis techniques can suffer from state explosion under certain circumstances. In addition, it is not clear if LTL techniques can extend to more sophisticated missions outside the realm of motion planning.

This paper presents yet another approach to verification. Process algebras (PA) model parallel or distributed systems and reason about their properties through algebraic techniques [3]. While originally developed for software systems, applications range from robotics to biology [8],[5]. Process Algebra for Robot Schemas (PARS) is a specification language capable of representing software, hardware, and the environment as interacting processes [10]. The following sections overview the software system in development to utilize PARS, the methods of verification, and results from physical implementation.

3 Methods

3.1 System Overview

In the complete robot mission design system, an operator specifies the robot controller in MissionLab. This controller is compiled first into configuration network language (CNL), an internal language of MissionLab. This CNL code is then translated to PARS. The operator also specifies models of the robot and environment, as well as desired performance criteria. VIPARS evaluates the complete system and returns the results to the operator, creating a feedback loop. If performance is unacceptable the operator refines his or her design. This may entail revising the controller or selecting new hardware for implementation. Once the criteria are satisfied, MissionLab creates an executable for the selected platform. Figure 1 shows an overview of this architecture.

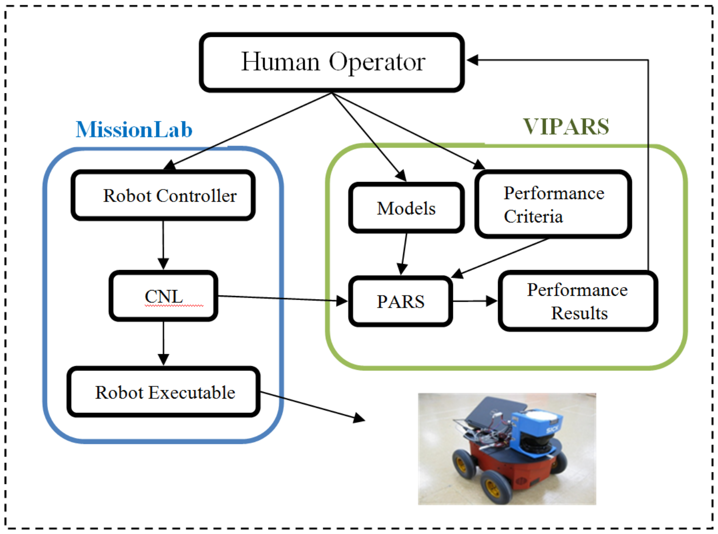


Fig. 1. Overview of system archiecture

3.2 MissionLab and CNL

MissionLab is a software package developed in the GT Mobile Robotics Lab. MissionLab allows users to design missions and robotic controllers graphically, allowing for quick implementation of control schemes without the need for programming experience or concern with low-level details. It incorporates simulation of missions as well as the ability to compile controllers for execution on hardware. The Configuration Network Language (CNL) is one representation of robotic controllers used in MissionLab. CNL is a superset of C++ developed to separate a behaviors implementation from its integration with other behaviors [16].

In CNL, all behaviors for any robotic controller are specified as assemblages of more primitive nodes. Currently, all implemented behaviors are schema-based, from the AuRA architecture [1]. Any node may have a variety of inputs, but only a single output. Primitive nodes only take input from sensors. Assemblages of these nodes are constructed by feeding the output of these primitives into new nodes. Through various arbitration schemes, a single output command is determined and sent to the robot for execution. The implementation of a CNL nodes internal processing is done via traditional C++. See Figure 2 for an example network.

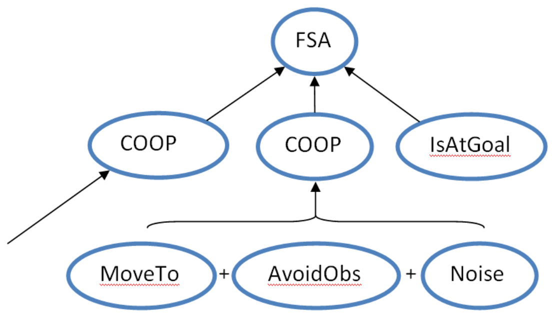


Fig. 2. Part of a CNL node network. Nodes, such as MoveTo, take input from more primitive nodes. The node COOP performs vector summation from all of its inputs, creating a new behavior. The node IsAtGoal is a trigger, tracking conditions to change states. The Finite State Automaton (FSA) selects the output of one behavior for execution on the robot.

3.3 PARS

Process algebras are specification languages that allow for formal verification of concurrent systems. Process Algebra for Robot Schemas (PARS) is a language developed to adapt these techniques to robotic systems [10]. PARS is capable of representing a robotic controller, hardware, the environment, and the interactions between them. A process P is called with initial parameters u1,u2,,un, resulting values v1,v2,,vq, input ports i1,i2,,im, and output ports o1,o2,,op.

$$P\langle u_1, u_2, , u_n\rangle(i_1, i_2, , i_m)(o_1, o_2, , o_p)\langle v_1, v_2, , v_q\rangle \tag{1}$$

Table 1. The basic PARS processes

Process	Stop Condition	Abort Condition
$Delay\langle t\rangle$	After time t	If forced by #
$Ran\langle d\rangle\langle v\rangle$	Returns random sample v from a distribution d	If forced by #
$In\langle p\rangle\langle v\rangle, Out\langle p, v\rangle$	Performs input and output of value v on port p	If forced by #
$Cond\langle op, a, b\rangle$	$a = b, a \neq b, a > b, etc \ldots$	Otherwise

All PARS processes are formed by composition of several basic processes shown in Table 1. This is achieved by three operators. The sequential/conditional operator (;) allows the next process to start if the first process stops, but not if it aborts. For example, a simple process may take in a value on one port, and afterwards output this value on a second port.

$$Pass = In\langle port1\rangle\langle value\rangle; Out\langle port2\rangle\langle value\rangle \tag{2}$$

In addition the concurrent composition (|) and the disabling composition (#) operators allow for processes to run simultaneously. The disabling composition will abort all other processes when one process terminates (stops or aborts). The concurrent composition operator will allow all processes to terminate independently. This notation is sufficient to build complex behaviors via assemblages of simpler processes, analogous to the methodology in MissionLab. A simple example mission constructed with only sequential composition is shown in Figure 3.

```
Mission<w, i> = Goto<w(i)> ; Neq<i ,n> ; Mission<w, i+1>
Goto<a> = TurnTo<a> ; MoveTo<a>
MoveTo<g> = In<p><r> ; Neq<r ,g> ; Out<v ,u(g−r)> ; MoveTo<g>
TurnTo<g> = In<p><r> ; Out<h ,d(g−r)>
```

Fig. 3. Simple PARS Mission

A final critical ability for PARS is looping. Tail-recursion, a process calling itself at the end of its execution, is the method chosen. The Mission process in Figure 3 provides an example of tail-recursion. The restriction of all processes to tail-recursion can allow for more efficient verification (see Section 3.5).

3.4 Translation

The automation of the translation of a robot controller into PARS is a critical step to creating a usable software system. By automating the translation of a robot controller to PARS, and incorporating the VIPARS verification module into MissionLab, a nave user could leverage the formal verification techniques in the field. In addition to improving usability, automation ensures the accuracy and reliability of the final translation. Translation can be a challenging problem. In a model-checking approach the first step is the translation of the program to be verified into a transition system, the formal structure in which verification occurs. However, this translation into a transition system is one of the key points at which state-space explosion can occur [2].

Two sets of lexes (lexical analyzers) and grammars are required to parse a CNL file. The first set is for the CNL code; which defines the CNL network and structures a nodes definition. The final set parses the C++ code inside a node definition. This is only required for the switch statement inside the FSA node. Therefore a subset of the C++ grammar, along with some unique tokens, is adequate. The common compilation tools Flex and Bison were used to produce the final scanner and parser.

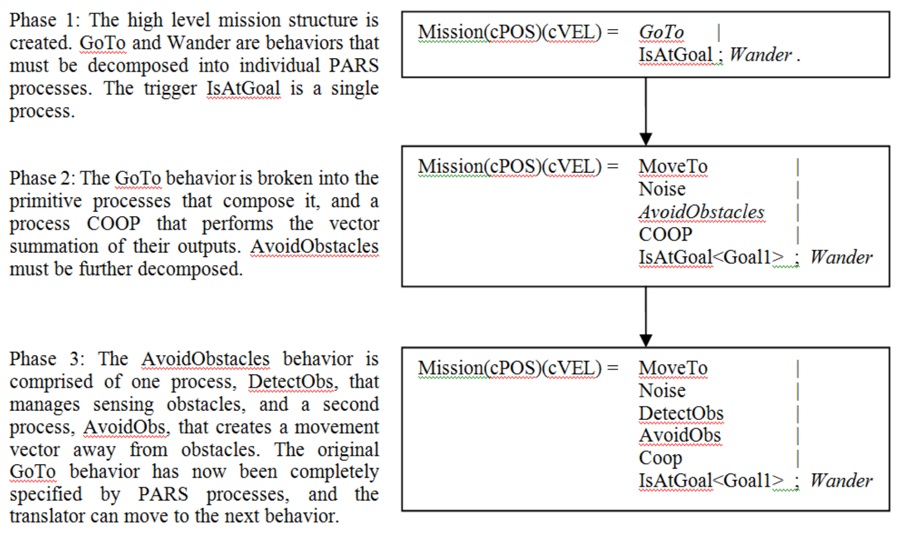
MissionLab is a behavior-based programming environment where users create complex actions from a library of primitive nodes. A matching library of PARS implementations was created. The translator inputs the PARS definition of any node used by the robot controller into the final PARS file. These processes are later called inside the Mission process in a similar manner to how functions are defined and called in programming languages.

The heart of the translation from CNL to PARS is the creation of the Mission process. PARS code matching the structure of the FSA must be created. This structure can be, in general, any finite state automaton. A PARS implementation of a switch statement was created to represent any mission. However, many missions are linear in nature, and the more complex Mission process structure is not required. By checking certain properties of the triggers in the FSA node, the translation software can determine if a mission is linear, and select the appropriate Mission process design. An example linear mission structure in PARS is below in Figure 4. This structure was utilized for the experimental verification described in section 4.

```
Mission(cPOS)(cVEL) =    Behavior1(cPOS)(cVEL)              |
                 Trigger1 ; Behavior2(cPOS)(cVEL)           |
                 Trigger2 ; Behavior3(cPOS)(cVEL)           |
                 Trigger3 ; Behavior4(cPOS)(cVEL)           .
```

Fig. 4. Linear mission process

This initial mission structure is a high level representation of the controller. As discussed before, in MissionLab the highest-level behaviors are assemblages of more primitive nodes. The PARS processes representing lower-level CNL nodes must be created as well. Before the translation software creates a process for a high level behavior in PARS, any additional primitive processes required are created first. For some CNL nodes, this is hardcoded, while for others the number and type of input nodes can vary. The PARS operators needed to coordinate these new processes are also created at this time. As each process calls for the creation of lower level processes, the node network is traversed from top to bottom, creating PARS code that accurately matches the original node network defined in CNL. Figure 5 provides a simple demonstration of this process.


Phase 1: The high level mission structure is created. GoTo and Wander are behaviors that must be decomposed into individual PARS processes. The trigger IsAtGoal is a single process.

Phase 2: The GoTo behavior is broken into the primitive processes that compose it, and a process COOP that performs the vector summation of their outputs. AvoidObstacles must be further decomposed.

Phase 3: The AvoidObstacles behavior is comprised of one process, DetectObs, that manages sensing obstacles, and a second process, AvoidObs, that creates a movement vector away from obstacles. The original GoTo behavior has now been completely specified by PARS processes, and the translator can move to the next behavior.

```
Mission(cPOS)(cVEL) =   GoTo       |
                        IsAtGoal ; Wander .
```

```
Mission(cPOS)(cVEL) =   MoveTo              |
                        Noise               |
                        AvoidObstacles      |
                        COOP                |
                        IsAtGoal<Goal1> ;  Wander
```

```
Mission(cPOS)(cVEL) =   MoveTo              |
                        Noise               |
                        DetectObs           |
                        AvoidObs            |
                        Coop                |
                        IsAtGoal<Goal1> ;  Wander
```

Fig. 5. Initial steps taken to create the complete controller from the PARS implementation of a simple mission FSA. In this example all new processes are executed with concurrent composition, but in general any PARS operators can be selected.

3.5 VIPARS and Validation

The entire system to be verified by VIPARS can be expressed, generally, as:

$$SYS = Env\langle initparams\rangle(vel)(pos)|Mission\langle initparams\rangle(pos)(vel). \quad (3)$$

Which is the concurrent, communicating composition of any number of Controllers (i.e., Mission) and Environment models (i.e., Env). The SYS process in (3) represents a very simplistic situation where the Mission process takes as input a position and outputs a velocity. The Environment process (which currently includes the robot hardware) concurrently inputs a velocity and outputs a position. A simple environment process is broken down in equation (4) to provide an example. Initially, three processes run in parallel. $At\langle r\rangle$ represents the current robot position r in VIPARS. $Odo\langle r\rangle$ represents the odometry sensor, which repeatedly transmits current location information with a normal sensor noise distribution Φ. The $Delay\langle t\rangle$ process ends this group after time t. After words the current velocity, from port v, is combined with the actuator's normal noise distribution Θ to update the robots position. While this is a simple kinematic model, more complex models can include dynamics, battery life, or other properties of the hardware and real world.

$$Env = (Delay\langle t\rangle \# Odo\langle r\rangle \# At\langle r\rangle); Ran\langle\Theta\rangle\langle z\rangle; In\langle v\rangle\langle u\rangle; Env\langle r + (u+z)t\rangle \quad (4)$$

$$Odo = Ran\langle\Phi\rangle\langle e\rangle; Out\langle p, r+e\rangle; \langle r\rangle Odo\langle r\rangle \quad (5)$$

Recall that the robot program must operate and interact with a continuous, unstructured and dynamic environment. This effectively rules out a purely state-based method for verification, such as model checking, where the well-known state-space explosion problem leads to intractable state graphs. We leverage the reactive, recurrent nature of behavior-based robot programs (a behavior-based robot will continually respond to a fixed set of affordances in the environment) to isolate regularities in the combined state-space of Env and Mission. This regularity allows verification to be carried out in a very efficient manner. To make clear the method by which we extract and analyze these periodic regularities in the state-space, first recall that PARS supports iteration in the form of tail-recursion (TR):

$$T\langle v\rangle\langle ...\rangle = P\langle v\rangle\langle u\rangle; T\langle ...\rangle\langle f(v,u)\rangle. \quad (6)$$

The process T in (6) is TR iff its body, P, is a sequential composition of non-recursive processes. In standard TR fashion, input parameters (v) are transformed by some function f, for each successive execution of the process. In [15], we developed an interleaving theorem, a relation between parallel and sequential operations in a process algebraic framework, which allows us to express a parallel, communicating composition of TR processes as a single TR process:

$$SYS = P_1|P_2|\dots|P_m = S(P_1, P_2, \dots, P_m); SYS \quad (7)$$

where $S(P_1, P_2, \dots, P_m)$ is the System Period process that is constructed from an analysis of the scope and communication structure of component processes in Sys. This allows us to recast the analysis of the recurrent system to the analysis of some sequential ordering (using a Maximum Likelihood approach in the case that SYS contains processes with probabilistic behavior) period processes $S(P_1, P_2, \dots, P_m)$.

Once the periodic nature of the concurrent system is determined, VIPARS produces a set of equations called flow-functions by analyzing the port connectivity and TR-transformations of variables for each $P_i \in S(P_1, P_2, \dots, P_m)$. These flow-functions relate values in the network of the kth time step to values in the network of the (k+1)th time step [12]. The flow functions are used to build a Dynamic Bayesian Network (DBN), and verification is carried out by applying a filtering algorithm to the DBN and monitoring for achievement of the performance specification. For a more detailed discussion on this verification process, the authors recommend [14].

The VIPARS system computes within the network of flow-functions and assesses whether performance criteria are met, given the environment model(s) and controller(s) provided by the operator [13]. The VIPARS verification module provides output in the form of: (1) A Boolean answer of whether the performance criteria are met, and (2) detailed output that allows for iterative refinement of the controller. The environment models can be culled from libraries of robot and sensor models. The current work of automatic translation drastically reduces the need for operator intervention in the MissionLab + VIPARS verification system.

4 Validation

To evaluate the verification, experimental results from a physical implementation are compared to the predicted (verification) results from VIPARS. The metric used for comparison is the success rate of a mission. The general procedure, given a mission, is to first develop an appropriate controller in MissionLab. This controller will both be compiled to a hardware executable for the robot, and translated into PARS for verification. Appropriate models of the chosen hardware platform are imported to the VIPARS system. Performance criteria (such as a time limit and spatial accuracy) are selected and given to VIPARS as well. Results from the VIPARS verification and the physical experiment are compared.

A mission related to the search for a biohazard has been selected. This missions were previously verified with manual CNL to PARS translations in [7]. Here, we reproduce those results using automatic translations. A Pioneer 3-AT was the chosen hardware platform.

4.1 The Search Mission

This mission simulates the search for a target, in this case a potential biological weapon. The controller used is shown in Figure 6. The robot explores until the target is found. A camera was used for detection of the target and a SICK laser scanner for obstacle detection and avoidance. In this test, the target was represented as an orange bucket. Once detected, the robot moves to the target and stops. No counter-measure actions were simulated. This provides the opportunity to test random search behavior as well as object detection within the framework of VIPARS. A time limit of 60 seconds to locate the target was chosen as the performance criteria.

The mission was executed on a physical system 106 times. Due to the randomized search pattern, a large number of trials were used to yield accurate results. Failures occurred when the search pattern did not explore near the target within the time limit.

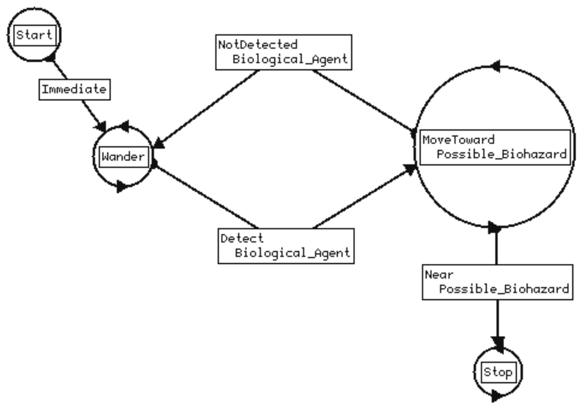


Fig. 6. MissionLab controller for the search mission, shown as it is displayed in the Cfgedit graphical programming tool.

4.2 Results

The primary method of validation is the comparison between empirical success probabilities and the predicted success probabilities. Table 2 lists the results for the mission. While the original manual translation showed strong results, 85% versus 83% success rate, the new results still showed qualitative improvement, matching the experimental validation at 83%.

Assuming the null hypothesis is that the probability of success is actually 85%, and the alternative hypothesis that it is smaller, one can use a z-statistic proportion test and calculate a value of z = 0.58, and $P(Z < z) = 0.28\%$. Therefore we cannot claim that the improvements are statistically significant, even though the prediction is more accurate. We can conclude that the validity and significance of the VIPARS performance guarantee, originally demonstrated in [7], still holds.

Table 2. Final probability results for the mission

Mission	Total Runs	Experimental P(Success)	VIPARS Manual P(Success)	VIPARS Auto P(Success)
Explore	106	83%	85%	83%

The automatic translation produced exactly the same PARS structure for this mission as the manual translation. However, the verification results reported here differ slightly. The most important contribution to this difference is that the automatic translation used a set of PARS processes built to more directly model the CNL nodes than processes used in the manual translation. The quality of the verification therefore depends on the accuracy with which the PARS primitives model their corresponding CNL nodes.

Selecting a small set of relatively low-level behaviors simplifies the reliance on this correspondence. This takes advantage of the behavior-based controller design. Simple nodes in CNL can be implemented directly in PARS, and the complex behavior can be modeled by implementing the CNL network structure in PARS. An alternate approach is to separately verify these PARS processes. This has the advantage of allowing high-level behaviors that may not easily decompose into simple primitive processes. Both options will be considered in ongoing work.

5 Conclusion

The system described in this paper allows a user to design a robot controller graphically, select hardware for implementation, and evaluate the effectiveness of the system in a chosen environment against specific performance criteria. This information can be used in multiple ways: to refine controller design, to evaluate hardware choices, and to inform the operators decision to execute the mission.

Experimental results were used to validate the verification software. The correspondence between empirical and predicted success probabilities was shown to be very accurate. In addition, these predictions were made with automatically generated PARS files; removing the need for any manual translation.

Some readers may observe that with appropriate models of a robot and the environment, one could run randomized simulations to achieve similar probabilistic results. This would be analogous to sample-based planning, but for verification. The method described in this paper is, in contrast, deterministic and returns complete results (such as the probability of being at any location) without multiple executions of the mission. For many classes of problems, this will more accurate and efficient.

The research presented is being extended in several directions to better test the capabilities and limits of the VIPARS system. Development and testing for multi-agent teams, both homogeneous and heterogeneous, has begun. Testing with two Pioneer 3-ATs has been performed, and future plans include introducing a quad-rotor into the team. SLAM will be utilized in the future to verify with more recent navigation techniques. Finally, the translation software and VIPARS system will be fully incorporated into MissionLab. User studies will be performed to demonstrate that with this system, a nave user can leverage the formal verification tools when designing a mission [18].

Acknowledgments

This research is supported by the Defense Threat Reduction Agency, Basic Research Award #HDTRA1-11-1-0038.

References

1. Arkin R, Balch T, (1997) AuRA: Principles and practice in review. Journal of Experimental & Theoretical Artificial Intelligence 9(2-3):175-189.

2. Baeir C, Katoen J (2008) Introduction to Model Checking. Cambridge MA: MIT Press
3. Baeten J (2005) A brief history of process algebra. Theoretical Computer Science, 335:131146
4. Belta C, (2010) Synthesis of provably-correct control and communication strategies for distributed mobile systems. In: ICRA Workshop on Formal Methods, Anchorage Alaska
5. Guerriero M, Heath J, Priami C (2007) An automated translation from a narrative language for biological modelling into process algebra. In: CMSB 2007, LNBI 4695, pp. 136151
6. Jhala R, Majumdar R (2009) Software Model Checking. ACM Computing Surveys 41(4) 21:53.
7. Jiang S, Arkin R, Lyons D, Liu T-M, Harrington D (2013) Performance guarantees for C-WMD robot missions. In: Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium, p 1-8
8. Karaman S, Rasmussen S, Kingston D, Frazzoli E (2009) Specification and planning of UAV missions: a Process Algebra approach. In: American Control Conference, St. Louis MO., 10-12 June 2009
9. Kress-Gazit H, Fainekos G.E, Pappas G (2009) Temporal-Logic-Based Reactive Mission and Motion Planning. IEEE Transactions on Robotics 25(6):1370-1381
10. Lyons D, Arkin R (2004) Towards performance guarantees for emergent behavior. Robotics and Automation. In: IEEE International Conference on Robotics and Automation, vol. 4, pp. 41534158
11. Lyons D, Arkin R, Nirmal P, Jiang S, Liu T-M, Deeb, (2013) Getting it right the first time: Robot mission guarantees in the presence of uncertainty. Intelligent Robots and Systems (IROS) pp 5292-5299
12. Lyons D, Arkin R, Jiang S, Liu T-L, Nirmal P, Deeb J (2013) Performance Verification for behavior-based Robot Missions. AAMAS ARMS 2013 Workshop on Autonomous Robotics and Multirobot Systems, St. Paul MN, May 2013.
13. Lyons D, Arkin R, Jiang S, Nirmal P, Liu T-L (2013) A Software Tool for the Design of Critical Robot Missions with Performance Guarantees. Conf. on Systems Engineering Research (CSER13), Atlanta, GA, March 2013
14. Lyons D, Arkin R, Jiang S, Harrington D, Liu T-L (2014) Verifying and Validating Multirobot Missions. (Submitted) Available via GT Mobile Robot Lab `http://www.cc.gatech.edu/ai/robot-lab/online-publications/GIRTFT_IROS_2014_v5.pdf` Accessed 10 June 2014
15. Lyons D, Arkin R, Nirmal P, Jiang S (2012) Designing Autonomous Robot Missions with Performance Guarantees'. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Algarve, PT
16. MacKenzie D, (1996) The configuration network language user manual. In: Georgia Tech Mobile Robot Lab. `http://www.cc.gatech.edu/aimosaic/robot-lab/research/MissionLab/mlab_manual-7.0.pdf`. Accessed 10 June 2014
17. MacKenzie D, Arkin R, Cameron J, (1997) Multiagent mission specification and execution. Autonomous Robots 4(1):29-52
18. MacKenzie D, Arkin R (1998) Evaluating the Usability of Robot Programming Toolsets. International Journal of Robotics Research, 4(7):381-401
19. Simmons R, Pecheur C, Srinivasan G (2000) Towards automatic verification of autonomous systems. In: Intelligent Robots and Systems, IEEE/RSJ International Conference, vol 2. p1410-1415
20. Xu Chu Ding, Kloetzer M, Chen Yushan, Belta C (2011) Automatic Deployment of Robotic Teams. Robotics & Automation Magazine, IEEE , 18(3)75-86