Software Solutions for Distributed Autonomous Multi-Functional Robotics in Space

James Edmondson, Sagar Chaki, Jeff Hansen, David Kyle
{jredmondson,chaki,jhansen,dskyle}@sei.cmu.edu
Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA USA 15213

1. Abstract

Robotics systems in space must deal with a host of software challenges in addition to and amplified by the challenges of a hostile space environment and the remoteness in which they operate. Software automation has been used to good effect in managing some of these challenges to control single robotic systems, even multi-functional systems, in previous unmanned missions without the physical presence of human operators. However, these previous control software artifacts tend to be platform-specific, single-system-focused and not really applicable to multi-agent teams of multi-functional, multi-generational robotic systems. In this paper, we discuss a software architecture based on the Multi-Agent Distributed Adaptive Resource Allocation[1] (MADARA) and Group Autonomy for Mobile Systems[2] (GAMS) open source middleware projects that is intended to be deployed in a multi-agent, multi-functional robotic system called the Keck Institute for Space Studies Multi-Planetary Smart Tile. We discuss our solution approaches to addressing scalability and quality-of-service in deployments of multi-agent systems, codifying group intelligence in hostile space environments, portability for future missions and systems, and assurance and verification of software controllers and algorithms.

2. Introduction

Autonomy is difficult within a single, multi-functional robotic system. Creating a distributed autonomous system of many multi-functional robotic systems is even more difficult, especially if the system is meant to survive and perform missions in space. One core reason operating in space is more difficult is because major failures and issues experienced while in space cannot be immediately repaired or addressed by a human technician due to the potential remoteness of the robotics system. Additionally, messaging can be more difficult, transmission times longer, and loss of packets can be a real problem in distributed systems, especially if the sent information is important. Distributed autonomous systems add layers of complications involving communication between robotic systems and the management and control of computation and messaging overhead between robotic agents.

In this paper, we discuss the planning and prototype software artifacts for the distributed control node concept deployed in the Keck Institute's Multi-planetary Smart Tile project, which aims to perform large-scale power generation, power beaming, locomotion, and coordinated activities throughout the solar

[1] http://madara.sourceforge.net
[2] http://jredmondson.github.io/gams/

system. Because the ultimate goal of the Multi-planetary Smart Tile project is to deploy thousands and even millions of distributed, autonomous power generators with various sensor and actuator payloads, the software challenges presented for such a system in space are daunting.

The first challenge (C1) is with scale and quality-of-service requirements with a distributed autonomous network of at least thousands. The Smart Tile operates as a decentralized swarm, and especially chatty participants can cause bad emergent behaviors such as thrashing and starvation. Thrashing takes place when everyone tries to talk at once and consequently, all messages are garbled. Starvation takes place when one or more participants talk so much that more polite network participants never get a chance to communicate, instead deferring to the high traffic participants. Both of these situations would be unacceptable in a distributed control node (the concept of autonomous agents collaborating together and controlling themselves and each other for some mission-focused objective), such as we propose in the Smart Tile project.

The second challenge (C2), after communication is possible at this type of scale, is with expressing, pursuing and monitoring group intelligence. Group intelligence amongst robotic or software agents comes down to the pursuit of two high-level goals: control and timing. Control is a pervasive need that must be enforced at not only the robotic agent or node level but also at the group level. If a robot performs micro-tasks perfectly but it crashes into other robots and damages itself and others, the control of the system is still flawed. Consequently, control is important.

Timing is also important in group intelligence but can also be more relaxed to give better control (these are often related). In group intelligence, there is a spectrum of group-wise timing that flows along a gradient from synchronous models of computation (SMoC) to asynchronous models of computation (AMoC). SMoC can be best thought of as barriers amongst a group that prevent individual agents from proceeding until the group is at a synchronization point. Asynchronous models of computation occur when robotic agents are free to pursue their own individual objectives or missions at any time. The Smart Tile project has needs that fall along the gradient between these two extremes, and in this paper we'll describe how trending toward SMoC is generally better for our implementation of system assurance in space.

The third challenge is with portability (C3). The Smart Tile project, if successful, will lay groundwork for a long term mission in space. Over time, the Smart Tile will increase capabilities, have different vendors, and need to interoperate with other space systems. Without a portable infrastructure, and especially an open infrastructure, NASA or other space agencies would be tied to a single, possibly proprietary system that is not as extensible, more expensive, and detrimental toward community involvement.

The fourth challenge (C4) is with assurance, namely in the formal verification, if possible, or validation of the system. Before investing billions of dollars in a space project, the project really needs to be verified as operationally feasible, safe, effective and secure. Security is probably the hardest thing to ever truly verify and not really the topic of this paper. However, the verification and validation of operational properties such as feasibility, safety and effectiveness in the target space environment is something we have made progress on via methods like Software Model Checking and other fields in formal methods like Statistical Model Checking.

In this paper, we will present a software infrastructure roadmap and technologies for addressing these four core challenges in the large scale Multi-planetary Smart Tile project. In the context of the first challenge, we will discuss contributions of the Multi-Agent Distributed Adaptive Resource Allocation (MADARA) project for providing quality-of-service for agent communication in the context of a FDMA/CDMA protocol for multiplexing high numbers of chatty agents. For the second challenge, we will discuss the contributions of the authors to distributed symbiotic relationships and multi-agent management and

prosecution with the Group Autonomy for Mobile Systems (GAMS) project. For the third challenge, we will talk about the extensibility options for algorithms, platforms and quality-of-service features of the two middlewares and how they fit into the community outreach aspect of the Smart Tile. For the fourth challenge, we will outline research and results into the verification of the Smart Tile group intelligence using Software Model Checking via the Distributed Adaptive Real-Time (DART) project and via Statistical Model Checking techniques from the Statistical Model Checking (SMC) for Swarms project at Carnegie Mellon University.

3. Related Work

Software architectures tend to refer to high level structures of software systems and also to the process of creating and documenting such structures. The concept of software architectures has been around for a long time and can generally be divided into two main categories in modern computing: service-oriented architectures and middleware.

Service-oriented architectures are often RESTful, a concept popularized by services and websites on the internet. RESTful architectures are generally scalable, given appropriate cluster hardware, have well supported standards, and integrate well with the TCP protocols used on the internet because they are stateless and expect reliable communication. RESTful service-oriented architectures have been used in satellite-based systems before [2]. Unfortunately, RESTful services can be difficult to use in space because of the blocking semantics of the underlying TCP protocols that are frequently used, as these tend to break once communication is lost to earth, and also because satellites/space systems tend to have low powered CPUs instead of the high powered CPUs in clusters that can launch and run many threads—an important part of RESTful architectures that is essential to scalability. We expect disconnections and communication failures to happen frequently with the later phases of the Multi-planetary Smart Tile, so RESTful services do not appear to be an appropriate software architecture for our multi-agent system that may operate in very remote parts of the solar system.

Middleware is a programming technique that provides software layers between an operating system and a developer application that facilitates rapid development of systems. Middleware is very common in networked and distributed application development paradigms and tools, including CORBA, DDS [12], ROS [13], etc. Some of these middlewares have even been used in space systems [11]. However, middleware usage in satellites is limited, and most space system software is monolithic, specialized to the hardware, and brittle. Our approach includes a portable middleware that should be appropriate for most space systems, including the Multi-planetary Smart Tile.

4. Solution Approach

4.1 The Smart Tile's Phase 1 Hardware

The Smart Tile is being developed in multiple mission phases, with the first phase being targeted at simple low-earth orbit (LEO) deployments of basic functionalities. Later phases are intended to be more robust and expensive and include energy beaming technologies, localization equipment, and advanced electro-magnetic antennas for missions to the Moon and Mars. The first phase, which is expected to be deployed in LEO in 2017, needs only communication between six tiles over short-range Wifi radios in space and a long range radio that supports communication with ground crew. The main hardware components of interest to the software architecture in the Phase 1 mission are listed in Table 1.

Table 1. Smart Tile Phase 1 Hardware Components

ID	Part	Function
1	Iridium 9630N SBD Modem	Long-Range Communication
2	Raspberry Pi 3	Computation and Short-Range Communication
3	Sandisk 16GB microSD Industrial	Long-Term Storage
4	5V 160mA Polycrystalline Solar Cell PV	Power Generation
5	3.7v 500mAh Lithium Ion Polymer Battery	Battery
6	TI 16bit Ultra-low-power Microcontroller	Sensor Controller

Unlike later mission phases, the Phase 1 Smart Tile has no locomotion mechanism after being deployed. The Phase 1 Smart Tile instead communicates between other Smart Tiles in LEO orbit and also updates a ground station through the Iridium satellite network.

Each of the parts in the

Table 1 listing creates issues with scale, timing, and assurance (*e.g.*, the Sensor Controller is not really appropriate for managing network connections from many agents due to memory and processing power constraints, which is why we have included a dedicated higher-powered computer, the Raspberry Pi 3, that will be hardened for space). To complicate the situation further, these hardware components are intermediaries, and none of these are likely to be present in the later mission phases of the Smart Tile concept. Consequently, software portability is important in order to support arbitrary future hardware configurations.

4.2 Software Architecture Overview

We have developed two core middleware projects to support distributed systems at scale. The first middleware is called the Multi-Agent Distributed Adaptive Resource Allocation (MADARA) project [7] and provides autonomous systems with a distributed knowledge base, network transports for updating knowledge, portable application threads, and scalable reasoning services. The second middleware is called the Group Autonomy for Mobile Systems (GAMS) project [1, 5] and provides autonomous systems with interfaces for single agent and multi-agent algorithms, hardware actuators and sensors, and full integration with the MADARA project for knowledge, threading, and reasoning. The integration of these two middlewares is shown in Figure 1.

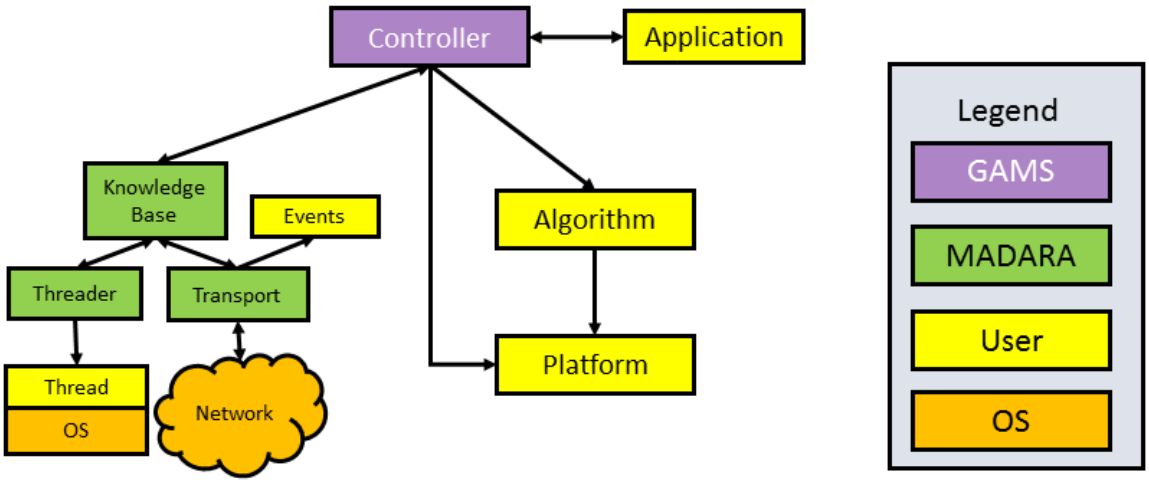

Figure 1. GAMS and MADARA Middleware Integration

The GAMS project is built directly atop MADARA, so it inherits or uses all of its features. For the purposes of programming the Smart Tile, developers create an application that manages a Monitor, Analyze, Plan, and Execute with Knowledge (MAPE-K) controller. The controller is instantiated with a Smart Tile platform appropriate for the hardware installed on the Smart Tile. For example, on the Phase 1 Smart Tile, a platform is provided to the developer that can query the battery level available in the Lithion Ion Polymer Battery or can disable or enable the Iridium 9630N Modem.

The developer also creates or uses a single agent or multi-agent algorithm. Algorithms contain analyze, plan, and execute methods (components of the MAPE-K loop) that are evaluated by the MAPE-K controller at appropriate times. The controller is extensible, but the default behavior provides consistency and quality-of-service mechanisms that are important for assurance techniques that will be discussed later in this paper. Developers can start concurrent threads within inherent consistency and quality-of-service properties that are bound to either the algorithm or the hardware platform's instantiation.

In addition to the MAPE-K control loop, which essentially executes a predictable finite state machine within a distributed system, developers can also program in a reactive model using MADARA events. MADARA events are user code callbacks that occur during I/O operations on the agent (in this case, the application running on the Smart Tile). These user code callbacks events can be on-receive, on-send, and on-rebroadcast operations.

Because we are using the Raspberry Pi 3 as our core computer, we have many options for operating systems (OSes), including real-time kernels such as rtlinux with CONFIG_PREEMPT_RT options compiled in. As of the writing of this paper, the final OS for the Phase 1 Smart Tile has not been decided, though rtlinux is our recommended candidate and is supported by MADARA and GAMS.

4.3 Mapping Software Architecture Features to Challenges

MADARA and GAMS were built to provide scalability and quality-of-service (QoS) to group intelligence (Challenge C1). MADARA provides scalable, feature-rich knowledge, threading and networking services to autonomy developers. GAMS provides interfaces and models for fine-grained control over single agent and multi-agent algorithms that interact with a hardware platform, in this case the hardware of the Multi-Planetary Smart Tile.

In terms of scalability, MADARA and GAMS have programming primitives and interfaces that allow for gigahertz processing rates on the Raspberry Pi 3 being used in the Phase 1 Smart Tile. MADARA uses UDP protocols to provide the networking infrastructure necessary to execute and monitor group intelligence. Not only does the usage of UDP increase scalability of group interactions, but unlike TCP and reliable, ordered history networking protocols required by other middlewares like the Robot Operating System (ROS) 1.0 and 2.0 [13], these UDP-based protocols are also designed for lossy networks and frequently dropped packets between agents within the group intelligence. This is a core feature that meets the need for control and consistent timing of the networked Smart Tile system in a hostile space environment (Challenge C2).

MADARA is built on ACE, which is a well-supported middleware for networking that has been ported to most operating systems (OS) and processor architectures since the 1990s, including modern OSes like Android and MacOS. Consequently, the software architecture deployed on the Phase 1 Smart Tile is expected to be portable to any future platform that has a C++ compiler, effectively addressing Challenge C3.

The MADARA and GAMS middlewares also provide consistency and QoS features that make assurance and formal verification possible (Challenge C4), even in distributed systems in difficult networking environments like the Smart Tile in space. The key features that enable these assurance qualities are in the networking and threading layers of MADARA.

Networking in MADARA is kept consistent and predictable by a system of Lamport clocks enforced on each knowledge record and in the aggregation of knowledge record updates. For verification techniques, these clocks provide predictable guarantees about when and what updates will be applied to the knowledge base, even in the presence of dropped packets. Threads created in MADARA have timing epochs enforced for not only the finite-state-machine-like execution in the GAMS MAPE-K controller, but also in network read threads and other threads started by algorithms and platforms. Additionally, QoS features like bandwidth filtering provide assurance against overusing network bandwidth in transports like the one over the Iridium 9630N SBD Modem in the Phase 1 Smart Tile.

4.4 Building Applications for the Smart Tile

The primary method of programming the Smart Tile is to create a GAMS algorithm which interacts with a Smart Tile platform and a MADARA knowledge base. GAMS algorithms extend the *gams::algorithms::BaseAlgorithm* class and implement three methods: *analyze, plan, and execute*. Algorithms are compiled into native binary formats on the Smart Tile before it is deployed and a custom factory method is implemented and registered with the MAPE-K controller to allow for remotely starting the algorithm or for including the algorithm execution as part of a preprogrammed algorithm sequence.

Smart Tile applications communicate with each other by modifying records in the provided MADARA knowledge base. These modifications are aggregated together into update messages that are sent to other Smart Tiles in the network.

The underlying network protocol is UDP and consequently unreliable in delivery. If synchronization of knowledge is important, the algorithm updates or remodifies important knowledge so that it is resent to the other Smart Tiles in the network. For true reliability, acknowledgements should be sent by receivers and checked on the sender side of client applications.

Each Smart Tile populates its knowledge base with information about status (e.g., algorithm status, the orientation of the tile, etc.), and algorithms react to the statuses of other Smart Tiles in the network by looking for information in predefined variables in the knowledge base. The Smart Tiles each have a unique

identifier for its status variables. By default, this unique identifier is a prefix in the knowledge base that starts with *agent.id*, where *id* is a number or unique string that identifies the agent. The Smart Tile identified as *agent.11* would store its location in the *agent.11.location* variable and can store its battery level in the *agent.11.battery* variable.

Location can be a Cartesian grid, GPS position, etc., and GAMS features a pose system that allows for translations between arbitrary reference frames (*e.g.*, a virtual Cartesian overlay from a GPS spherical grid). Multi-functional device information would also be populated into the agent prefix in the knowledge base, so the information on actuators and sensors can be shared as needed or relevant to other Smart Tiles participating in a group mission.

Algorithms look at such variables in the knowledge base and can respond according to the objectives of the algorithm. For multi-functional systems, such as later phase Smart Tiles, status information for functional actuators and sensors can be populated within the agent prefix in the knowledge base to inform internal and external processes on the state of the individual robotics system or even the collaboration of synergetic Smart Tiles that are using each other's actuators and sensors for common tasks and algorithms.

4.5 Verifying Applications for the Smart Tile

Verifying correct behavior in systems of distributed autonomous agents is extremely challenging. Due to these challenges, Statistical Model Checking (SMC) [10, 14] has emerged as a key technique for quantitative analysis of stochastic systems. Given a stochastic system $\mathcal{M}$ (such as a collection of Smart Tiles performing some task) depending on random input x, and a predicate Φ (e.g., that the system behaves "properly"), the primary goal of SMC is to estimate the probability $P[\mathcal{M} \vDash \Phi]$ that Φ is satisfied in $\mathcal{M}$ within some specified level of confidence (*e.g.*, relative error). SMC, which is based on Monte-Carlo methods, has some major advantages over methods such as probabilistic model checking. It can be applied to larger and more complex systems, and to the actual system software rather than an abstract model of that software. Moreover, it can analyze a system as a "black box" observing only its inputs and outputs.

While estimating the probability that a predicate holds is important, it is also important to understand the factors that contribute to that estimate. In our prior work [8] we refer to this as *input attribution*. An input attribution is a human-understandable quantitative model explaining the relationship between the random inputs and the specified predicate Φ (e.g., a mathematical expression of the input variables that predicts whether Φ will be satisfied). Input attribution is of invaluable use in helping the designer to better understand the sources of risk in a mission. Input attribution can essentially map environmental issues or algorithmic issues to mission failure, assuming the Smart Tile is modeled properly in Monte-Carlo simulations.

We have also investigated a more exhaustive, state-space-exploration process called Software Model Checking to verify distributed systems built with the MADARA and GAMS middlewares [3, 4]. Software Model Checking is the algorithmic analysis of programs to prove properties of their execution and is based on decades of work in logic and theorem proving. Our version of the Software Model Checking approach is called Distributed Adaptive Real Time[3] (DART) [9].

DART provides a language for distributed systems called DMPL that creates abstract descriptions of individual finite state machines and their interactions in a distributed system. A user encodes the distributed system using the DMPL language and this system model is generated into C++ code that links to the MADARA and GAMS middlewares.

[3] http://cps-sei.github.io/dart/

This generated code is edited by the application developer with functionality and compiled using a formal verifying compiler. If successfully compiled, the result is executable on real systems or in simulations and is guaranteed to satisfy the properties verified by the formal verifying compiler. The executed code enforces a synchronous model of computation between the participating nodes or agents to ensure correct behavior. Because it is a synchronous computation, each node must maintain its mission state until each agent is ready to move to the next state in its finite state machine.

Though using such a model of computation is powerful for assurance of safety and functionality, it can also be problematic in space systems where communication is not guaranteed. To get around these limitations, care must be taken to create a DMPL model that is not based on full collaboration between all participants but instead only synchronizes collaboration amongst local cliques that are in communication with each other. However, the holy grail of the DART idea is to be able to apply DART formal verifying compilation techniques to asynchronous models of computation, which is not yet a solved problem.

5. Performance and Scalability

Before deployment in LEO, the software architecture is being tested on similar hardware in laboratories on earth for scalability and throughput. In addition to real-world deployments tests with Platypus LLC Lutra boats in formations between five and twenty boats, we have been performing scalability tests in clusters of ARM processors that we expect to use on the Multi-planetary Smart Tile. Unfortunately, we were unable to acquire a cluster of Rapberry Pi 3s before this paper deadline. Instead, we report scalability tests of a similar candidate computer unit, the ODROID XU4. The ODROID XU4 features a Samsung Exynos5422 Cortex-A15 2Ghz processor and 2 GB of LPDDR3 933MHz 32 bit RAM. We used a cluster of 20 ODROID XU4s connected via 100 MB cat5.

Table 2. Per Agent Send and Receive Rates for 1KB packets of Knowledge in 20 ODROID XU4s

Per Agent Operation	Min Hz	Max Hz	Avg Hz
Publish rate	500.00	517.30	507.46
Receive rate	432.00	490.00	468.12
Total Receive Rate	8,820.00	9,082.00	8,921.38

Table 2 shows the observed publication rate for large aggregated knowledge packets (1 KB of knowledge in each packet) within each of the 20 ODROID XU4s. The GAMS controller was single threaded and the MADARA UDP multicast transport was configured to only use one receive thread. The software agents on each ODROID were set to bursting publication rates (essentially trying to send packets as fast as possible) over the Ethernet backbone. This can cause extreme thrashing within an operating system and Ethernet driver, but the MADARA transports handled the traffic quite well.

The Phase 1 Smart Tile deployment is expected to only be six tiles, roughly one third this deployment, and the publish rates are expected to be much lower—between one and five hertz. So, we are handling orders of magnitude more hertz than Phase 1 requires of the Raspberry Pi 3. The average hertz processing power of each agent with only 1 read thread allocated appears to be 1486x what is needed for 1hz send rates in a six tile deployment and 297x what is needed for a 5hz send rate per agent. 1 KB packets are also way larger than those we have planned for any phase of the Smart Tile project. Our testing has shown that packets are between 150 and 250 bytes, depending on the implemented GAMS algorithm and what it needs to update.

However, 140 bytes of this is the standard MADARA header, which has a lot of QoS-related settings. The MADARA standard reduced header removes much of the QoS-related information and results in a 30 byte header. If necessary to reduce network and operational overhead, we can remove all of this with a custom header and transport to reduce this to a maximum of 150 bytes with no real change to the knowledge marshelling and demarshelling process.

These results show the raw throughput power available in the MADARA and GAMS middlewares for processing knowledge updates when communication is available in space. Because the MADARA middleware is asynchronous and focuses on robustness in the presence of dropped packets, algorithms can be designed to scale to even thousands of communicating tiles, assuming communication bandwidth is available and tiles are within communication range of each other.

6. Conclusion

In this paper, we have discussed the software architecture being developed for the Keck Institute Multi-Planetary Smart Tile project. We presented four challenges to creating software for a multi-agent system in space, and we outlined our solution approach for scalability, quality-of-service, portability and verification. We are in the process of building prototypes of the Phase 1 Smart Tile, which is expected to deploy into low earth orbit in 2017, and the software architecture described in this paper should be part of that deliverable. Future work for the software architecture includes human interaction interfaces between ground and the multi-agent system in space as well as collaboration enhancements, such as consensus (*e.g.*, voting and auctions) and machine learning primitives, for the autonomous Smart Tiles expected in Phase 2 and Phase 3 deployments to places farther from Earth with more autonomy and locomotion in the Smart Tiles.

7. Acknowledgment

Copyright 2016 Carnegie Mellon University. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute. NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT. [Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. DM-0003932

8. References

1. Dukeman, Anton, Julie A. Adams, and James Edmondson. "Extensible collaborative autonomy using GAMS." Proceedings of the 31st Annual ACM Symposium on Applied Computing. ACM, 2016.
2. Cappelaere, Pat G., Stuart W. Frye, and Daniel Mandl. "Flow-enablement of the NASA SensorWeb using RESTful (and secure) workflows." 2009 IEEE Aerospace conference. IEEE, 2009.
3. Chaki, Sagar, and James Edmondson. "Toward parameterized verification of synchronous distributed applications." Proceedings of the 2014 International SPIN Symposium on Model Checking of Software. ACM, 2014.
4. Chaki, Sagar, and James Edmondson. "Model-Driven Verifying Compilation of Synchronous Distributed Applications." International Conference on Model Driven Engineering Languages and Systems. Springer International Publishing, 2014.

5. Clarke, Edmund M., and Paolo Zuliani. "Statistical model checking for cyber-physical systems." International Symposium on Automated Technology for Verification and Analysis. Springer Berlin Heidelberg, 2011.
6. Edmondson, James, Gene Cahill, and Anthony Rowe. (2014). On Developing User Interfaces for Piloting Unmanned Systems. Proceedings of the International Workshop on Robotic Sensor Networks 2014. Berlin, Germany. 2014.
7. Edmondson, James, and Aniruddha Gokhale. "Design of a scalable reasoning engine for distributed, real-time and embedded systems." International Conference on Knowledge Science, Engineering and Management. Springer Berlin Heidelberg, 2011.
8. Hansen, Jeff, Sagar Chaki, Scott Hissam, James Edmondson, Gabe Moreno, and David Kyle. “Input Attribution for Statistical Model Checking using Logistic Regression.” Runtime Verification. Springer International Publishing, 2016.
9. Hissam, Scott A., Sagar Chaki, and Gabriel A. Moreno. "High Assurance for Distributed Cyber Physical Systems." Proceedings of the 2015 European Conference on Software Architecture Workshops. ACM, 2015.
10. Kyle, David, Jeffery Hansen, and Sagar Chaki. "Statistical Model Checking of Distributed Adaptive Real-Time Software." Runtime Verification. Springer International Publishing, 2015.
11. Pan, Jiantao, et al. "Robustness testing and hardening of CORBA ORB implementations." Dependable Systems and Networks, 2001. DSN 2001. International Conference on. IEEE, 2001.
12. Pardo-Castellote, Gerardo. "OMG data-distribution service: Architectural overview." Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on. IEEE, 2003.
13. Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.
14. Younes, Hakan L. Verification and planning for stochastic processes with asynchronous events. No. CMU-CS-05-105. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2005.
15. Yang, Chao, Nengcheng Chen, and Liping Di. "RESTFul based heterogeneous Geoprocessing workflow interoperation for Sensor Web Service." Computers & Geosciences 47 (2012): 102-110.