The DARPA CODE White Force Network

David Scheidt
Weather Gage Technology, LLC
Edgewater, MD
dscheidt@weathergagetech.com

Robert Lutz
JHU/APL
Laurel, MD
robert.lutz@jhuapl.edu

James Sturim
CENTRA Technology, Inc
Arlington, VA
james.sturim.ctr@darpa.mil

ABSTRACT

The DARPA Collaborative Operations in Denied Environments (CODE) program is developing highly advanced collaborative autonomy capabilities that will allow unmanned aircraft systems (UAS) to successfully engage mobile targets in denied or contested electromagnetic environments. The core goal is for CODE enabled Air Vehicles (AVs) to autonomously sense and evaluate the state of its operational environment, form teams, and carry out defined mission objectives with limited human supervision.

The testing of CODE capabilities on live ranges requires an interacting set of live, virtual, and constructive (LVC) assets to provide the necessary stimulus to the system under test (SUT). The CODE White Force Network (WFN) is designed to dynamically interject operationally-relevant effects, such as denial of communications or GPS, into the CODE software during flight as a means of stimulating and then verifying the performance of CODE autonomy algorithms. The WFN also allows large numbers of high fidelity virtual assets running the actual CODE software to be part of the test scenarios. In addition, the WFN ground station provides synthetic forces generation services and various control, visualization, and logging functions that interact in real-time with the on-board WFN flight software to create the desired effects.

This paper provides an overview of the WFN design and describes how the WFN was integrated into recent CODE test campaigns at NAWC-WD in China Lake, CA. The paper also discusses the increase in complexity planned for the next CODE phase test environment and how the WFN will address the associated technical challenges.

ABOUT THE AUTHORS

David Scheidt is the founder and CEO of Weather Gage Technology, LLC where he conducts research on autonomous test systems and intelligent fault management. Mr. Scheidt has 30 years of experience in the research and development of distributed information management systems, robotics, artificial intelligence and process control systems. Throughout his career Mr. Scheidt has conducted research in concert with his development efforts, publishing over 50 peer reviewed publications for research funded by the National Computer Security Center (NCSC), DARPA, ONR, NASA, DISA, OSD NII and the US Army. Mr. Scheidt currently conducts autonomous systems and intelligent controls research and is the principal investigator on research initiatives that focus on the intelligent diagnosis, reconfiguration and planning of ship auxiliary systems, spacecraft and unmanned vehicles.

Robert Lutz is a principal staff scientist at The Johns Hopkins University Applied Physics Laboratory in Laurel, MD. His background includes 37 years of practical experience in the development, use, and management of models and simulations across all phases of the DoD systems acquisition process. He currently serves as the Navy's MQ-4C (Triton) Program M&S lead in the Airspace Integration (AI) area. He also supports LVC testing for several autonomy science and technology (S&T) programs, such as the Safe Testing of Autonomy in Complex Interactive Environments (TACE) Project and the Collaborative Operation in Denied Environments (CODE) Program. In addition, Mr. Lutz serves as the Chair of the Simulation Interoperability Standards Organization (SISO) Board of Directors and Vice Chair of the SISO Executive Committee; serves on the Tutorial Board and Fellows Committee at the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC); and he is a guest lecturer on various M&S-related topics in The Johns Hopkins University Whiting School of Engineering.

James Sturim is a Senior Flight Test Engineer at CENTRA Technologies, Inc. in Arlington, VA. He is a retired Air Force Colonel with 25 years of Active Duty. He has been involved in flight test and acquisition for numerous programs including fighters, ISR, and unmanned platforms and their associated subsystems. He is the Flight Test Lead for several DARPA programs including the Collaborative Operations in Denied Environment (CODE).

The DARPA CODE White Force Network

David Scheidt
Weather Gage Technologies, LLC
Edgewater, MD
dscheidt@weathergagetech.com

Robert Lutz
JHU/APL
Laurel, MD
robert.lutz@jhuapl.edu

James Sturim
CENTRA Technology, Inc
Arlington, VA
james.sturim.ctr@darpa.mil

MOTIVATION

Within the scope of this paper, an autonomous system is defined as a physical system that can devise an appropriate course of action in response to unanticipated circumstances encountered in an intractably complex world. In contrast, systems that can operate without human intervention when responding to simple problems that can be solved in the design phase and systems that require human operators to be effective are not considered to be autonomous. This is not to say that autonomous systems do not interact with humans; they do. At all times, it is expected that autonomous systems are subordinate to a human commander that is responsible for system deployment, establishes the goals and objectives that motivate the autonomous system and when feasible, monitors autonomous system performance. None-the-less, for a system to be considered autonomous, at some time during deployment the system must be required to independently devise and act on a course of action that satisfies commander's intent without support from a human.

A system's ability to autonomously devise a course of action under complex, uncertain engagements can provide a decisive military advantage under several widespread use cases. The most widely discussed autonomy use case is the potential for cost savings associated with removing decision-making tasking from humans. The second use case arises from the potential for artificial intelligence (AI) to produce higher quality decisions than humans. The third use case arises when the time required for a human to make the necessary decisions, when compared to an autonomous system response time, damages mission-effectiveness by prohibiting timely decision-making. A common example of the third use case occurs when communications between unmanned vehicles and unmanned vehicle operator(s) are denied or delayed. The DARPA Collaborative Operations in Denied Environments (CODE) program is developing the software for autonomous unmanned air vehicles to specifically address this third use case both because of the likely loss of communication during mission execution, and because of the complexity of the mission would exceed a human's ability to respond fast enough. For a discussion and mathematical basis that defines the importance of autonomy in denied environments see (Scheidt, 2014a). Regardless of use-case, all autonomous systems include a reasoning system that is responsible for assessing the complex world and devising a course of action.

	External World	Self-External	Self-Internal
Perception	Target Detection Target Classification Target Tracking	Localization Pose/Orientation Payload Status	Health Status
Action	Path Planning Manipulation Task Assessment	Payload Deployment	Systems Reconfiguration

Figure 1 - The reasoning engine within an autonomous system is composed of multiple reasoning components that work together to devise and execute autonomous responses.

Just like human cognition, an autonomous system's reasoning system is composed of a diverse collection of cognitive subsystems. The National Institute of Standard's Alfus framework (Huang, 2004) recognized that autonomous systems are decomposable along multiple dimensions. Figure 1 identifies six major cognitive subcomponents common to autonomous systems divided by two axes: cognitive domain and perception-action. Cognitive domain defines the scope of the reasoning, which includes: (a) exteroceptive reasoning, which is reasoning about the world in which the system operates; (b) proprioceptive reasoning, which is reasoning about the systems place within the external world and (c) interoceptive reasoning, which involves reasoning about the system's internal components. The horizontal axis of Figure 1 denotes the two primary reasoning tasks within the domains: perception and action. Autonomous systems require that reasoning components that address all six modules defines in Figure 1 work together to satisfy mission objectives. While the aggregate reasoning system addresses complex, unsolvable problems, some reasoning components may be tasked to address simple sub-problems that can be solved during design. For example, a rotorcraft

designed to fly through unknown canyons may produce autonomy by combining an artificial intelligence (AI) path planning algorithm with LIDAR-based map-building algorithms that are not, by themselves, autonomous.

Some autonomous systems are designed to operate in worlds that are composed of complex, unpredictable locations/movements of artifacts with known characteristics. For example, a chess playing robot must be capable of devising a move for any one of 2^{155} possible scenarios (Allis, 1994), yet the chess playing robot is allowed to know, a priori, how each chess piece can move. In contrast, military systems developed by a clever adversary will include systems, tactics and behaviors that are unknown to the autonomous systems at the start of an engagement. Autonomous systems designed to operate in worlds containing these "clever adversaries" require reasoning system components that utilize machine learning. While all autonomous systems present test and evaluation challenges, autonomous systems that utilize machine learning present additional challenges as the system response to specific stimuli is subject to change over time.

Testing Autonomous Systems

The complexity of the decision-making process prohibits exhaustive autonomous system testing. Design of experiments (DoE) and statistical analysis, methods normally used to support the testing of complex systems, struggle to produce the evidence required to make a sound assurance argument for the safe, productive, fielding of autonomous systems. These difficulties stem from several sources. First, inter-dependencies between cognitive subsystems with the reasoning engine make it difficult, if not impossible, to isolate independent variables required to manage complexity using DoE methods. Second, system complexity is driven not only by system under test's (SUT) decision process but the complex, iterative, cognitive interactions between the SUT and other decision-makers found in the world. The interplay between SUT and other actors requires the test system to consider not only the SUT response to all possible circumstances but the responses to SUT actions that could potentially be made by other decision-making actors in the world. Third, recognizing that not all external decision-makers will be friendly, SUT testing cannot rely solely on statistical methods as adversaries are likely to observe SUT functionality and design unpredictable response sequences that are cannot to be uncovered using statistical test methods.

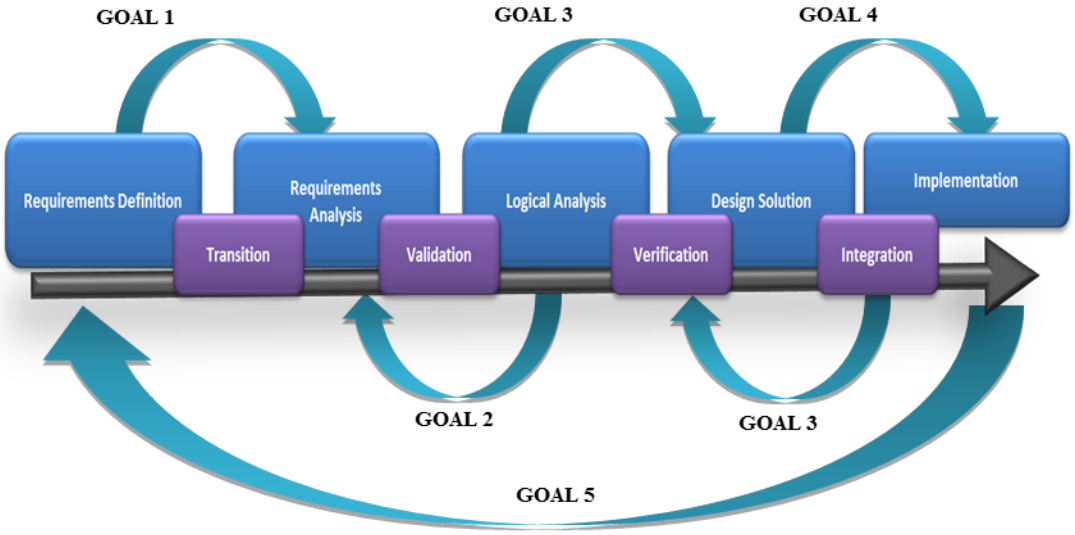

Figure 2 - The OSD Autonomy COI TEVV Process provides a framework for testing of autonomous systems that establishes the need for live-virtual constructive testing.

The DoD Autonomy Community of Interest Test, Evaluation Verification and Validation working group (OSD ACOI TEVV-WG) has identified a process for testing autonomous system that mirrors the systems engineering "V" process (Schaefer, 2015). As shown in Figure 2, this process provides for incremental testing of the SUT throughout the development cycle to support five test, evaluation, verification and validation (TEVV) goals which are:

1. Assure the correctness and completeness of autonomous system requirements through "*precise, structured standards to automate requirements evaluation for testability, traceability and de-confliction"*;

2. Reduce T&E burden by using evidence-based design to assure that the SUT makes appropriate decisions based upon traceable evidence at every level of design;
3. Provide cumulative evidence through research and developmental test and evaluation, developmental test, operational test and post deployment testing (which is required for learning systems) by using progressive sequential modeling, simulation, test and evaluation;
4. Utilize real-time monitoring, just in-time-prediction and mitigation of unsafe decisions to migrate algorithm testing into a "lifetime sport" that is used during DT&E, OT&E and post deployment and finally
5. Produce *"an assurance argument based upon previous element building blocks"* generated by tools used to satisfy goals 1..4.

The OSD TEVV process is mirrored by Torens' 6-step autonomy TEVV process (Torens, 2014) that starts with (a) formal analysis, (b) static testing, (c) unit testing, (d) software in-the-loop testing, (e) hardware in-the-loop testing and finally (f) flight testing.

OSD TEVV WG's goals *3 and 4* and Toren's steps *e and f* involve the test of a functioning autonomous hardware prototype and/or the final autonomous SUT. As previously discussed, autonomous system performance is dependent upon complex, iterative interactions between the autonomous systems and other decision-makers in the environment, which may or may not be friendly and may or may not be human. Keeping in mind that fully testing the autonomy requires understanding the decisions made in the presence of numerous (hundreds or even thousands) independent actors, conducting autonomous system tests entirely with live assets is likely to be cost-prohibitive and unsafe. By example, if you were testing to see whether an autonomous unmanned air vehicle could see and avoid a college student in a hang glider, should you use an actual college student and hang glider?

Because autonomy TEVV is dependent upon examining the interaction between the cognitive processes within the SUT and between the SUT and other decision-makers in the engagement, live-virtual-constructive (LVC) environments leveraged to test autonomous systems must explicitly model the impact of decisions made by virtual and constructive entities and inject synthetic observations of those effects into the decision-making process of nearby entities with near-zero latency.

WHITE FORCE NETWORK

The use of LVC environments is pervasive across DoD test ranges today as a means of verifying the ability of systems that enable warfighting capabilities to meet a given set of requirements. In such environments, a series of test scripts are executed which replicate the conditions in which a live SUT must operate so that accurate system behaviors and performance can be evaluated. Virtual and constructive (synthetic) actors are used to stimulate the SUT in ways in which adherence to system requirements can be effectively measured. However, since the subsystems on-board the SUT cannot actually detect or engage synthetic actors, workarounds are frequently required to create the desired effects. For instance, a surrogate sensor simulation(s) capable of detecting synthetic entities can be used instead of the real sensor(s), but then the detection of live entities becomes problematic. Another approach is to simulate the entire SUT and employ live entities as a potential augmentation to the test environment. This approach works well during early developmental testing, but final verification of system requirements generally requires a live SUT. The inherent limitations of these workarounds suggest the need for a more innovative approach to live-synthetic interactions on DoD test ranges.

Since the main purpose of CODE is to demonstrate the ability of collaborative autonomous unmanned air vehicles (UAVs) to operate effectively in denied environments, the CODE LVC environment provided by the White Force Network (WFN) must be able to deny both communications and Global Positioning System (GPS) information to CODE-enabled virtual or live UAVs dynamically during the execution of a test event. Figure 3 illustrates a typical communications denied use case. The CODE Control Center is the main ground station used to monitor and control all test activities. The SUT can be either a live or virtual UAV with the CODE autonomy software. The Synthetic Ground Actor is a constructive entity (such as a truck or a tank) that is instantiated and controlled via the CODE synthetic forces generator (SFG). Finally, the Synthetic Communications Jammer is also an SFG-generated constructive entity, but with an ability to jam communications to a range determined by transmitter power and the jammer-receiver geometry. During the execution of a test, as blue forces attempt to communicate, the WFN will block the reception of any message by any entity that is too close to the jamming source. The airborne component of the WFN accomplishes this by intercepting the received message before it can be processed by the autonomy software in

the CODE mission computer. As the test script advances in time, blue entities may move into and out of jamming range depending on their movement path, affecting the ability of the CODE autonomy to cooperate with other CODE entities and thus creating the desired test conditions.

Figure 4 illustrates the GPS denied use case. Whether a CODE-enabled virtual or live UAV is in a GPS-denied state currently depends on a simple slant range calculation from the jamming source, although more sophisticated physics-based calculations may be implemented in later developmental phases. When the UAV is GPS-denied, the UAV is forced into an inertial navigation only mode with drift errors calculated by the WFN ground component according to an exponential distribution. The drift delta is then sent up to the WFN air component to apply to the navigation state messages being sent to CODE mission computer. This causes the CODE autonomy to perceive an increasingly false own-ship location, which adversely affects its ability to coordinate operations with other CODE-enabled vehicles. Thus, in both the communications and GPS denied cases, the WFN creates the conditions to study the associated effects on the ability of CODE-enabled vehicles to autonomously conduct defined missions in denied environments.

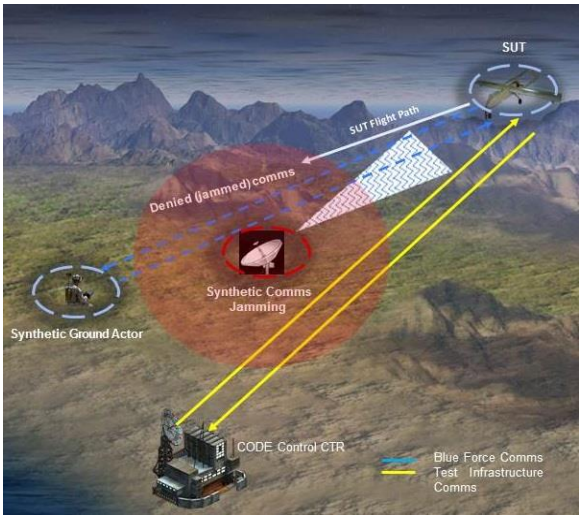

Figure 3 - Communications Denied

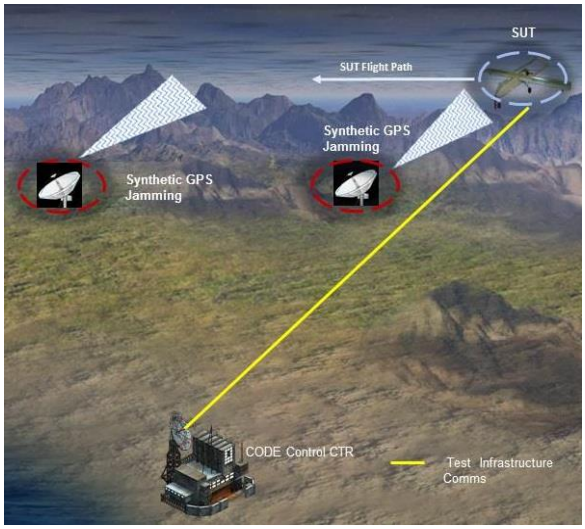

Figure 4 - GPS Denied

Another important innovation provided by the WFN (to be implemented in the next iteration of CODE) addresses the problem stated earlier regarding live interactions with synthetic entities. Specifically, for a live player with an actual sensor, how can synthetic entities be made to appear within the signal processing stream of the sensor? WFN solves this problem by integrating a simulation of the live UAV sensor with the ground-based CODE SFG. The CODE SFG receives periodic UAV state updates via a wireless (Wave Relay Radio) link to the CODE ground station and determines detectability based on sensor orientation, sensor performance characteristics, and position/signatures of synthetic entities in the local operational area. For those synthetic entities that are detectable, the target IDs and locations are sent via the same wireless link to the airborne WFN component. The WFN then inserts the synthetic target information into the same signal processing stream used for the detection of live entities. This insertion is performed after the UAV's automatic target recognition (ATR) system has processed the raw sensor data into individual target tracks to avoid the inherent complexity of real-time scene generation with synthetic actor insertions.

In addition to the insertion of synthetic entities into the UAV's sensor stream, it is equally important to address the need to exclude live targets from potential detection in some circumstances. For instance, if the CODE SFG determines that a live target has been destroyed by a synthetic weapon at some point during test execution, the continued execution of the test script requires that this target cannot be detected by any other entity for the duration of the test. However, since the live target was not actually destroyed, the live UAV's sensor(s) will continue to observe its presence. To avoid this situation, the WFN has the ability to remove targets (post-ATR) as well as add them. In this case, a unique identifier for live targets considered destroyed by the CODE SFG is uplinked to the CODE airborne WFN component, and any subsequent target detection that possesses the same identifier will be deleted before it can be processed by the CODE autonomy. This has the effect of dynamically eliminating live targets from the test execution although they still have a physical presence.

WFN SOFTWARE ARCHITECTURE

The WFN is an enhanced version of the Testing of Autonomy in Complex Environments (TACE) infrastructure (Scheidt, 2014b) that was developed to support the safe testing of autonomous unmanned vehicles in complex, interactive engagements. WFN improvements allow for the system to be used to support teams of autonomous unmanned vehicles operating in contested RF environments. The TACE-WFN software architecture is a distributed architecture that contains ground-based software and software located on-board the SUT as shown in Figure 5. The WFN architecture is designed to leverage high-powered ground-based computers to model and simulate the complex interactions between hundreds of autonomous actors while using lower-powered on-board processors to inject synthetic data modeling RF events directly into the SUT's on-board autonomous control system.

WFN ground software uses a service-oriented architecture that is based on the Test and Training Enabling Architecture (Hudgins, 2005). Developed for the Test Resource Management Center, TENA is an emerging standard for test support systems at DoD Major Range and Test Facility Bases, including NAVAIR's range in China Lake, CA. Key ground software services are: the Test Executive, Synthetic Forces Generator (SFG), communications server and operator graphical user interface. In addition to the four primary modules, the ground system is capable of supporting multiple ground-based virtual and/or constructive actors to include human-piloted synthetic vehicles (flight simulators).

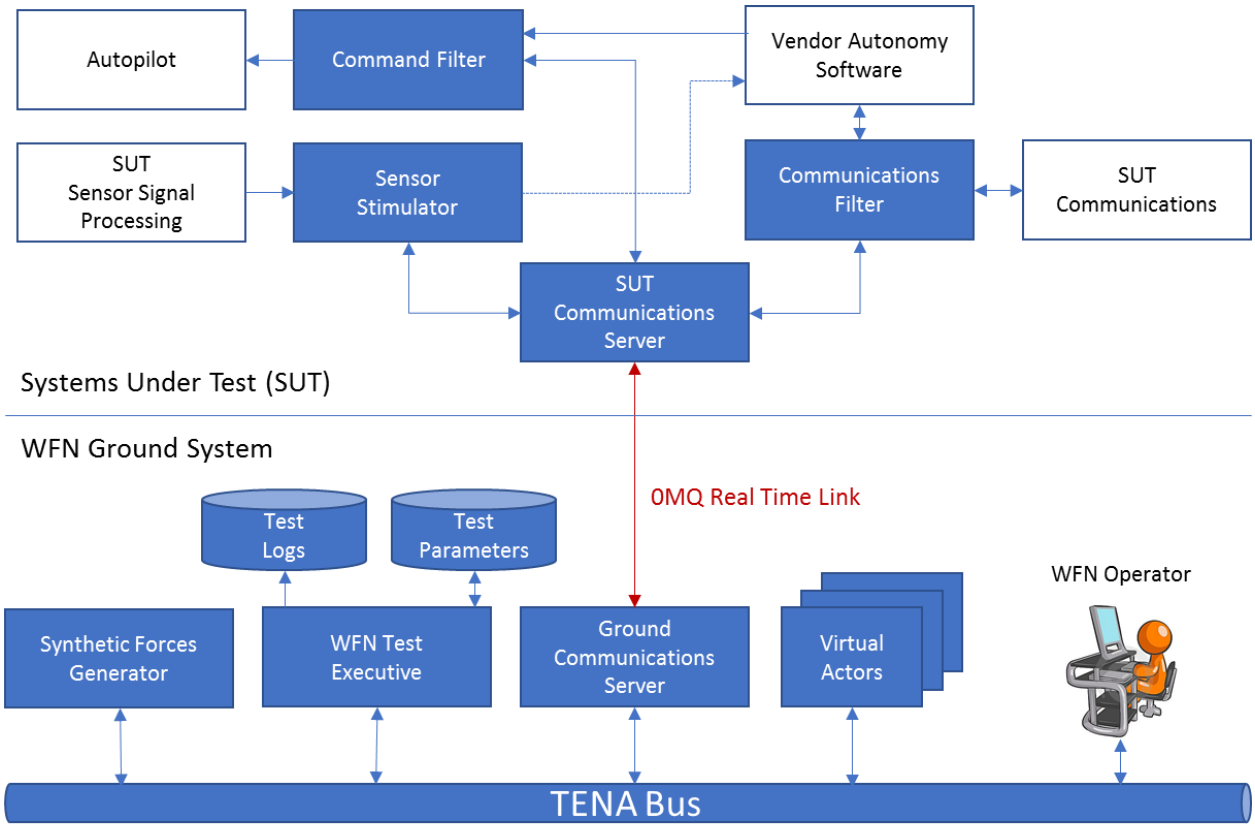

Figure 5 - WFN software is a distributed two-part system that includes ground-based modules communicating over a TENA bus that communicate to software modules embedded in the SUT over a real-time network link.

Automated test management and synchronization is provided by the Test Executive. The Test Executive interprets operator-defined test parameters to establish the test's initial conditions, start, stop and synchronize WFN services and to send messages to other software modules initiating test events. The Test Executive also logs test events and SUT responses, working with the WFN operator graphical user interface to display real-time test status and test playback.

As previously discussed, the SFG is an agent-based simulator that models both live and synthetic actors and the interactions between them. The SFG uses status updates from live and virtual actors to maneuver constructive proxies within the SFG engagement simulation. Simulation-based events are reported by the SFG via interaction messages that report live-virtual, live-constructive, virtual-constructive and virtual-virtual actor relationships. Interaction reports are sent from the ground system to live actors over a real-time, mobile ad hoc network link. Interaction reports are sent to virtual actors over TENA.

The ground communications server works in tandem with its airborne counterpart, the SUT communications server, to pass messages between ground system components and WFN software components embedded within the SUTs. The ground and SUT communications servers use 0MQ, which provides a real-time link. Real-time network protocols, unlike TCP/IP, are designed to guarantee timely delivery of all delivered packets, but do not guarantee that all packets will be delivered. By using a real-time network protocol the WFN can prevent delays, or "lagginess" in injected synthetic data.

Accurate assessments of an SUT's autonomous response to a synthetic stimulus requires that the WFN inject stimuli into an SUT signal processing chain within one decision cycle. Failure to inject synthetic data at this rate can invalidate the test as old data can cause an SUT to base a decision on engagement data that is inappropriately inconsistent with another actor's world views. When supporting CODE, the WFN is required to inject synthetic EO signatures and synthetic RF events, including communications and communications jamming at a > 50 Hz rate. Injected synthetic data is dependent upon both the overall engagement status, which is modeled in the ground-based SFG, and the current location and orientation of the SUT, which is observed by SUT on-board sensors. Note that SUT sensors may have varying orientations due to the use of a gimbal to position the sensor field of view. The separation of knowledge between the ground and air presents the WFN's core architectural challenge, how to simulate a highly complex, interactive tactical engagement, which, due to the tight timing requirements, cannot be accomplished using distributed SUT-hosted processors communicating over an RF network, while simultaneously providing near instantaneous on-board data injections that are dependent upon the current field of view of the SUT sensors. The WFN software architecture addresses this challenge by using a segregated simulation architecture. The ground-based SFG simulation is capable of modeling large numbers of actors and actor relationships because it uses lower-fidelity, kinematical models of actors. High-speed, high fidelity simulation of proximate actors and events is supported on-board the SUT, which has access to real-time data feeds from on-board guidance and control sensors. The two-part system is coordinated by communicating actor positions over the 0MQ link. This is accomplished with minimal latency and over low-bandwidth by only transmitting those reports that have the potential to interact with a specific SUT. For example, if a SUT's sensor has an effective range of one kilometer, the communications server will filter out all reports on actors outside of one kilometer. Messages sent from the SFG are, depending upon content, used on-board the SUT by the Sensor Stimulator, Communications Filter and Command Filter.

The Sensor Stimulator serves two functions. First, the Sensor Stimulator modifies readings from on-board ELINT, SIGINT and GPS payloads in accordance with jamming events precipitated by virtual and constructive actors. Second, the Sensor Simulator injects artificial readings that represent detections of synthetic and virtual actors by EO/IR sensors.

The Communications Filter also serves two functions. First, the Communications Filter uses virtual and constructive jamming event messages by appropriately removing and/or delaying packets deemed to be "jammed". Second, the Communications Filter injects additional communications packets that are produced by virtual peers of the SUT.

One of the key capabilities being pursued by CODE is an ability for autonomous air vehicles to recognize and manage GPS-denied events. In order to test capability without enduring the flight safety risks associated with jamming the GPS signal, the Sensor Stimulator modifies GPS-packets being provided by the on-board GPS receiver to the SUT's autonomy system, presenting the SUT autonomy package with GPS data that is equivalent to data that would be produced during a jamming event. However, for flight safety preservation the WFN does not modify the GPS data used by the autopilot. Not modifying the autopilot's GPS data produces an inconsistency between the autopilot and GPS data used by the SUT autonomy. Inconsistency between GPS and autonomy GPS data produces the problem that, when the SUT autonomy correctly detects the GPS denial and produces a set of autopilot commands that correctly achieves mission objectives in spite of the GPS denial, the autopilot will not fly to the waypoint intended by the autonomy software because the GPS data used by the autopilot has not been modified. To correct this problem, the

Command Filter modifies commands sent from the SUT autonomy system to the autopilot, subtracting out all biases injected by the Sensor Stimulator and flying the SUT to the waypoint(s) intended by the autonomy software.

CODE Testing

The RQ-23 TigerShark (Figure 6) is a lightweight, long endurance, multi-mission capable, low cost, and recoverable UAS. The TigerShark has a high wing design with a single pusher engine and is flown by a computer assisted autopilot or pilot in a Ground Control Station (GCS). The CODE software was loaded into a specially modified RQ-23 to support the live test. This software was designed to conduct collaborative and autonomous functions and interface with the RQ-23 to command the physical actions to implement these functions. As test safety is always a priority, sufficient safety measures were put into place to allow the SUT to operate without unnecessary limitations, but to minimize the chance of loss of the UAS, injury to people, or damage to property. The CODE team developed an architecture of subsystems that they added to the RQ-23s and on the ground to create a safe environment for the SUT software to operate without putting the RQ-23s at undue risk.

Figure 6 - RQ-23 TigerShark

RQ-23 aircraft are controlled from the ground via the TigerShark GCS which is located in the mission shelter (see Figure 7). The mission shelter has stations for the Air Vehicle Pilot (AVP), Mission Commander (MC), Payload Operator, and two auxiliary stations for additional personnel. The ground station computer communicates with the aircraft using a data link.

The added hardware components were a Wave Relay Radio with antennas, an F-Box Smart Switch, a Watchdog Computer, a video encoder, and a CODE Mission Computer. Custom software was developed for the F-box and Watchdog. The WFN software was hosted on the CODE Mission Computer along with the performer's CODE software. Each one of these components had specific capabilities that were critical for the conduct of the test.

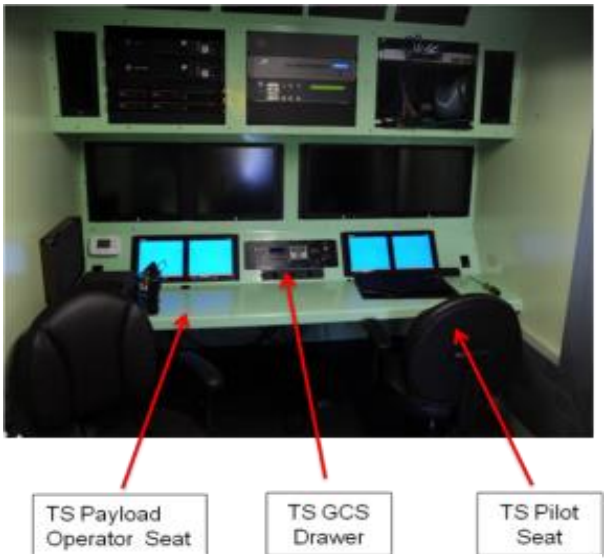

Figure 7 - RQ-23 Tigershark Ground Control Station

The CODE Program is predicated on unmanned systems being able to communicate within their teams. This requires the use of radio frequency communications. In order to not interfere with the normal RQ-23 operations, the decision to add a Wave Relay Radio onto the RQ-23 was made. The Wave Relay Radio was chosen due to its advantageous Size, Weight, and Power (SWAP) and organic Mobile Ad Hoc Networking (MANET) feature that increased the probability of communications between the UASs and the test systems on the ground. The CODE Software communicated with the other UAS, the Mission Commander Station, and the WFN via this radio. The Wave Relay Radios were used in their commercial off-the-shelf configuration.

The F-box Smart Switch was the single point where the CODE Mission Equipment and the organic RQ-23 network interfaced. The F-box had a few safety functions. First, it monitored the network traffic and metered the CODE network traffic so as to not overwhelm the organic network that was needed for safety of flight functions. Second, the F-Box monitored the primary Command and Control link between the RQ-23 and the GCS. If that link was lost for more than a set time, it logically disconnected the CODE Mission Equipment from the RQ-23 Network to allow

the approved/proven procedures to re-establish primary communications. The hardware and software for the F-box were designed and developed specifically for the CODE program.

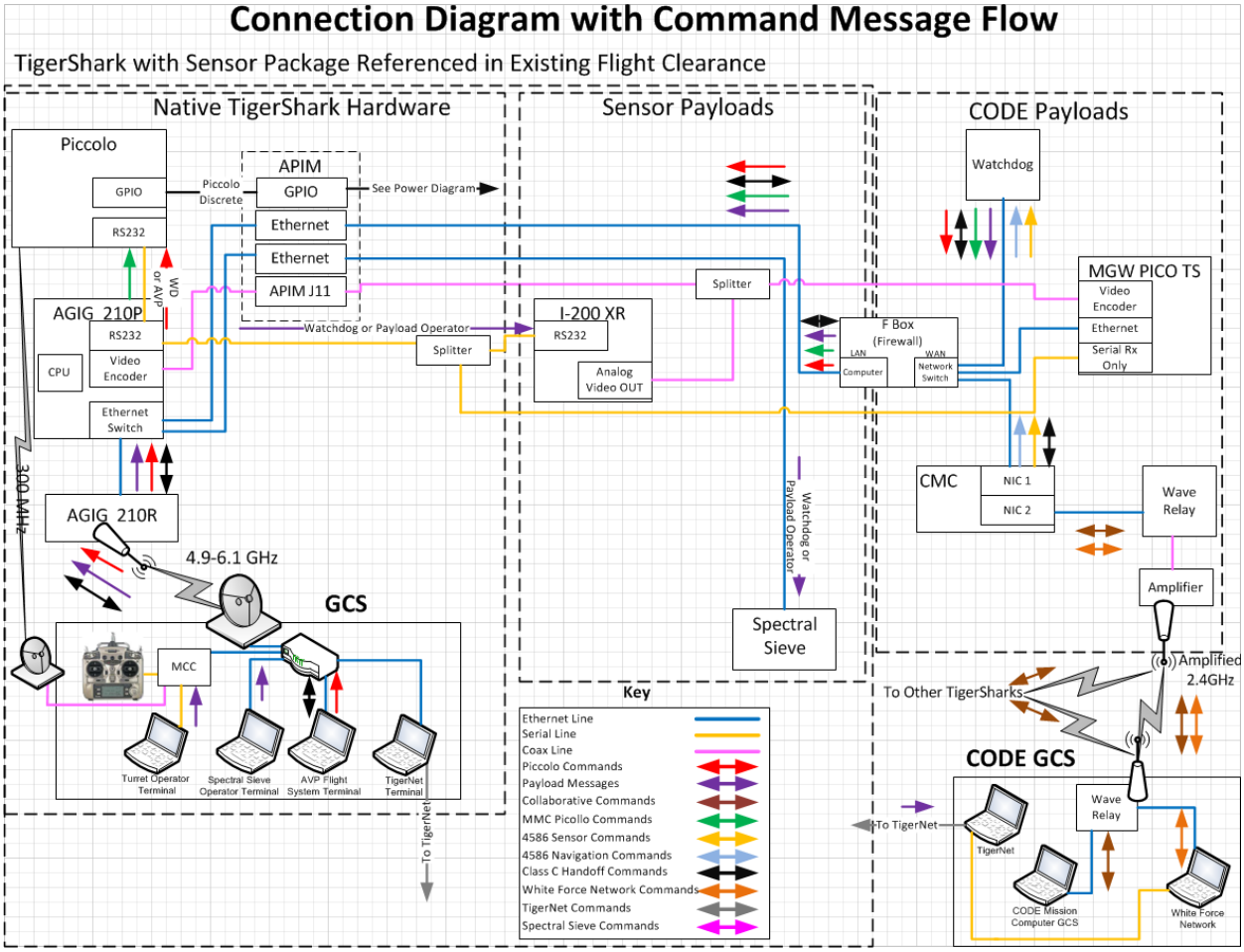

Figure 8 - CODE Mission Equipment Diagram with Command Message Flow

The Watchdog is a small computer on-board the RQ-23 that has both flight safety and test conduct capabilities. Commands to the RQ-23 from the CODE Software are routed to the Watchdog. The Watchdog then filters those commands on a subset of allowable commands. For example, the command to terminate the UAS is strictly prohibited and is caught by the Watchdog software. Additionally, the CODE Software commands are in STANAG 4586 compliant format. The Watchdog converts these messages into a Piccolo[1]native format to be sent to the RQ-23 autopilot. The conversion requirement provides two benefits. First, the conversion makes the CODE software comply with a known portable standard, the STANAG 4586 was chosen. Second, conversion adds one more layer of safety as there is no possible way for a rogue command from the CODE software to get to the RQ-23 autopilot without this translation. The Watchdog also takes systems state inputs from the RQ-23, converts them into STANAG 4586 compliant messages, and passes them to the CODE Mission Computer. The Watchdog software was designed and developed specifically for the CODE program.

A video encoder was integrated to allow EO/IR sensor imagery and Key-Length-Value (KLV)[2] data to be paired in a way that the WFN could manipulate the data and the CODE software could consume it. This encoder was used in its commercial off-the-shelf configuration.

[1] Piccolo Autopilot is a Cloud Cap Technology Product. More information can be found at http://www.cloudcaptech.com/products/auto-pilots

[2] KLV is defined in SMPTE 336M-2007 (Data Encoding Protocol Using Key-Length Value), approved by the Society of Motion Picture and Television Engineers.

The CODE Mission Computer was a high-end, single board, Mini-ITX Computer that had the necessary processor, memory, solid-state drive, power supply, ports, and connectors to support the CODE software and integration. The processor was chosen so that almost any selected operating system and CODE software could run on it. The CODE software was loaded onto this computer before flight and then commanded to run during the test. The CODE Mission Computer consisted of Commercial-off-the-shelf (COTS) parts that were specifically packaged for the CODE program.

All of these components were integrated together onto a single avionics tray in the RQ-23 main equipment bay. The logical flow of these components is shown in Figure 8.
The flow of commands, data, and radio traffic both into and out of the CODE software on the CODE Mission Computer flow through the WFN Air software. This, in conjunction with the WFN Ground software, allows the data and radio traffic going into the CODE software and the commands and radio traffic coming from the CODE Software to be monitored, manipulated, or deleted. This also provides the capability to force the CODE software to deal with specific situations that could not be replicated in the real world. For example, actual UAS loss due to missile engagements, communication disrupted due to radio jammers, etc. could be simulated. When CODE software output is sent to the WFN, the WFN Air software manipulates the command if necessary and sends the commands to the Watchdog for processing. As information from the RQ-23 is passed from the Watchdog, the WFN Air software can also manipulate or delete this information before passing it to the CODE software for processing.

CONOPS

The testing Concept of Operations (CONOPS) was implemented by a range of supporting personnel with different responsibilities. The Test Conductor (TC) had overall responsibility for the safe, efficient, and effective conduct of the test. The TC orchestrated the other personnel supporting the test to meet these outcomes. The TC was responsible for understanding the SUT, the CODE software; the systems used to support the test, the RQ-23, the WFN, etc.; and the test objectives that were to be met. The WFN Operator was responsible for operating the WFN under the TC's direction to establish specific environments necessary to meet the test objectives. This included settings both on-board the aircraft and in the ground system. Additionally, the WFN Operator configured the WFN Graphic User Interface (GUI) so that real time insight was gained on how the test events were progressing and reported that progress to the TC as needed. The RQ-23 crew consisted of a pilot, sensor operator, and Mission Commander that were responsible for the safe flight of the RQ-23. Additionally, they were responsible for ensuring the hardware and software configuration of any non-SUT systems on the aircraft were set properly. The RQ-23 crew ensured that the aircraft was in the proper position, altitude, and airspeed necessary to start each test run so that the test objectives could be met. The crew also provided vital real-time truth data feedback to the TC and other test conduct personnel as to how the test was progressing and what was happening to the aircraft. Finally, there were the operators for the SUT – the CODE software itself. This differed based on the performing contractor's implementation. The operators ensured that the SUT was properly configured to meet the test objective. Their primary role was not to make sure that the SUT operated perfectly, but rather that it was operating within acceptable parameters to evaluate the SUT. That is, they were not constantly tuning the SUT to perform better in real-time, but rather to make sure that the system was operating properly and allowed the system to be evaluated in flight. All of the test personnel had specific responsibilities to ensure the tests were completed in a safe, efficient, and effective manner. They all had to understand the overall test construct and where they fit into that construct in order to function properly.

RESULTS

The testing of the CODE Phase 2 system was performed at the Naval Air Warfare Center Weapons Division (NAWCWD) in China Lake CA. In preparation for these tests, each CODE performer developed a detailed test plan with detailed methods of test (DMOTs) for every required capability. A run matrix was also provided that mapped the DMOTs to the flight test schedule. The flight testing program occurred over a two-month interval (April-May 2017) and included two major test events for each of the two CODE performers.

A notional DMOT is illustrated in Figure 9. The outer polygon depicts the playbox where RQ-23 flight operations took place. The dots represent both ingress (yellow) and egress (green) waypoints preprogrammed into the CODE mission computer. The circles represent obstacles for the CODE autonomy algorithms to recognize and avoid. The triangles represent point of interest where the RQ-23 may loiter to gather information before continuing its flight path.

Finally, the star represents a constructive communications or GPS jamming platform to initiate the appropriate WFN stimulus to the CODE autonomy.

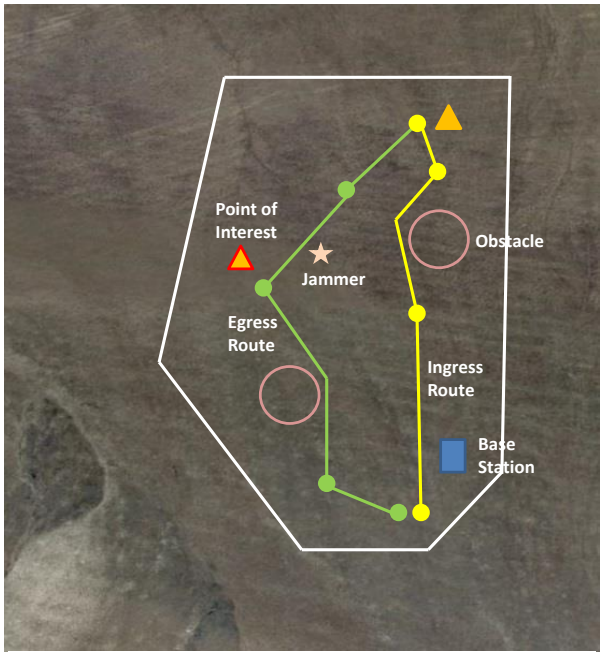

Figure 9 - Example CODE Test Scenario

The ground component of the WFN was located in the base station in the lower right corner of the playbox. Depending on the specific test being performed, the WFN was tasked with instantiating and modeling the virtual and constructive simulation entities needed for the test, including virtual RQ-23s configured with the CODE autonomy. The WFN also modeled the jammers for the opposing forces and created the communications or GPS lost conditions needed to verify the ability of the CODE platform to operate effectively in denied environments.

In addition to simulating the operational conditions needed to test the CODE autonomy, the WFN was also tasked with the collection of real-time test data. For example, the WFN ground component recorded the relative geometry of all CODE-equipped entities with respect to all active jammers to ensure that the desired communication and GPS denied states were implemented correctly. The air component of the WFN recorded all incoming STANAG 4586 message traffic as well as waypoint data and aircraft state information. The analysis of this data was instrumental in verifying that WFN requirements had been met as well as providing insight on the relative performance of the two competing CODE designs.

This testing was the first time that a LVC environment was used to test an autonomy system on a Class 3 UAS. The system met all of the requirements derived from the SUT test scenario requirements. This success was not without challenges and lessons learned. One of the biggest challenges that dogged the program and flight test through the end of this phase was the rapid evolution of Operating System and CODE Mission Computer internal configurations. For instance, ensuring that Firewalls and Ports were all configured properly and consistently was a bookkeeping challenge throughout the development and flight tests.

CONCLUSION

The CODE Phase 2 Program has successfully demonstrated the collaborative autonomy capabilities necessary for CODE-equipped UAVs to operate effectively in communications or GPS-denied environments. The WFN was a critical component of the broader CODE architecture, as the WFN provided the LVC infrastructure needed to create the desired test conditions and stimulate the CODE autonomy during a series of live technology demonstrations. This capability directly supported the verification of key CODE operational capabilities.

The WFN will continue to evolve and mature based on new capability requirements for CODE Phase 3 as well as providing the synthetic forces representation for what will be a much more complex operational environment in this next phase. In addition to an increase in the number of CODE-equipped UAVs (both live and virtual) in the CODE Phase 3 scenario, the need to accurately capture system behaviors and performance across a wider range of neutral and opposing forces requires a more sophisticated and capable SFG than was used in CODE Phase 2. It is currently envisioned that Real-Time Suppressor (RTS) will fill this need due to the availability of data sources/files that can be leveraged from present users.

The WFN represents a potentially reusable capability that could be leveraged by future intelligent autonomous system programs. Although currently configured for CODE, the WFN has been designed to interface seamlessly with future autonomous systems employing “open architecture” standards. While not entirely “plug and play”, the WFN provides a robust LVC foundational capability that can be readily extended to address the needs of collaborative autonomy S&T or actual production systems in the future.

The Defense Science Board (DSB) and the OSD Autonomy Community of Interest Test, Evaluation, Verification and Validation working group (OSD ACOI ATEVV) state that a suite of tools and methods is required to test autonomous systems. Multi-step autonomy test processes are being developed by the OSD Autonomy Community of Interest, Test Resource Management Center, Air Force Research Laboratory and the Naval Air Warfare Center. These test processes include LVC testing as a necessary portion of autonomous systems testing. Once complete, it is anticipated that these processes will be adopted by the Major Range and Test Facility bases (MRTFB) for future autonomous system and that the WFN will become standard equipment for MRTFB hosted for autonomous system tests.

ACKNOWLEDGEMENTS

The White Force Network was developed for DARPA under contract HR0011-12-D-0001 and extends the TACE system, which was developed for the Test Resource Management Center under contract W900KK-13-C-0036. The U.S. Department of Defense enjoys Unlimited rights to TACE and WFN. The authors would like to thank JC Ledé and Reed Young for their steadfast support, Dwight Cass and Bill D'Amico for their valuable technical insight and advice and the WFN software engineering team Kristi Ramachandran, Geoff Osier, Michael Biggins, Robert Chalmers, Lee Varanyak, Chris Dohoposki and Frank Harris.

REFERENCES

Huang, H. (2004), "Autonomy Levels for Unmanned Systems (ALFUS) – Version 1.1", National Institute of Standards and Technology (NIST).

Scheidt, D. (2014a), Unmanned Air Vehicle Command and Control, *Handbook of Unmanned Air Vehicles*, Springer-Verlag.

Allis, V. (1994). Searching for Solutions in Games and Artificial Intelligence (PDF). Ph.D. Thesis, University of Limburg, Maastricht, The Netherlands. ISBN 90-900748-8-0

Schaeffer, A. (2015) Autonomy Test and Evaluation, Verification and Validation Strategy 2015-2018, OASD R&E.

Torens, C., Adolf, F. (2014), "V&V of Automated mission planning for UAS", SCI-274 Workshop Verification and Validation of Autonomous Systems, Imperial College, London, June 24-25.

Scheidt, D., D'Amico, W., Lutz, R., (2014b). Safe Testing of Autonomy in Complex, Interactive Environments (TACE), The International Test and Evaluation Association (ITEA) Journal, Vol. 35-4, pp. 323-331.

Hudgins, G. (2005) "The Test and Training Enabling Architecture, TENA, Offers Range Interoperability and Resource Reuse Solutions", 2005 U.S. Air Force T&E Days, U.S. Air Force T&E Days Conferences

The NATO Standardization Office (5 April 2017) STANAG 4586, Standard Interfaces of UA Control Systems (UCS) for NATO US Interoperability