AFRL-RH-WP-TR-2018-0094

Fusion Framework: Research Testbed for Human-Autonomy Teaming

Allen Rowe, Sarah Spriggs
711 HPW/RHCI

James Boyer, Tom Hughes
InfoSciTex

August 2018

Final Report

AIR FORCE RESEARCH LABORATORY
711TH HUMAN PERFORMANCE WING
AIRMAN SYSTEMS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

ROWE.ALLEN.J.1259846044 Digitally signed by ROWE.ALLEN.J.1259846044 Date: 2019.03.27 11:20:04 -04'00'

ALLEN ROWE
Fusion Program Lead
Supervisory Control and Cognition Branch

PRICE.JOSEPH.C.1092674149 Digitally signed by PRICE.JOSEPH.C.1092674149 Date: 2019.03.28 10:48:19 -04'00'

JOSEPH PRICE, Maj, USAF
Chief, Supervisory Control and Cognition Branch
Warfighter Interface Division

CARTER.LOUISE.ANN.1230249128 Digitally signed by CARTER.LOUISE.ANN.1230249128 Date: 2019.03.28 16:17:10 -04'00'

LOUISE A. CARTER, Ph.D.
Chief, Warfighter Interface Division
711 Human Performance Wing
Air Force Research Laboratory

REPORT DOCUMENTATION PAGE					Form Approved OMB NO. 0704-0188
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YY) July 2018			2. REPORT TYPE Final		3. DATES COVERED (From – To) October 2014 – April 2018
4. TITLE AND SUBTITLE Fusion Framework: Research Testbed for Human-Autonomy Teaming					5a. CONTRACT NUMBER FA8650-14-D-6501 5b. GRANT NUMBER NA 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Allen Rowe, Sarah Spriggs (711 HPW/RHCI) James Boyer, Tom Hughes (InfoSciTex)					5d. PROJECT NUMBER 5e. TASK NUMBER 0006 5f. WORK UNIT NUMBER H0JF
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) InfoSciTex 4027Colonel Glenn Hwy, Beavercreek OH 45431 OH 45431					8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory 711th HPW/RHCI 2210 8th Street, Bldg 146 Wright-Patterson AFB, OH 45433					10. SPONSORING/MONITORING AGENCY ACRONYM(S) 711th HPW/RHCI 11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-RH-WP-TR-2018-0094
12. DISTRIBUTION/AVAILABILITY STATEMENT Distribution A Approved for public release. Distribution is unlimited.					
13. SUPPLEMENTARY NOTES Cleared, 88PA, Case 88ABW-1812					
14. ABSTRACT (Maximum 200 words) Future unmanned systems operators and heterogeneous unmanned systems must be able to work as agile synchronous teams to complete tactical reconnaissance, surveillance, and target acquisition related missions. Interface research in this area is critical to the success of human-automation teaming, thus requiring a research test bed that brings together humans, autonomy, and systems. Fusion is a framework that enables natural human interaction with flexible and adaptive automation. There are four primary research threads that the Fusion framework is addressing to accomplish these goals: cloud-based simulation architecture, software extensibility, interface instrumentation, and human-autonomy dialog through retrospection. The current report documents the Fusion Framework's features and the motivation for its development.					
15. SUBJECT TERMS (U) Autonomy, Human-Autonomy Interface, Human-Autonomy Teaming, Operator Interface, Testbed, Software Frameworks, Software Architecture, Service Oriented Architecture					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified	Unlimited	28	Allen Rowe 19b. TELEPHONE NUMBER (Include Area C 937-713-7028

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

Table of Contents

List of Figures

1.0 Background

The research, development and acquisition landscape is littered with attempts at automated capabilities that have failed to live up to their promise and potential. Much of this failure can be attributed not to the capabilities of the autonomy or automation themselves, but rather the limited capacity of these systems to effectively adapt to the complexities and dynamics inherent in real-world operations. Operators often reject such systems because of 1) their inability to fully understand autonomous reasoning processes (observability), 2) the limited competency boundaries of these systems and the operators' ability to effectively direct these systems in response to these boundaries (directability), and 3) the uncertainty associated with how these systems may behave in highly consequential situations (predictability). Many of the autonomous and automated systems previously deployed have failed to appreciate the critical role that human-machine communication and coordination plays in successful human-autonomy teaming. Support for such communication and coordination has been a primary design driver in the development of the Fusion Framework. In fact, the primary motivation for the development of the Fusion Framework was the development of a simulation testbed built to support research into human-autonomy teaming. Research directed at gaining greater insight into the socio-technical interaction between human-operators and their autonomous teammates is the foundation upon which Fusion was conceived.

Automation, and particularly autonomous systems, have been gaining popularity in large part due to 1) recent advances in the field of artificial intelligence and machine learning (technology push) and 2) increased operational demands for more capable and adaptive systems to deal with the real challenges of complex operational situations (requirements pull). Autonomy is a prominent tenant of the DoD's recent Third Offset Strategy which emphasizes the need to create increased efficiencies and effectiveness in our ability to project power in an increasingly complex, dynamic and diverse national security environment. One of the focuses of this strategy is the increased incorporation of unmanned systems and their associated autonomy required to effectively adapt in the face of complex operational situations and to combat anticipated vulnerabilities in datalinks within Anti-Access Area Denial (A2AD) environments (Martinage, 2014). Fusion has been developed to explore the technical, operational, and organizational challenges associated with increased autonomy and the means for supporting a more robust and resilient form of coordination between human operators and these autonomous systems.

The Fusion Framework was conceived to serve as a testbed that would enable researchers the ability to explore this space in a meaningful manner. To achieve this objective, Fusion had to 1) facilitate the integration of automation/autonomous services from a broad range of developmental organizations serving a broad range of operational activities, 2) support flexibility and creativity in the development of operator interfaces allowing designers and analysts the opportunity to rapidly prototype and assess alternative design concepts, and 3) enable reliable assessment of these concepts providing insights that will serve as feedback in the refinement of autonomous services and the interfaces that support operator-autonomy communication and coordination.

2.0 Tenants

The Fusion framework was built upon the fundamental tenant of supporting the integration and coordination across humans, autonomy and systems. In the context of complex operational environments characterized by highly dynamic situations, uncertainty and conflicting goals, understanding the interdependencies that are inherent in the systems and capabilities used to operate in these environments is critical to successful system performance. The Fusion Framework was envisioned as a testbed that would enable researchers the ability to investigate issues and challenges that reside at the intersection of human operators, autonomous agents, and the systems under their control.

Several challenges have emerged as we have explored the introduction of autonomous systems into the realm of complex military operations. Foremost among these challenges is supporting the coordination demands between human operators and their autonomous counterparts. Human-autonomy interaction or teaming has become a major research thrust across a broad range of operational domains (e.g., autonomous cars, manufacturing, package delivery, emergency response, etc.). The key to realizing the force multiplying potential of autonomy, particularly within complex operational environments, is understanding and resolving how to effectively support coordination across human and autonomous agents.

3.0 Objectives

3.1 Testbed for Human-Autonomy Interaction

In 2014 the Assistant Secretary of Defense Research and Engineering (ASD R&E) office initiated a series and Autonomy Research Pilot Initiatives (ARPI). The goals of these initiatives were to 1) identify key laboratory initiatives in autonomy, 2) strengthen community interest and improve communication across initiatives and 3) increase joint collaboration in autonomy research across the Department of Defense (DoD). In response to the ARPI, development of the Fusion Framework was initiated, and supported by the ARPI initially. Development and continued refinement of the Fusion Framework is motivated by the desire to explore interfaces that better support human-autonomy coordination and collaboration by introducing immersive dialogs, task delegation, cooperative planners, intelligent agents, goal-reasoners, and machine learning into a comprehensive autonomy framework.

As a research organization, the Airmen Systems Directorate Supervisory Control and Cognition) branch of the 711th Human Performance Wing within the Air Force Research Laboratory (AFRL) was working to establish a human factors testbed to enable 1) rapid prototype of human-machine interfaces (HMI) for human-autonomy teaming, 2) robust instrumentation to include human physiological instrumentation and human interaction with the HMI, 3) rich data sets so we can determine the effectiveness of the interfaces and the ability to support various system performance measures. Our goal was to explore novel concepts with a fully instrumented system tailored to the needs of research scientists that could provide the necessary feedback further informing the design process.

The Fusion developers identified four key focus areas that endeavored to answer the following questions:

- How can a software system create a framework where every public element, regardless of its role as a model, user-interface element, etc., is customizable, Extendable and override-able by any other software developer in the system?
- How can a software system generalize disparate and similar messaging protocols to be protocol-agnostic while allowing a many-to-many relationship between Cloud-based Networked Systems for the generation, distribution and consumption of messages?
- How can a software system be Instrumented to gather real-time user/machine interactions and system details for use in experimentation, software agents, and machine learning?
- How can a software system record the state of each of its components at a rate near 30Hz and make it user-accessible to enable discrete and continuous Retrospection of the system in real-time?

Recognizing there existed a gap in these areas drove the team toward a new architecture that would support the research needs for innovative development of human-autonomy interactions and teaming. The following sections explore each of these focus areas in more detail.

3.1.1 Software Extensibility

Fusion is being used in several different projects, all of which share the goal of improving operator interactions with highly autonomous systems, but have vastly different HMI designs and algorithms. Due to this, Fusion was built with the goal of extensibility.

Fusion's infrastructure allows developers to override aspects of the HMI by adopting a layered architecture in which the framework contains the building blocks for HMI tools and services. Developers can add new HMI tools and services by overriding those building blocks and developing new modules. Thus, developers can override or extend aspects of Fusion without altering the original or previous extensions. Modules can be either universal or project-specific. Through this architecture, the user can choose which modules are loaded, and therefore affect how the Fusion user interface looks and reacts.

One example of the extensibility currently realized in Fusion is the vehicle symbol. In test beds that allow operators to control or supervise unmanned systems, vehicle symbols are important and appear in many different places in the user interface. Within Fusion, vehicle symbols appear on the map, in various notifications, on the vehicle status tool, in tasking tools, in many project specific tools, and other places.

Most projects have their own vehicle symbol design and it typically requires a lot of work to replace that symbol everywhere for every project. Fusion was built with this in mind, therefore, there is a default simplistic vehicle symbol included in the framework. Every project has the ability to design and implement their own vehicle symbol. With a single line of code in the project-specific vehicle symbol, all vehicle symbols in the entire Fusion test bed can be replaced. An additional benefit allows developers to no longer need to consider this during implementation: the framework handles it at run time.

Extensibility saves a great amount of development time and allows designers to explore multiple design solutions. A user interface can be designed and implemented in multiple ways and, depending on which modules the user loads, a specific design is realized. This also facilitates experimentation to explore tradeoffs between alternative designs.

3.1.2 Cloud-based Network System

Fusion has established an application programming interface (API) for external software components to communicate and interact with Fusion within a service oriented architecture. To date, vehicle simulations, intelligent task allocation agents, vehicle planners, speech interpreters, chat systems, sensor visualizations, operator assistance components, map layer data, and monitoring components have been incorporated into the Fusion network API. These networked components employ various messaging transport layer protocols (e.g., UDP, TCP/IP, ZeroMQ) and communicate using a variety of messaging serialization formats (Distributed Interactive Simulation (DIS), AFRL's Lightweight Message Construction Protocol (LMCP), Javascript Object Notation (JSON), and custom). In some form, all the components are linked together in their communications modalities by use of a publish/subscribe hub. Where appropriate, the connections and protocols are also realized into appropriate interface components in Fusion, and are intended to aid in creating a more immersive and interactive system for human-autonomy teaming.

The goal of this network API is to make the incorporation of external software as transparent and natural as possible while leveraging data efficiently. All of the instrumentation data (discussed in detail later in this section) is published to the hub, and any component that wishes to consume the data can do so with a simple subscription. Likewise, communication messages from the other components are published to the same hub, and Fusion (or any other component) can subscribe and receive those messages. Each of the networked components may also communicate with another networked component using this same structure. The Fusion team has worked closely with developers of other software components in order to ensure a seamless integration. The publish/subscribe architecture present on the communications hub makes for a natural assembly: all the associated data published by any software entity is available to any other entity that needs to leverage it, thus enabling great flexibility in the potential interactions between the entities, including Fusion and its operator(s). It also establishes the framework that will be needed for our near-future extension of Fusion to a MOMU (multiple operator/multiple unmanned system) interface.

With Fusion as an operator's interface, the communication with the components are more transparent and natural. For example, in one of the programs that Fusion has recently supported, a user can define high-level goals that Fusion dispatches to an intelligent agent. The agent allocates resources to be used for the achievement of the goals and submits the realized requirement to the resource planner. The planner reports the plan back to Fusion, which is then shown as a candidate solution. When accepted, the plan is delivered to the resource simulation for execution, while another component monitors the plan execution to alert the operator in case of issues. Each of these components communicate using a different set of message protocols. To the operator, it appears as if all these components are part of Fusion: the operator defines a goal and approves a plan, and it appears to the operator that the deliberative activity occurs exclusively within Fusion. In this way, Fusion enhanced the operator's teaming with the autonomous system components while ensuring that the communication and feedback are transparent to the operator. Since the operator is able to define high-level goals for the project, the system enables a realization of an enhancement in the dialog between the operator and the various software entities that provide access and control of autonomous systems. This enhancement was exercised in an initial evaluation and received positive feedback from all the subjects. The flexibility and extensibility of the communications/network framework have provided a baseline from which the development team will further enhance human-machine teaming, creating a more immersive and flexible interactive environment.

3.1.3 Interface Instrumentation

Data collection, agents, and machine learning all require capturing of data, which must be stored or packaged up and sent across the network. User interface interactions are one of these critical data sources. This capability was built into the Fusion framework and serves as a fundamental capability enabling the exploration of key human-machine interactions. It is fairly non-invasive to the developers and provides a host of information, both in real time and post-hoc. Every user interaction, such as button clicks, typing, and mouse clicks are recorded and saved to a file.

All instrumented data is also packaged up and sent through the network to any agent, machine learning algorithm, cognitive modeling service, or other automated service that is subscribing to the data source. Instrumentation of all operator interactions is critical for human-autonomy teaming. There are many uses for this feature of Fusion. Agents can leverage it to better understand user behavior and take or recommend actions. Machine learning employs instrumentation data to learn how individual users perform, and potentially recommend an interface change either after the fact or in real time (e.g., if a user uses certain buttons more often, the buttons can be reorganized to better suit their use). The data can inform cognitive modeling services, improving researchers' understanding of how the operators are performing.

During evaluations, all instrumentation data are recorded into a comma delimited log file, allowing the experimenter to analyze performance data. Reaction times and accuracies based on the times of various user actions saved in the file can then be determined. This was utilized in an initial evaluation for one of the early applications of the Fusion framework. For example, a chat module was employed to request tasks from the user. The chat requests and responses were instrumented, as were the user reactions. The experimenter leveraged this data to analyze how quickly and effectively the task was performed. The experimenter also noted any extra steps the operator performed, the sequence of steps taken, and the modality of their actions. All data were used to analyze and improve the user interface as well as any automated services.

3.1.4 Human-Autonomy Dialog through Retrospection

All of the instrumentation data can be used for retrospection. Since all the data is stored, it is natural to allow it to be re-played post process or played back during runtime. Retrospection has two main applications (and potentially more): experimenters can observe what was occurring to analyze why an operator performed an action or series of actions, and operators can "pause" and "rewind" the scenario to get another look at something that occurred in the past, further enhancing the human-autonomy dialog.

The concept of retrospection however goes beyond the notion of "replay" of the scenario. Retrospection includes not only the ability to replay the scenario but also at any point during replay to terminate the injection of recorded state data and proceed with real-time operator inputs. This allows analysts to execute a broad range of "what-if" scenarios. How would the situation have played out if the system chose this course of action over another course of action? It is envisioned that the capabilities made available through the retrospection feature will allow both analysts and potential operators the ability to gain greater insight into the reasoning processes of the autonomy services. Specifically, this feature will help address challenges associated with the implementation of autonomous systems. One challenge is the "black box" effect. A "black box" obscures the internal mechanisms providing insight only into its inputs and resultant outputs. Given the nature of advanced autonomous systems, courses of actions recommended by the system may run counter to operator expectations prompting operators to challenge the recommendations. The "black box" nature of the autonomy has no explanation to provide and thus in many cases operators may reject the recommendation merely because they do not understand the reasoning behind it or is inconsistent with expectations (with a potential valid reason, the autonomy may have access to or be able to conduct analysis on data that is unavailable to operators or operators are unable to effectively analyze).

Likewise, another challenge is, upon analysts identifying situations that lie at the edge of the autonomy's competency boundary, exploring or experimenting with the reasoning processes of the autonomy. Use of this feature may allow an analyst to better understand the demands and constraints of the problem, the tradeoffs that may be required, and refine algorithms in an effort to expand or extend the overall robustness of the system and the competency boundary of the algorithm. Retrospection has been envisioned as a tool that will allow stakeholders to explore and discover the decision space upon which autonomous systems will be engaged.

4.0 Fusion Applications

4.1 Autonomy at Rest

4.1.1 IMPACT

As previously noted, the Fusion Framework was formed to support multiple ASD R&E ARPI projects (see Section 3.0; two described herein). The purpose of this initiative was to foster research and push the envelope for autonomy-based research. One such project, Realizing Autonomy via Intelligent Adaptive Hybrid Control, developed an "Intelligent Multi-UxV Planner with Adaptive Collaborative/Control Technologies" (IMPACT). This was a three year effort (fiscal years 2014-2016) with a focus on maximizing human-autonomy team agility. The first year of this effort focused on designing and implementing the user interface for higher level, goal-oriented plays (analogous to the sports ontology), which included asset management and integrating the various autonomous components. This "play" centric concept allows operators to focus on higher level goals for the vehicles, leveraging autonomous aids in accomplishing those goals, thus reducing the need for the user to direct and control vehicles manually.

The IMPACT instantiation of Fusion (Figure 1) employs a four screen layout: system tools, real time tactical situation awareness (SA) display, sandbox tactical SA display, and sensor management. The operator uses the sandbox display to perform all of their play calling and chat monitoring tasks. The other screens display tools to enhance the operator's SA.

All of the goals of the Fusion framework were critical to the success of the first year of IMPACT. Fusion, autonomous agents, external simulations, a cooperative control planner, and machine learning algorithms were combined to form a comprehensive, richly interactive environment. This was enabled by Fusion's flexible software architecture through a robust simulation environment, software extensibility, and interface instrumentation.

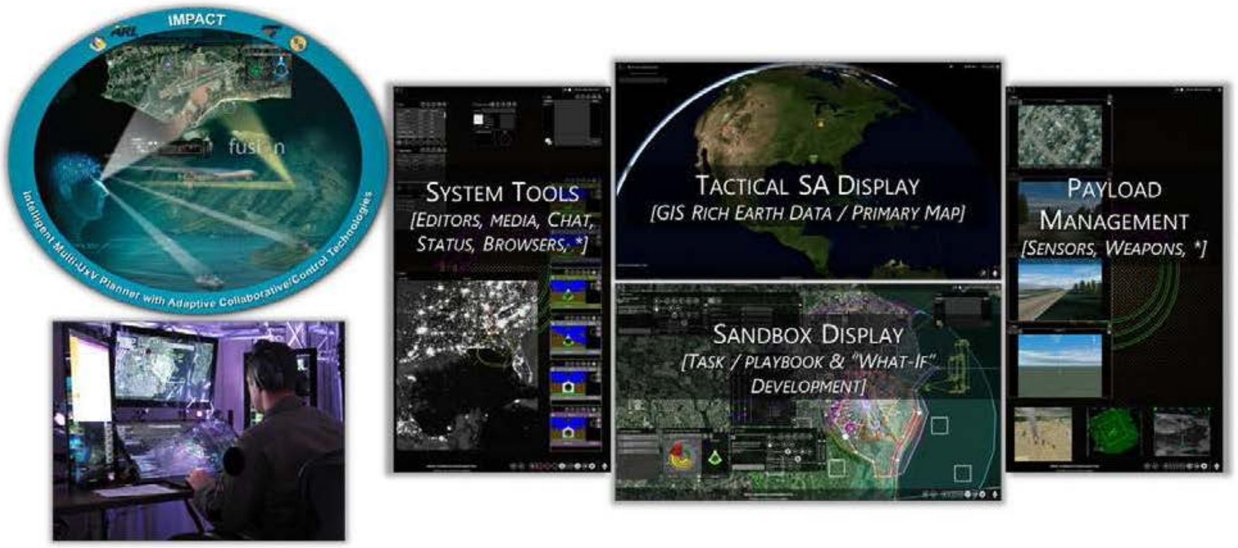

IMPACT
fusion
Intelligent Multi-UxV Planner with Adaptive Collaborative/Control Technologies
SYSTEM TOOLS
[EDITORS, MEDIA, CHAT, STATUS, BROWSERS, *]
TACTICAL SA DISPLAY
[GIS RICH EARTH DATA / PRIMARY MAP]
PAYLOAD MANAGEMENT
[SENSORS, WEAPONS, *]
SANDBOX DISPLAY
[TASK / PLAYBOOK & "WHAT-IF" DEVELOPMENT]

Figure 1. IMPACT Instantiation of Fusion

4.1.2 SWAMPED

In order to realize the full potential of mixed airman and machine teams, airman workload must be managed to prevent overload that could degrade team performance. A first step to achieving this goal is measuring operator workload and projecting it into the near future, enabling proactive management. Existing workload measures suffer from a number of problems making them unsuitable for this purpose.

AFRL funded a multi-directorate effort to augment traditional physiological and performance data analyses with a cognitive model, allowing integration of measures for a more robust multidimensional workload metric. For this effort, we were responsible for the collection and data analysis using the IMPACT system to provide operator performance measures. These performance measures, together with capabilities from other AFRL teams (behavior measures, physiological measures, and cognitive modeling), aim to: 1) better understand the complex relationship between an operator's physiological and behavioral measures with performance and workload associated with a human completing multiple tasks, and 2) apply cognitive modeling techniques to help achieve assessment and prediction capabilities that could be leveraged to inform dynamic workload management through adaptive task allocation. This project, referred to as "SWAMPED" (System for Workload Assessment and Monitoring for Predicting Effective Decision Making), is illustrated in Figure 2's schematic and is described in more detail via this reference (Stevens, et al., in press).

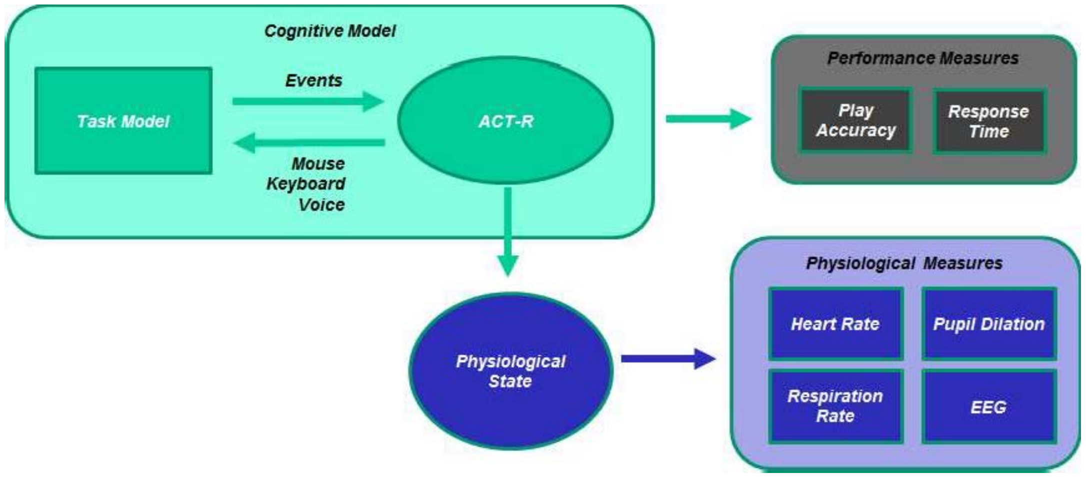


Figure 2. Conceptual Depiction of SWAMPED Effort (Stevens, et al., in press)

4.2 Autonomy in Motion

4.2.1 ATACM

US Forces must be able to conduct air combat operations in denied areas with highly dynamic, uncertain, and adversarial conditions involving coordinated air-to-air and surface-to-air threats. One concept for dealing with this challenge is the introduction of mixed manned and unmanned air combat teams. Autonomy capabilities to enable cross-platform coordination and operations within an adversary's planning/decision/action timeline in an A2AD environment will be required, and have been called for in numerous recent DoD and USAF planning documents (Ernest, et.al, 2016). The DoD Unmanned Systems Roadmap specifically lists development of technologies and tactics for manned/unmanned teaming as a critical challenge. The objective of the Autonomy for Air Combat Missions (ATACM) project was to develop critical autonomous decision and machine learning technologies to enable manned/unmanned air combat teams to operate in highly contested environments, and integrate them into a tactical battle manager (TBM). The TBM enables a single human to manage the team and coordinate unmanned air vehicles with manned aircraft (we refer to these as mixed teams). To develop the TBM, refine its capabilities, and assess its combat effectiveness, the ATACM project used high fidelity constructive and virtual analysis and training simulators for air combat missions using the Fusion framework to also create a pilot-vehicle interface to support pilot interaction and coordination with a set of unmanned wingmen. The project included a diverse, multi-domain team of experts from the United States Air Force and Navy.

Since its initiation in 2013, the ATACM project pursued a spiral development approach, integrating automation development and Pilot Vehicle Interface (PVI) design to enable a single pilot to command multiple unmanned wingmen from the cockpit of a fighter aircraft. A series of simulation studies were conducted in order to explore the feasibility and potential operational utility of a mixed manned-unmanned configuration within the context of Offensive Counter-Air (OCA) operations. The primary objective of each study was to investigate human-machine interaction and teaming, along with different levels of system autonomy in the execution of OCA scenarios. Secondary objectives include:

- Assess the extent to which support for effective human-autonomy coordination is achieved
- Assess the pilot's workload and situation awareness when controlling unmanned wingmen
- Assess operator/system performance between manned and unmanned wingman across various levels of wingman autonomy
- Solicit operator feedback regarding the quality of the operator interface relative to directing the autonomous wingman and the feedback received from the autonomous wingman
- Assess the allocation of mission functions between pilot and automation
- Solicit pilot feedback on the level of observability and controllability of wingman operations
- Solicit pilot input on novel ways to employ an unmanned aircraft as a wingman

5.0 Fusion Architecture

Fusion is a framework that enables natural human interaction with flexible and adaptive automation. It employs multiple components: intelligent agents' reasoning among disparate domain knowledge sources (Douglass & Mittal, 2013); machine learning that provides monitoring services and intelligent aids to the operator (Vernacsics & Lange, 2013), cooperative planners (Kingston, et al., 2009), and advanced simulation via an instrumented, goal-oriented operator interface (Calhoun, et al., 2017). These empower scientific experimentation and advancement across multiple autonomous technologies (Figure 3).

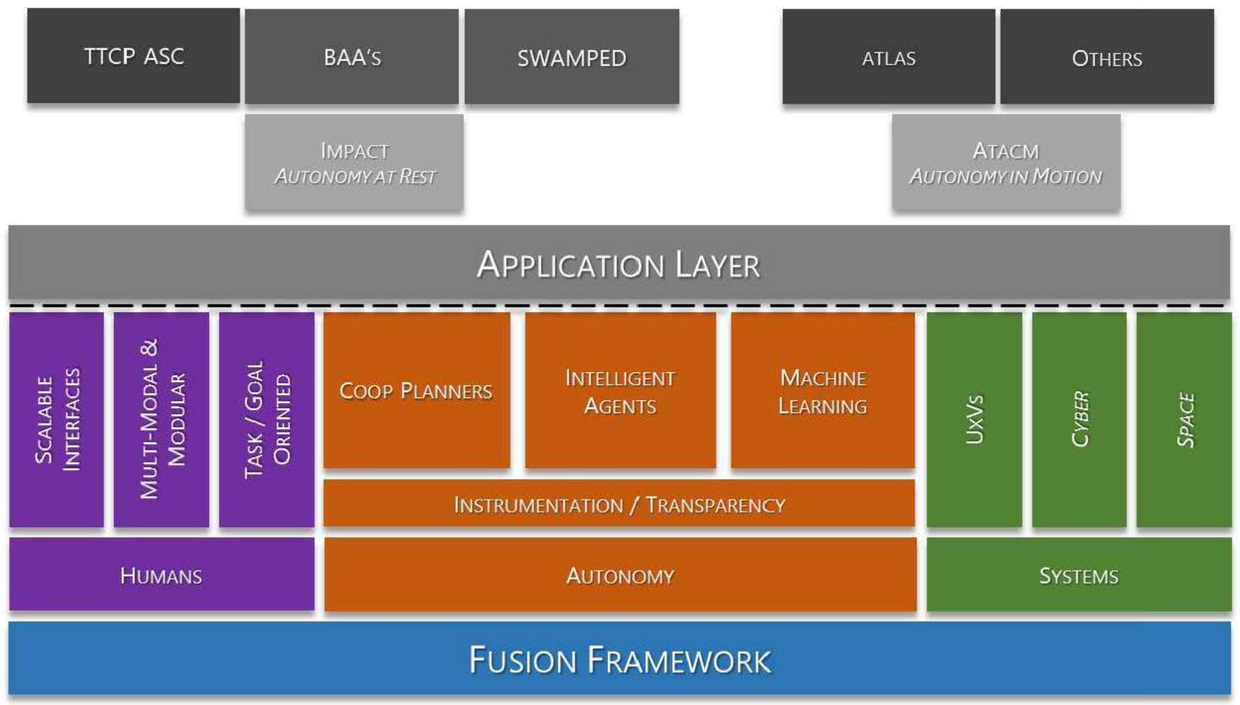


Figure 3. Fusion High Level Framework for Current and Future Applications

The Fusion Framework is essentially a Software Development Kit that provides generic extensible code to enable autonomy based research. The framework consists of four fundamental layers: (a) the core framework layer, (b) the extensibility and API layer, (c) the module / messaging layer, and (d) the application layer (Figure 4). The core framework layer provides foundational software classes and an API. This layer enables functionality for module lifecycle, user profile, and display layout management. Additional features of this layer include system level notifications, multi-modal interactions and feedback, workspace management, asset management (vehicles, tracks, sensors, named areas of interest, etc.), geospatial information services (GIS) data and earth mapping capability, and user interface elements. All software modules have a public framework API to support interface extensibility. This is accomplished in the extensibility and API framework layer. The module and messaging layer contains code written for single and specific purposes. This is the layer that contains user interfaces, utility classes, and messaging protocol support for communication to external software components. Finally, the application layer contains code related to executable applications such as a test-bed, utility application, or test operator console. All code is written utilizing agile software development principles, namely the SOLID principles (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion) (Martin, 2012).

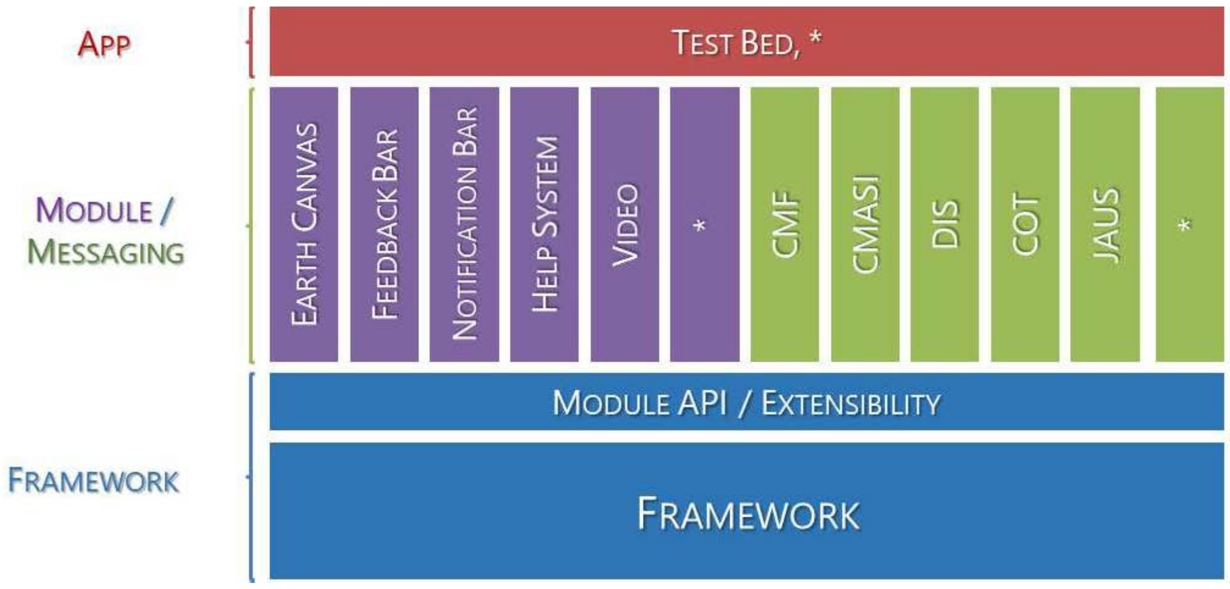


Figure 4. Fusion Layered Architecture

5.1 Framework

5.1.1 Features

While most of Fusion is completely customizable, there are a few core aspects that are common across all projects. Each project maintains a scenario that basically contains the instructions on which modules should be loaded as well as how Fusion should look and operate. The Fusion visual framework is broken into six key concepts: (a) Login, (b) Layout, (c) Notification, (d) Feedback, (e) Canvas, and (f) Tiles/Views (Figure 5). All of the core user interface items can be customized to a certain extent as well as turned on or off to suit the needs of this project

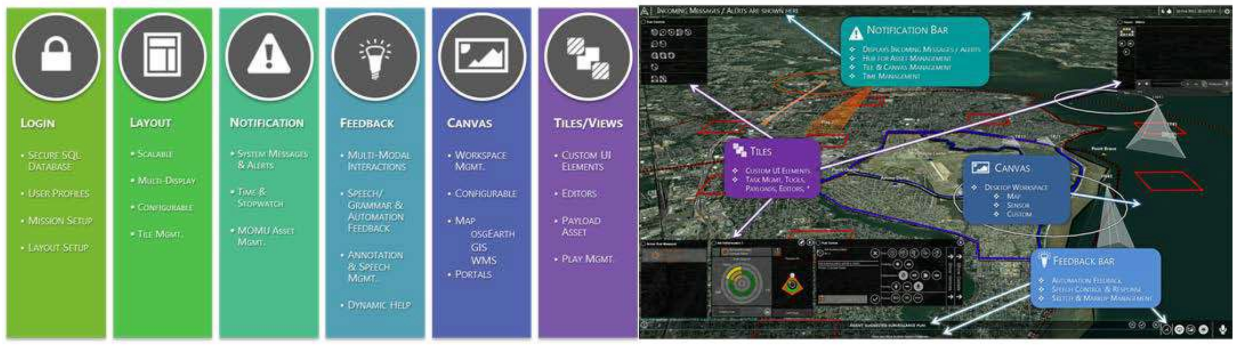


Figure 5. Fusion Visual Framework Components

5.2 Visual Framework

5.2.1 Login/Profile

Fusion requires a user login and profile which contain information about a specific user such as last selected scenario and layout. There are also several key user interface components common across all scenarios, such as screen layouts, canvases, feedback/notification bars, and tiles. All of these are completely configurable to meet the needs of the scenario.

5.2.2 Layout

The layout system gathers information from the operating system on the number of screens as well as their size. To avoid confusion, Fusion internally renumbers the screens based on their top left corner position; ordering is from left to right, top to bottom. This allows the layout to be consistent across machines with potentially different screen layouts and resolution configurations. The Fusion layout per screen identifies which physical screens are to be used, what canvas to show on those screens, if the notification/feedback bars are to be shown, and if that screen should be a sandbox. These concepts will be described in the following sections.

5.2.3 Notification

The notification bar is another core user interface element of Fusion. This bar lives at the top of each screen and can be turned on or off for individual displays. The notification bar contains quick actions, a scrolling notification summary area, an indicator area, a time area, and an expanded notification area. Quick actions are customizable and currently contain map functions (if tied to a map canvas) such as reset map to north up and center on a particular symbol. The scrolling notification area simply provides a quick summary of each notification present. Notifications are used to inform the user of events, such as system alerts, errors, or custom events defined by the developer. These are present in the expended notification area that can be opened by double clicking on the scrolling summary area. The indicator area is also customizable and contains general system indicators such as the vehicles present in the system, whether a multi-modal device is connected, or custom system status indications defined by the developer. The time area contains the current system time as well as a stopwatch. This element is completely extensible by the Fusion developer for specific requirements of the system under design.

5.2.4 Feedback

The feedback bar is similar to the notification bar and is located at the bottom of each screen. This is the area the user receives feedback on their interactions with the system. It is primarily used for automation and multi-modal feedback. The user receives speech command feedback, such as confirmation of what was understood, or speech related errors on the feedback bar. Automated agents or other processes can also feed messages to Fusion and can be displayed in this area. Like the notification bar, the feedback bar contains a hidden expansion area to allow the user to see all of the current feedback items, since only a single item displays on the bar at once for a set amount of time. The full view can be expanded clicking on an indicator that displays the number of feedback items present in the system. This element is completely extensible by the Fusion developer for specific requirement of the system under design.

5.2.5 Canvas

Each screen can have either an earth canvas, a blank canvas, or a custom canvas. The canvas can be thought of as an artist's canvas of which to place widgets developed by the Fusion developer for specific projects. The widgets can be embedded in the canvas itself, such as the earth, or can be a space to place tiles (see Section 5.2.6). Currently fusion contains an earth canvas which hosts OSG earth, a blank canvas, and one custom canvas that hosts video management widgets. Custom canvases can be made to suit any projects' needs. This element is completely extensible by the Fusion developer for specific requirement of the system under design.

5.2.5.1 Mapping

SharpEarth is a three-dimensional (3D) mapping tool used within Fusion to display geospatial data and layers onto a 3D earth. SharpEarth was created as a wrapper in C++/CLI to extend the functionality of the C++ toolkits (OsgEarth and OpenSceneGraph (OSG)) into the C# environment. Extending such massive toolkits allows the framework to have a feature rich map with a huge set of layers draped over the earth such as WMS imagery, elevation data, weather layers, and tiled image layers (tiff, png, jpg). This also allows for the drawing of any 3D object on the map such as shapes, text, icons and indicators. SharpEarth runs completely independent of Fusion by being placed inside of its own thread. Communication between Fusion and SharpEarth occurs through thread safe callback layers. Touch and mouse manipulations of the map are fully supported and can easily be hooked into programmatically allowing complex interactions on the map such as shape manipulations. The earth can be displayed as either a tile or a canvas within Fusion and can be completely customized through an earth configuration file without the need to change the code-base.

5.2.6 Tiles

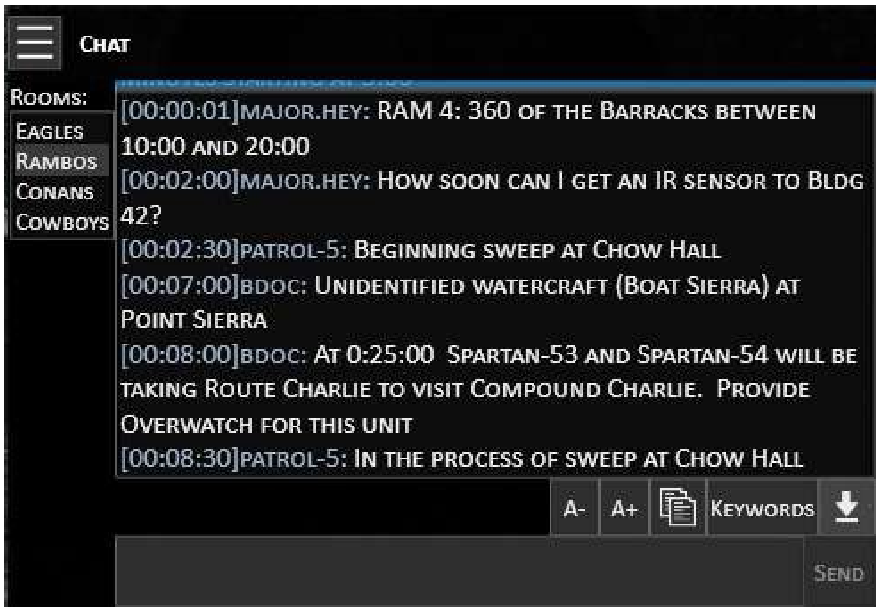
Another key user interface component is the concept of a tile. Tiles are smaller windows that are placed on a canvas that contain widgets to display data to the user or accept user inputs. They are completely customizable and serve as the primary mechanism used to build user interfaces within Fusion. The Fusion Framework includes several core tiles that are described in the following sections.

5.2.6.1 MMC Chat

The multi-modal communications (MMC) chat tile (Figure 6) allows multiple Fusion users and different processes to communicate with one another. There are two components to an MMC tile, the chat server connection management and the user interface.

The MMC framework within Fusion manages connections to an XMPP chat server running as either a local service or a network service. The MMC framework manages chat users and chat rooms from within Fusion. Users are automatically logged in using their Fusion login profile and they have the option of logging in as other users. The MMC framework also handles connection errors as well as providing a way to programmatically send chat messages.

Each tile represents a single chat room which is selected upon creating a MMC tile. The tile is the mechanism by which the user interacts with the chat system by typing in chat messages, engaging speech to text, text to speech, basic text editing functions, specifying keyword highlighting, changing the text to speech volume, assigning a 3D audio location for that chat channel, or playing audio messages. The 3D audio feature allows the user to hear the chat messages in their headset from a variety of spatial locations. The MMC tile has also been extended to support an internet relay chat (IRC) client functionality as well as a server-less broadcast chat functionally.

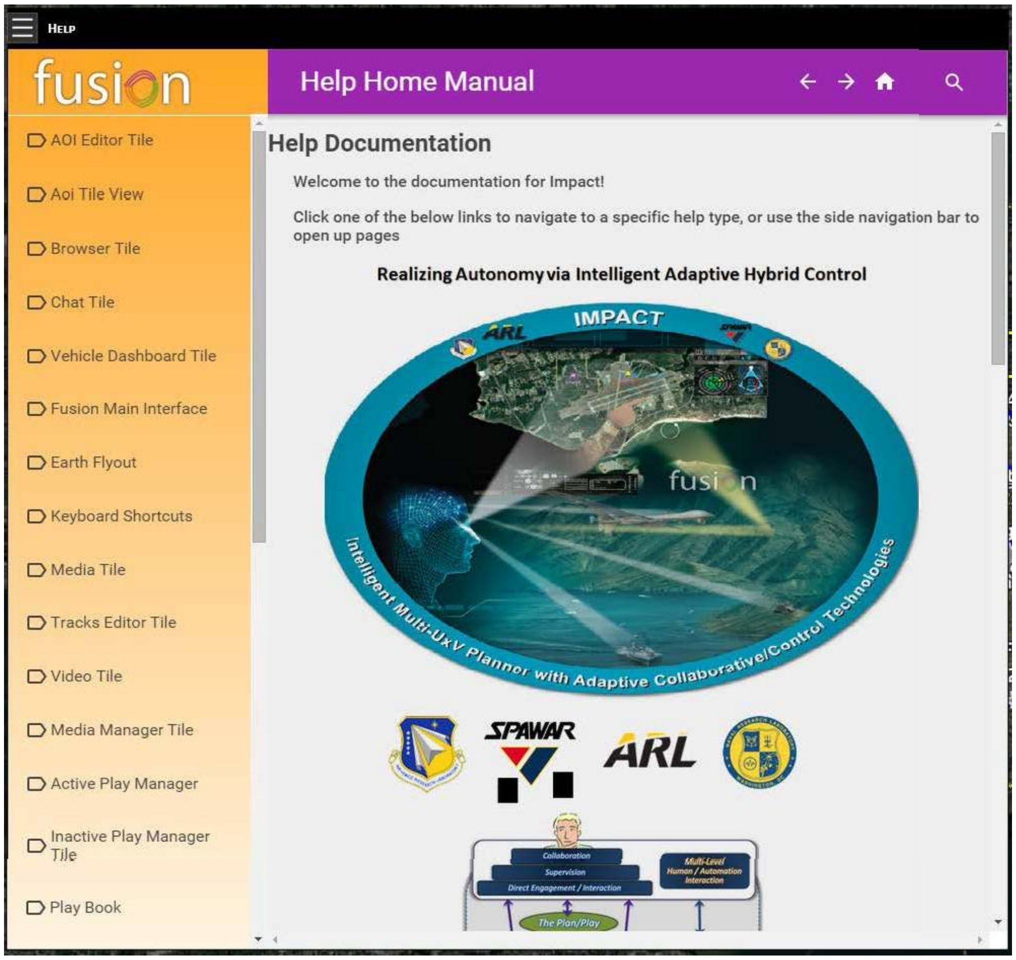


[00:00:01]major.hey: RAM 4: 360 of the Barracks between 10:00 and 20:00
[00:02:00]major.hey: How soon can I get an IR sensor to Bldg 42?
[00:02:30]patrol-5: Beginning sweep at Chow Hall
[00:07:00]bdoc: Unidentified watercraft (Boat Sierra) at Point Sierra
[00:08:00]bdoc: At 0:25:00 Spartan-53 and Spartan-54 will be taking Route Charlie to visit Compound Charlie. Provide Overwatch for this unit
[00:08:30]patrol-5: In the process of sweep at Chow Hall

Figure 6. MMC Chat Tile

5.2.6.2 Integrated Help System

Inherent in the Fusion Framework is the concept of integrated help for all aspects of the user experience. Each user interface component has the ability to specify a popup help tile or tooltip that contains an HTML page with data pertaining specifically to that tile. These are accessed by clicking the question mark button while Fusion is in help mode. Each tile contains a key, which is dynamically created based on tile name at the time of its creation. Each HTML help file contains meta-data used to associate it with certain Fusion concepts. The “helpType” meta-tag describes the type of category of a help page (Tile, Frame, Voice, and Input). The project meta-tag describes the association of a project to its help file.



Help Documentation

Welcome to the documentation for Impact!

Click one of the below links to navigate to a specific help type, or use the side navigation bar to open up pages

Realizing Autonomy via Intelligent Adaptive Hybrid Control

Figure 7. Help Tile

The Help Manual (Figure 7) is a navigable, searchable tile in Fusion utilizing web components. The help manual is dynamically generated based on the HTML help files and the associated meta-tags. The manual injects every HTML file it needs based on the project meta-tag into a book-style web app. It simply displays a list of help files from all related projects loaded for a particular scenario. For example, while running IMPACT the user would see a list of help files relevant to Fusion Core and IMPACT. These same help files are also displayed if clicking a tile’s question mark button in help mode.

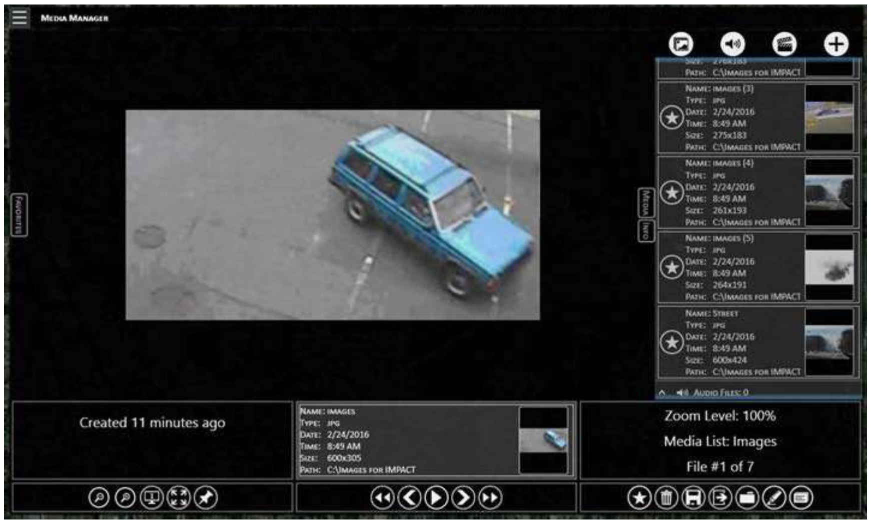
The Help Manual dynamically creates a data store in JavaScript to be utilized as a search tool, since there is no connected database. The search function processes the pages based on keywords the user inputs, and allows them to easily find the associated help page.

5.2.6.3 Media Manager Tile

The media manager tile (Figure 8) allows the user to view and markup images, view videos, and listen to audio files. By default, the media manager monitors the output directory for the current Fusion run and loads any existing media therein as well as any new media that becomes available in this directory during execution of a particular scenario. Additional media directories can also be specified and monitored.

The Main Media Display Panel shows the media that is currently selected. The buttons located at the bottom of the media manager Tile interact with this display area. Image media can be zoomed in this display using the mouse wheel and slewed by dragging with the left mouse button clicked.

Within the media manager, media that can be displayed are shown in individual media tiles. These tiles contain a thumbnail image, an information summary, and a "Favorite" button indicated by a star. Clicking on the thumbnail or information summary sets the specified media as the currently selected media for the media manager and will display it in the Main Media Display Panel. Clicking on the "Favorite" button sets the specified media as a "Favorite" with in the media manager but does not select it.



Path: C:\Images for IMPACT

Name: images (3)
Type: jpg
Date: 2/24/2016
Time: 8:49 AM
Size: 275x183
Path: C:\Images for IMPACT

Name: images (4)
Type: jpg
Date: 2/24/2016
Time: 8:49 AM
Size: 261x193
Path: C:\Images for IMPACT

Name: images (5)
Type: jpg
Date: 2/24/2016
Time: 8:49 AM
Size: 264x191
Path: C:\Images for IMPACT

Name: Street
Type: jpg
Date: 2/24/2016
Time: 8:49 AM
Size: 600x424
Path: C:\Images for IMPACT

Created 11 minutes ago

Name: images
Type: jpg
Date: 2/24/2016
Time: 8:49 AM
Size: 600x305
Path: C:\Images for IMPACT

Zoom Level: 100%
Media List: Images
File #1 of 7

Figure 8. Sample Media Manager Tile

The Favorites fly-out menu (aka fly-out) labeled "Favorites" is located on the left hand side of the media manager tile. This fly-out displays three drop down lists of Media Tiles, one each for images, videos, and audio. Each list header identifies the type and number of Media Tiles in the corresponding list. Located at the top of the fly-out are buttons for toggling the display of the various media types. This allows for the display of only media types of interest to the user.

The Available Media fly-out labeled "Media" is located on the right hand side of the media manager tile. This fly-out displays four drop down lists of Media Tiles, one each for images, videos, audio, and new media files available that have become available. Located at the top of the fly-out are buttons for toggling the display of the various media types. This allows for the display of only media types of interest to the user.

The Extended Info fly-out labeled "Info" is located on the right hand side of the media manger tile under the Available Media fly-out and displays pertinent information about the currently selected media. This information includes but is not limited to: metadata, user notes, and any miscellaneous information. The information is displayed in appropriately labeled individual drop downs.

The Media Information Bar is located below the Main Media Display Panel and is divided into three sections. These sections display information pertinent to the currently selected media, including but not limited to: file location, creation time, status, geo-location of the media, time last modified, and media format specific information.

5.2.6.4 Vehicle Dashboard

The vehicle dashboard tile (Figure 9) provides a visual overview of the dynamic components of a specific vehicle. It is used as a window or view into the vehicle itself, much as if one were to be sitting in the seat of the vehicle. All the components on this view are to provide at-a-glance information to inform the user of all major events occurring in a vehicle. As mentioned, the look and feel of the dashboard were designed to resemble a vehicle popup heads up display (HUD).

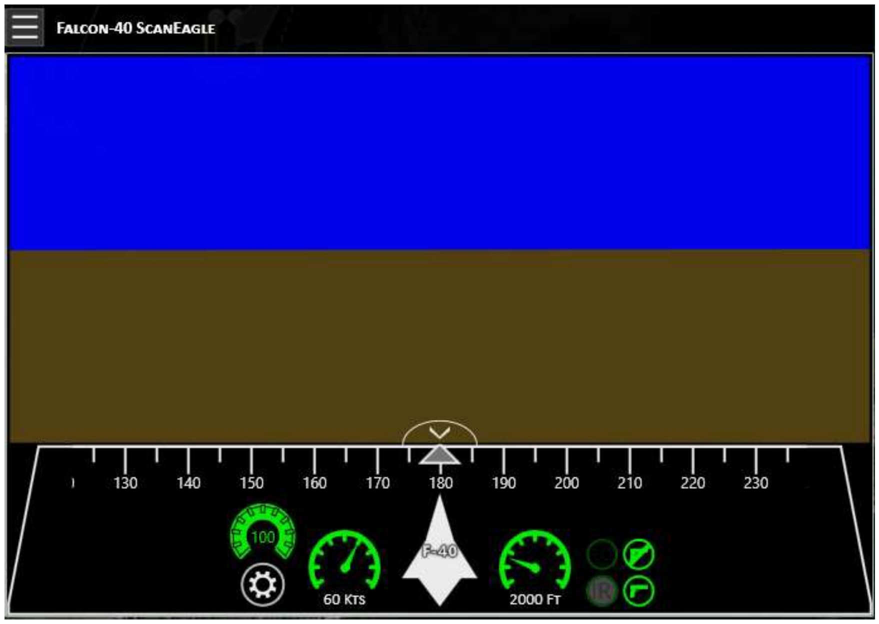


Figure 9. Sample Vehicle Dashboard Tile

The vehicle popup HUD provides the bulk of the high level information the user would need to make reactive decisions to the scenario on a per vehicle basis. At the core of this control is a Vehicle Presenter, enabling the user to see the specific type of vehicle being controlled, the amount of fuel left, the play that has been instantiated and the quality of that play. The notion of a play is analogous to a high level goal or task the user can assign to the system and depending on the services currently associated with a particular scenario, the system will allocate the necessary vehicles to accomplish the play. For each indicator on the HUD, color is used to encode the state of that subsystem (green, yellow and red). Green indicates normal state, yellow being degraded and red being severe. Accompanied with the vehicle presenter are two dials to show speed and altitude that follow the same indicator color system. At the far left of the vehicle presenter we have error indicators that show and hide depending on the severity of the specific system in the vehicle. These errors include items such as high fuel burn rate and malfunctioned battery. In between the vehicle presenter and the error indicators is a single button/indicator which allows the user to manually override the vehicle or then re-engage back to automation mode. This button provides several basic actions to control the vehicle. First is a toggle-able button to disengage/reengage automation mode of the vehicle and being in manual mode allows the user to select the other options. The first action available is Go To Waypoint, which allows the vehicle to immediately head towards a waypoint selected in the successive window. The second action available is Flight Directed, enabling the user to send the vehicle on a straight line path in its current heading indefinitely. The last action is the Loiter Now function which sets a loiter point at the current location. Lastly, on the right of the HUD are payload indicators showing which payloads the vehicle has as well the relative color coded status. The payloads consist of items such as lethal and nonlethal weapons, EO sensor or IR sensor. Finally, across the top of the HUD is the current heading of the vehicle. The next main feature of the dashboard is what lies underneath the popup HUD, the vehicle view.

The vehicle view is a toggle-able artificial horizon indicator and sensor video feed view. This view serves as the primary display / canvas. The default view is an artificial horizon indicator for the vehicle and on the top left of the tile is a toggle button to switch modes to a live feed of the active sensor. If the sensor has a gimbal, the sensor can then be steered via clicking on the video feed at the geo-location to focus on as well as zoom in to specific locations via mouse wheel scrolling. Next to the video toggle button is a center on command which is used to quickly find the vehicle, center on it and lock the camera to it as it moves. If the scenario supports the notion of goal-directed plays that define the actions of one or more vehicles, there is a color coded border around the edges of the vehicle view which directly correlates with the play color to easily see which dashboard is associated with what play.

5.2.6.5 Generic Cockpit Tiles

Since Fusion is intended to provide support to a broad range of autonomy research initiatives, the development of a generic cockpit to support baseline manned-unmanned teaming became a priority requirement. The Fusion development team created a series of tiles to represent a number of common flight instruments and control/display features required to support the virtual simulation of a 3rd or 4th generation aircraft. The development of these tiles provides the basic building blocks to easily construct a representative cockpit of almost any configuration, and with minor modification could be tailored to accurately represent any cockpit or flight deck. A version of the generic cockpit tiles being used for an ongoing program within AFRL is presented in Figure 10.

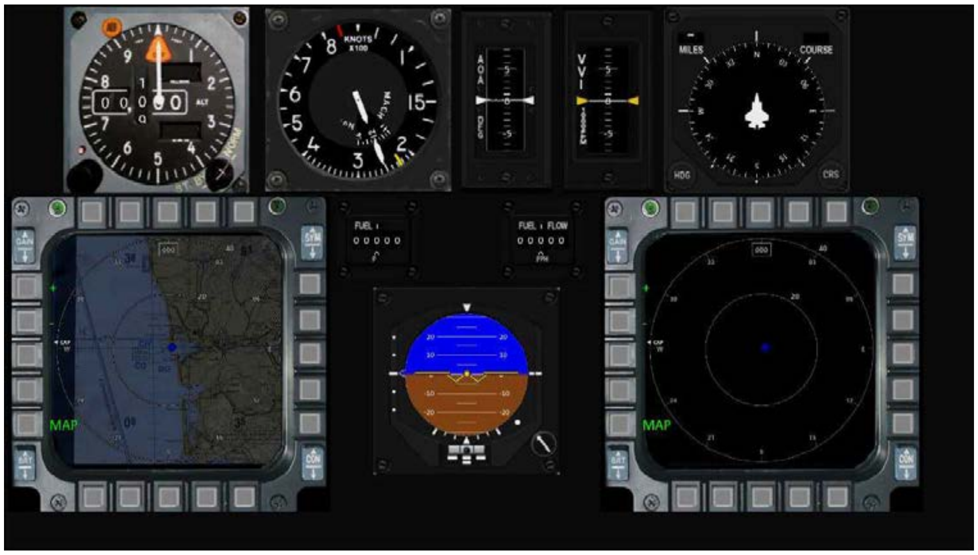


Figure 10. Generic Cockpit Instrument Tiles

5.3 Module / Messaging

Keeping with our philosophy of flexible interaction, Fusion can be adapted to interface to any number of messaging protocols and is therefore considered protocol agnostic. For example in one of the current Fusion projects, IMPACT (discussed in more detail in Section 4.1.1), all inter-machine communication is accomplished through DIS, LMCP, or the industry standard JSON formats (Figure 11). The DIS messages support simulation of entities, such as vehicles, people, weapons, and effects. The DIS messaging also drives the simulated sensor video feed and is rendered through SubrScene (see https://sourceforge.net/projects/subrscene).

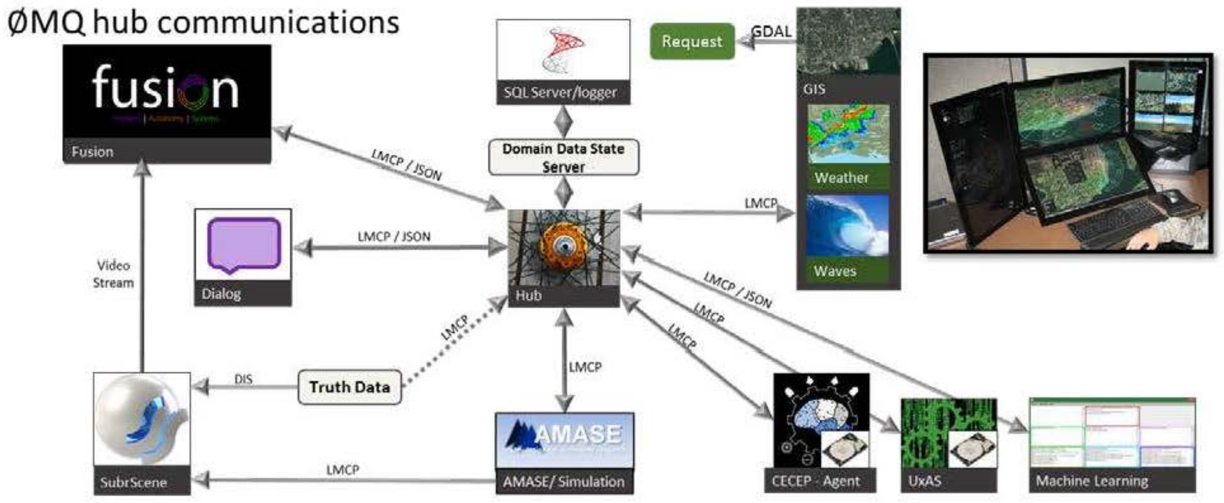


Figure 11. Example Cloud-based Network Architecture (IMPACT)

LMCP (see https://github.com/afrl-rq) is a protocol focused on vehicle control and simulation. Its message structures are defined through extensible markup language (XML) using its Message Data Model (MDM), which is employed to generate class libraries in several computer programming languages and documentation in html. The class libraries support serialization of the messages into binary format suitable for transfer over the network. This protocol is employed primarily by Fusion, AMASE (Aerospace Multi-Agent Simulation Environment; see https://github.com/afrl-rq/OpenAMASE/wiki/About-AMASE), and UxAS (Unmanned Systems Autonomy Services; see https://github.com/afrl-rq/OpenUxAS - (Rasmussen, Kingston, 2016)). AMASE defines vehicle dynamical models and propagates them through time to produce states using LMCP throughout its internal representations. UxAS employs LMCP primarily as its communication mechanism, collecting requests from Fusion and an intelligent agent service (Cognitively Enhanced Complex Event Processing (CECEP) – (Atahary, Douglass, Webber, 2015)) then generating plans to fulfill the requests, which are then sent on to AMASE. Fusion collects the plans generated by UxAS and the states from AMASE to illustrate current and expected activity to the operator.

The JSON format is employed by Fusion, CECEP, and the plan monitoring service to satisfy various communication needs. Fusion sends play specification information to CECEP, then proposes and invokes play solutions, and finally the plan monitoring defines play status during execution with all using JSON. Using JSON, we can create message structures in a much more concise manner and ease the interoperability between the various software applications. JSON does not specify how to serialize itself, and this is left up to the system developers.

This hybrid messaging structure has worked well to date, but there are a few ongoing issues that may justify alternative protocols:

1. The open representation supported by JSON message definitions has caused problems in the past where one software system is sending messages in a format that another software system does not expect. Casting the resulting message to a class can cause exceptions if the format is not agreed upon and rigorously enforced.
2. LMCP was designed to be very flexible in specifying messages and ease-of-use in a research setting, but may lack in performance for high-message scenarios where other serialization protocols are more appropriate. Its specification supports rigorous typing, but undisclosed MDM modifications can cause significant issues. Thus, the synchronization of the MDMs across ALL software components becomes a concern.
3. LMCP compiles to several popular programming languages, but unfortunately Javascript is not one of them. Conversely, JSON is a first class citizen in Javascript.

6.0 Fusion Development Approach

6.1 Organization

6.1.1 Fusion Team

The Fusion team is a small group of software developers that are a mix of government civilians and government contractors. The Fusion team consists of several key members that break down, plan, and execute requirements from many diverse research projects. These key roles are the project manager, product owner, scrum master (described in more detail in Section 6.3), and the developers. These roles and the team member performing them have evolved over time to best suit the needs of the team and the different projects. Due to having a smaller team, some team members often fulfill more than one role at a time. Usually all of these roles are filled by team members who are also developers on the project. The roles and responsibilities are:

- Project Manager
  - Builds the project team
  - Helps define the requirements
  - Defines the goals and the timeframes for them to be achieved
  - Handles the budget for the project
- Product Owner
  - Coordinates with the project manager, customers, and stakeholders
  - Determines product functionality and content
  - Decides when the product is ready for release
  - Manages and prioritizes the product backlog
  - Ultimate arbiter on requirement issues
- Scrum Master
  - Facilitates the software development process
  - Enforces full involvement in all meetings and roles
  - Communicates team progress and product status
  - Shields the team from external interference
- Developers
  - Self-organize and collaborate with the rest of the Fusion team to accomplish tasks defined for the sprint
  - Defines, as a group, which tasks will be worked for a given sprint cycle based on customer requirement priorities

6.1.2 Research Facilities

A key enabler of the research conducted under the Fusion effort was the development and sustainment of the research facilities. This work included the procurement, setup and installation of all development stations, simulation testbeds, specialized test equipment and the networks required to connect these systems within the Crew Systems Integration Laboratory (CSIL) (Figure 12).

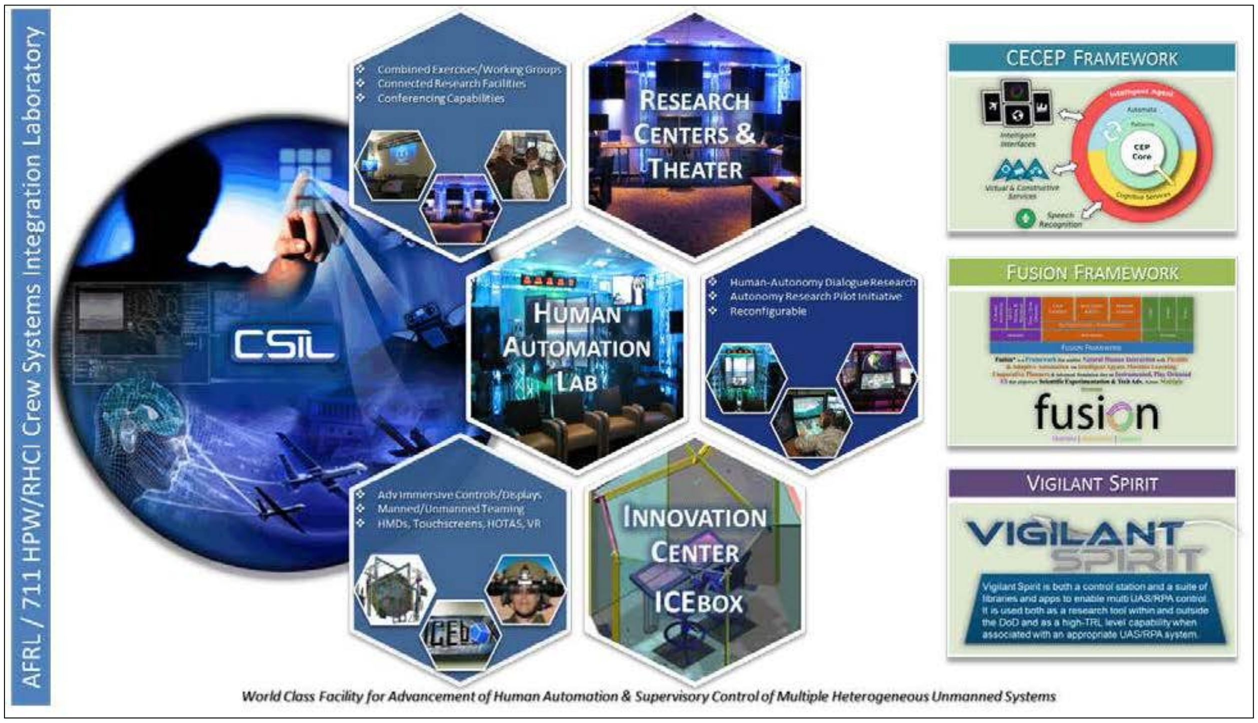

AFRL / 711 HPW/RHCI Crew Systems Integration Laboratory
CSIL
Combined Exercises/Working Groups
Connected Research Facilities
Conferencing Capabilities
Research Centers & Theater
Human Automation Lab
Human-Autonomy Dialogue Research
Autonomy Research Pilot Initiative
Reconfigurable
Adv Immersive Controls/Displays
Manned/Unmanned Teaming
HMDs, Touchscreens, HOTAS, VR
Innovation Center ICEBox
CECEP Framework
Fusion Framework
fusion
Vigilant Spirit
VIGILANT SPIRIT
Vigilant Spirit is both a control station and a suite of libraries and apps to enable multi UAS/RPA control. It is used both as a research tool within and outside the DoD and as a high-TRL level capability when associated with an appropriate UAS/RPA system.
World Class Facility for Advancement of Human Automation & Supervisory Control of Multiple Heterogeneous Unmanned Systems

Figure 12. CSIL Capabilities

6.2 Process – Virtual Distributed Laboratory

The notion of a virtual distributed laboratory (VDL) connecting various DoD and contractor sites throughout the CONUS was paramount to foster a more cohesive and distributed development and research environment. Fusion has adopted a DoD open source model, enabling joint development across a variety of projects and collaborators, all contributing to a single source repository. The core development team is located at Wright-Patterson AFB, and there are currently several offsite laboratory development teams (Figure 13). Fusion is hosted on a secure web server and program access can be requested at https://www.vdl.afrl.af.mil/ (please contact the authors for further instructions). The software is developed on a standard Windows 10 PC platform in Microsoft Visual Studio and several third party developer libraries (Figure 15). All distributed laboratory sites have similar hardware configurations and the software developers use a common set of software development and configuration management tools.

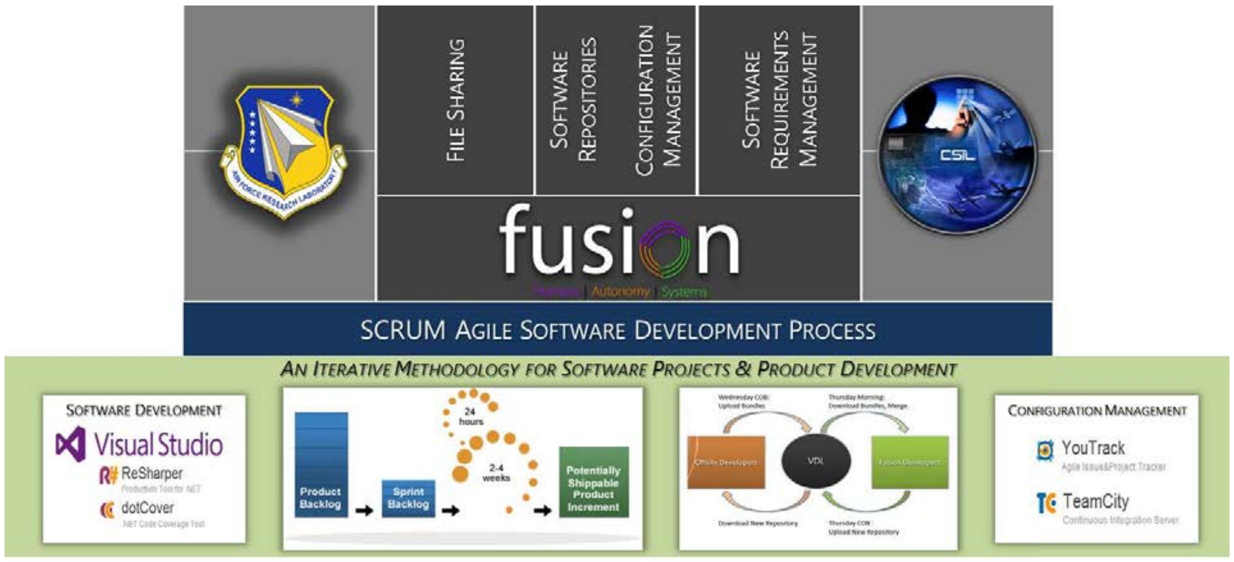

File Sharing
Software Repositories
Configuration Management
Software Requirements Management
fusion
Humans | Autonomy | Systems
CSIL
SCRUM Agile Software Development Process
An Iterative Methodology for Software Projects & Product Development
Software Development
Visual Studio
ReSharper
dotCover
Product Backlog
Sprint Backlog
24 hours
2-4 weeks
Potentially Shippable Product Increment
Offsite Developers
VDL
Fusion Developers
Configuration Management
YouTrack
Agile Issue&Project Tracker
TeamCity
Continuous Integration Server

Figure 13. Fusion Virtual Laboratory Concept.

6.3 Software Development – Agile

Fusion adheres to agile software development principles by utilizing an iterative and incremental software development framework known as Scrum (see https://www.scrum.org/). Scrum allows for the software development process to adapt, as needed, to changing work constraints and deadlines. While the durations and times specified may change, the following process will remain. Currently, the Fusion team will typically follow a 2-4 week sprint cycle depending upon the dynamics of the product backlog and the nature of emerging requirements (Figure 14).

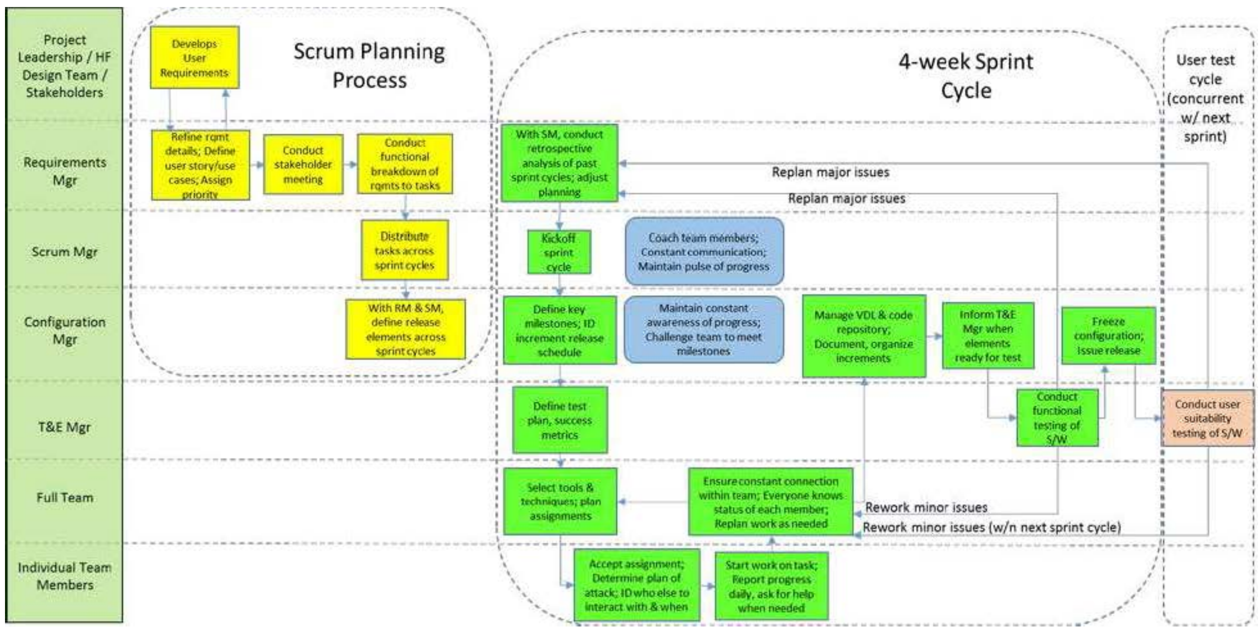

Project Leadership / HF Design Team / Stakeholders
Requirements Mgr
Scrum Mgr
Configuration Mgr
T&E Mgr
Full Team
Individual Team Members
Scrum Planning Process
4-week Sprint Cycle
User test cycle (concurrent w/ next sprint)
Develops User Requirements
Refine rqmt details; Define user story/use cases; Assign priority
Conduct stakeholder meeting
Conduct functional breakdown of rqmts to tasks
Distribute tasks across sprint cycles
With RM & SM, define release elements across sprint cycles
With SM, conduct retrospective analysis of past sprint cycles; adjust planning
Replan major issues
Replan major issues
Kickoff sprint cycle
Coach team members; Constant communication; Maintain pulse of progress
Define key milestones; ID increment release schedule
Maintain constant awareness of progress; Challenge team to meet milestones
Manage VDL & code repository; Document, organize increments
Inform T&E Mgr when elements ready for test
Freeze configuration; Issue release
Define test plan, success metrics
Conduct functional testing of S/W
Conduct user suitability testing of S/W
Select tools & techniques; plan assignments
Ensure constant connection within team; Everyone knows status of each member; Replan work as needed
Rework minor issues
Rework minor issues (w/n next sprint cycle)
Accept assignment; Determine plan of attack; ID who else to interact with & when
Start work on task; Report progress daily, ask for help when needed

Figure 14. Agile Software Development Process

From a development perspective, the Fusion team was similar to a scrum team, however the roles of the product owner and scrum master were shared between a government civilian and a government contractor representative.

Sprint Planning was held on the first day of every sprint cycle and was time-boxed to 4 hours. Attendees included the Fusion team and any subject matter experts (SMEs) as needed. However, SMEs attended primarily to answer questions or clarify misunderstandings as they arise from the Fusion Team. Standups were held at least once per week during every sprint cycle, time-boxed to 15 minutes, and only attended by the Fusion team. During a standup, attendees took turns sharing the following:

- What they accomplished prior to the previous standup
- What they plan to accomplish until the next standup
- What issues they may need help with until the next standup

Any and all problem solving, hypothesizing, or further inquiry discussions on issues or requirements shared by attendees occurred only after the standup meeting had concluded.

Change review board meetings were defined within the typical scrum process, but were added to help facilitate requirement expectations among the Fusion product owners and Fusion stakeholders. The change review board meetings were held 1 day before the end of a sprint cycle and time-boxed to 2 hours. Only unresolved change requests (CRs) submitted prior to the meeting by the Fusion team and stakeholders were typically addressed.

Backlog grooming meetings were held 1 day before the end of a sprint cycle and after the change review board meeting. Backlog grooming meetings were time-boxed to 3 hours and only attended by the Fusion team. A backlog grooming meeting was done when the time-box expires or enough product backlog items (PBI) had been groomed such that they would fill two sprint backlogs (two sprint cycles). Difficulty estimations were given to PBIs by Fusion developers and were determined by, but not exclusively: complexity, risk, time, and scope.

Sprint review meetings were held on the last day of the sprint cycle and time-boxed to 2 hours. Attendees included the Fusion team and any stakeholders who wished to attend. However, Fusion stakeholders attended primarily to observe and inquire for further clarification to what the Fusion team presents.

6.4 Configuration Management

The Fusion source code repository follows a strict configuration management process. Once a week, offsite developers submit their changes, and the core Fusion team integrates those changes and posts a new version of Fusion on VDL for the offsite developers and researchers. In the near future, the team will transition to a fully on-line software development cycle utilizing Git (a software configuration repository structure) and the secure Defense Research & Engineering Network (DREN). This process allows all offsite labs to keep up to date with the core Fusion team as well as keep their software well maintained. The concept of a virtual distributed laboratory has been successful due to Fusion providing a robust and flexible software architecture. The Fusion development team uses YouTrack during the scrum process to track the product and sprint backlogs, and TeamCity to create specific software builds (Figure 15).

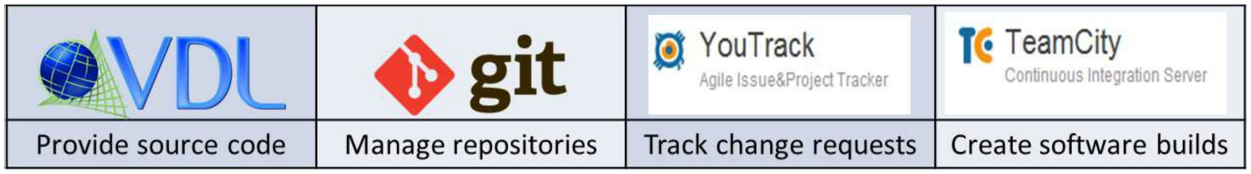


Figure 15. Fusion Configuration Management Tools

7.0 Lessons Learned

7.1 Software Development Processes in a Research Environment

Over the course of the project, several variations of software development processes have been explored. It is rather difficult to find a good process that works well in a research environment due to rapidly changing requirements. The type of software development process to utilize took trial and error before finding something that worked well for the team.

Agile processes sound great academically but can be challenging to practice as they tend to work best when requirements and priorities are well defined and can be planned for well ahead of time. The requirements can then be broken down into smaller pieces and planned out over one or more cycles. Often exact requirements are not known in a research environment or they change within a development cycle so it is hard to stay on task during the cycle. Developers get side tracked with new requirements that come up during a cycle and eventually tasks are forgotten or pushed out. The process is supposed to be agile and adjust to these situations but that isn't as easy as it sounds.

Software development cycle time is an important factor. Is the software development broken into smaller time spans or just treated as one giant timespan over the course of the project? Breaking it into some type of timeframe allows the developers to regroup at specific times, allows for requirements to change or be better defined without effecting the developers to often, and allows the developers to tackle issues one at time without being overwhelmed. We have tried large cycles that span months and small one week cycles. The shorter the cycle the more meetings and cycle management required, the longer the cycle the larger the requirement list is going to be.

Another issue to consider is team size and arrangement. Are all developers developing within a common framework treated as one giant team, are they assigned particular projects, or are tasks assigned based on skill sets? Large teams equal more people to work on more requirements but might be switching between types of requirements often, whereas smaller teams allow developers to focus on a smaller number of tasks or task types.

We have tried many different combinations of software development processes to follow and have found that you cannot strictly follow any given software development process within a research environment. The solution we have found to work the best is to break the development team down into smaller more focused teams that work on a single project or smaller subset of goals. Each team can decide and use the software development process that works well for them. Each small team also has the freedom to change the software development process if it isn't working. Sometimes teams will need to change their process for only a short period time to accommodate deadlines or particular tasks, but the important thing to remember is that they have the flexibility to make that decision. Most of the software sub teams use a variation of Scrum with 2-3 weeks cycles, then switch to a more queued task Kanban (see https://en.wikipedia.org/wiki/Kanban) like approach when needed based on upcoming events.

We have also learned the most important part of any software development process is communication. As long as everyone from the developers to the project managers are aware of what is going on, what the tasks are, the priorities of the tasks, and which tasks are being working on, everything seems to run more smoothly.

7.2 Challenges of Creating Extensible Software

One of the main goals and reasons for creating Fusion was to create an extensible software framework. The idea behind creating an extensible framework for user interface research was so that user interface elements as well as the data models can easily be changed to support a variety of different projects.

Developing extensible software comes with challenges. There is a delicate balance between making it too extensible and not extensible enough. It can be rather difficult at design time to decide whether or not that element should be extensible. Over the life of Fusion there have been elements they were made extensible that have never been extended so they are often viewed as over complicated. There are also elements that would have made tasks much easier if they had been written to be extendable.

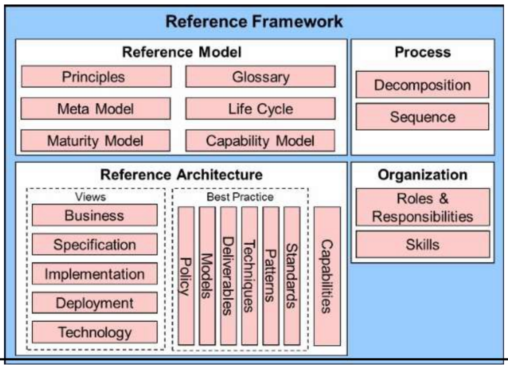
The core Fusion framework is fairly extensible but should projects that extend from Fusion also be extensible? When writing modules for new projects that extend from Fusion the developers don't always think that one day their project will be important enough that others may want to extend it. Sections of code are often refactored to make them more useable and extensible years after they were originally created and the team has acquired additional insight into how others will use them.

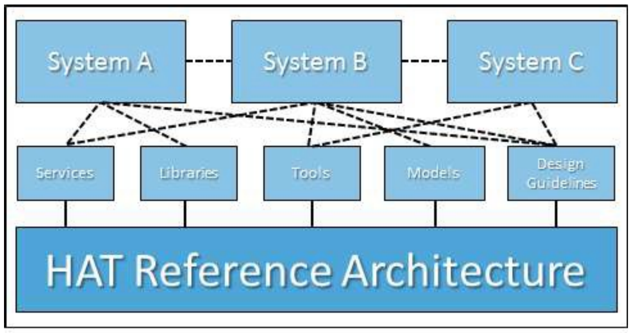
8.0 Future

Currently there does not exist a comprehensive Reference Framework that supports applying and extending RHCI's existing suite of software testbeds to provide symbiotic human machine interfaces for command and control of unmanned systems that harness potential benefits of autonomy across multiple domains (Air, Ground, Surface, Cyber, Space). Within RHCI there exists several testbeds and software services/tools to explore a rich set of research domains, each having its own unique characteristics and capabilities. As these systems mature over time, not having a common Reference Framework creates barriers for transiting concepts from one testbed/service/tool to another, both within and outside our branch. In an effort to overcome this barrier, the development of a common Reference Architecture and ontology will aid in the smooth transition of concepts across disparate testbeds, reduce software development costs/effort, and promote an enterprise approach to modeling, simulation, and analysis (MS&A) across AFRL (and potentially beyond).

Over the next several years our goal is to create a comprehensive Reference Framework / Architecture (Figure 16) that will provide a template/schema to define a Human Autonomy Teaming Testbed Suite (HATTS). This Reference Framework will include the following components …

1. Reference Model - Defines mission statement, principles, and ontologies/vocabulary.
2. Process – Define the processes used in developing this capability
3. Organization – Define roles and responsibilities and skillset needed
4. Human Autonomy teaming (HAT) Reference Architecture – Contains the Application Programming Interfaces (APIs), Interface Control Documents (ICDs), Development Standards, Schemas/Patterns.





The HAT Reference Architecture provides a fundamental software foundation by which components (such as services, libraries, tools, models, design guidelines, etc.) can be created to be compatible and extensible with each other to formulate a Systems of Systems Modeling and Simulation capability. This ensures that System A can effectively communicate with System B and so forth.

Figure 16. Future Human Autonomy Teaming Reference Framework / Architecture

This comprehensive reference framework will ultimately promote interoperability, standardized practices/interfaces, consistency across the tool suite, collaboration/communication/sharing across projects/organizations, and mobility of developers across our organization and beyond.

9.0 Acknowledgements

The authors would like to acknowledge the entire Fusion, IMPACT and ATACM teams for their contributions to the development and application of the Fusion Framework. It is only through the hard work of implementing the framework within the context of specific research programs attempting to explore real-world challenges that Fusion can continue to serve as a viable human-autonomy testbed. The authors would like to specifically acknowledge the contributions of those developers and analysts that were directly involved in the development of the Fusion Framework including Austin Bangert, George Bearden, James Boyer, Adam Buchanan, Bruce Clay, Michael Holland, Daylond Hooper, Michael Howard, Prothima Kollegal, Jerry Van Pelt, and Kenneth Wickline.

The authors would also like to acknowledge the software developed at AFRL for the Vigilant Spirit Program (Rowe, et al., 2009) as software engineering principles and concepts have transitioned into the Fusion Framework.

10.0 References

Atahary, T., Taha, T., Douglass, S. A., & Webber F. (2015). Knowledge mining for cognitive agents through path based forward checking. In Proceedings of the 16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2015), Takamatsu, Japan.

Calhoun, G. L., Ruff, H. A., Behymer, K. J., & Frost, E. M. (201a). Human-autonomy teaming interface design considerations for multi-unmanned vehicle control. Theoretical Issues in Ergonomics Science, 19(3), 321-352.

Ernest, N., Carroll, D., Schumacher, C., Clark, Cohen, K. & Lee,G. (2016). Genetic fuzzy based artificial intelligence for unmanned combat aerial vehicle control in simulated air combat missions. Journal of Defense Management, 6(144), 2167-0374.

Kingston, D. B., Rasmussen, S.J., & Mears, M. J. (2009). Base defense using a task assignment framework. AIAA Guidance, Navigation, and Control Conference, AIAA-2009-6209

Martin, R. C., Martin, Micah (2012). Agile Principles, Patterns, and Practices in C#.

Martinage, R. (2014), Toward a New Offset Strategy: Exploiting U.S. Long-Term Advantages to Restore U.S. Global Power Projection Capability. Center for Strategic and Budgetary Assessments, Washington D.C.

Rasmussen, S., Kalyanam, K., & Kingston, D. (2016). Field experiment of a fully autonomous multiple UAV/UGS intruder detection and monitoring system. IEEE International Conference on Unmanned Aircraft Systems (ICUAS), 1293-1302.

Rowe, A. J., Liggett, K.K., & Davis, J.E. (2009). Vigilant Spirit Control Station: a research testbed for multi-UAS supervisory control interfaces. Proceedings of the 15th International Symposium on Aviation Psychology.

Stevens, C., Spriggs, S., Dukes, A., Myers, C., Behymer, K., Ruff, H., Morris, M., Credlebaugh, C., Blackford, E., & Fisher, C., System for Workload Assessment and Monitoring for Predicting Effective Decision-making (SWAMPED): Year 1 (in press). Air Force Research Laboratory Technical Report.

Vernacsics, P. & Lange, D. (2013). Using autonomics to exercise command and control networks in degraded environments. 18th International Command and Control Research and Technology Symposium, Alexandria, VA. DTIC ADA 587015.

11.0 List of Symbols, Abbreviations, and Acronyms

3D – Three Dimensional
A2AD - Anti-Access Area Denial
AFRL - Air Force Research Laboratory
API - Application Programming Interface
AMASE - Aerospace Multi-Agent Simulation Environment
ARPI - Autonomy Research Pilot Initiatives
ASD R&E - Assistant Secretary of Defense Research and Engineering
ATACM - Autonomy for Air Combat Missions
CECEP - Cognitively Enhanced Complex Event Processing
CR - Change Request
CSIL - Crew Systems Integration Laboratory
DIS - Distributed Interactive Simulation
DoD - Department of Defense
DREN - Defense Research & Engineering Network
HATTS - Human Autonomy Teaming Testbed Suite
HMI - Human-Machine Interfaces
ICD - Interface Control Documents
IMPACT - Intelligent Multi-UxV Planner with Adaptive Collaborative/Control Technologies" (IMPACT)
IRC - Internet Relay Chat
JSON - Javascript Object Notation (JSON)
LMCP - Lightweight Message Construction Protocol
MDM - Message Data Model
MS&A - Modeling, Simulation, and Analysis
OCA - Offensive Counter-Air
OSG – Open Scene Graph
PBI - Product Backlog Item
PVI - Pilot Vehicle Interface
SA - Situation Awareness
SME - Subject Matter Expert
SWAMPED - System for Workload Assessment and Monitoring for Predicting Effective Decision-making
TCP/IP - Transmission Control Protocol/Internet Protocol
TBM - Tactical Battle Manager
UDP - User Datagram Protocol
UxAS - Unmanned Systems Autonomy Services
VDL - Virtual Distributed Laboratory
XML - eXtensible Markup Language
ZeroMQ – Zero Message Queue