Cloud-Based Distributed Control of Unmanned Systems

Kim B. Nguyen*[a], Darren N. Powell, Charles Yetman, Michael August, Susan L. Alderson, Christopher J. Raney
Space and Naval Warfare Systems Center Pacific
53560 Hull Street, San Diego, CA 92152

ABSTRACT

Enabling warfighters to efficiently and safely execute dangerous missions, unmanned systems have been an increasingly valuable component in modern warfare. The evolving use of unmanned systems leads to vast amounts of data collected from sensors placed on the remote vehicles. As a result, many command and control (C2) systems have been developed to provide the necessary tools to perform one of the following functions: controlling the unmanned vehicle or analyzing and processing the sensory data from unmanned vehicles. These C2 systems are often disparate from one another, limiting the ability to optimally distribute data among different users. The Space and Naval Warfare Systems Center Pacific (SSC Pacific) seeks to address this technology gap through the *UxV to the Cloud via Widgets* project. The overarching intent of this three year effort is to provide three major capabilities: 1) unmanned vehicle control using an open service oriented architecture; 2) data distribution utilizing cloud technologies; 3) a collection of web-based tools enabling analysts to better view and process data. This paper focuses on how the *UxV to the Cloud via Widgets* system is designed and implemented by leveraging the following technologies: Data Distribution Service (DDS), Accumulo, Hadoop, and Ozone Widget Framework (OWF).

Keywords: DDS, Accumulo, Hadoop, OWF, cloud, unmanned, widgets

1. INTRODUCTION

The use of unmanned systems in military applications is increasingly becoming more prevalent. Unmanned systems enable the warfighter to be removed from dangerous environments, simplify missions by reducing the number of humans required to accomplish a task, and provide surveillance and sensory data collection from physically demanding environments. The data collected is in high demand, however current unmanned system control software lacks the infrastructure needed to distribute and connect this data to the appropriate consumers. This limitation prevents consumers from discovering patterns, and formulating and sharing new findings efficiently.

The operator today has limited situational awareness with regard to the battlespace, and acts primarily as a creator of the data rather than a consumer. The data transmitted from unmanned systems is used tactically to help an operator perform the mission, but is often lost after it is displayed to the operator during mission execution. At best, the data is saved onto hard-drives and is accessible only by the local team. Data history in a form available and useful to the operator is limited to a local hardware instance, rather than distributed for use during vehicle hand over to another operator.

The analysts and strategic decision makers are thus restricted in their ability to view the data collected from the unmanned system, and generally receive the data only after it has been processed to georeferenced objects of interests. Available resources, often times, are allocated to support the visualization of live data streams (i.e. video), but not on the storage and distribution mechanisms for the collected data.

SSC Pacific's *UxV to the Cloud via Widgets* demonstrates the distribution and visualization of data collected by unmanned systems within a common presentation framework. The unmanned-vehicle-control software is built as a service-oriented architecture intended for transparency, exposing and making data available. While the unmanned systems are tactically controlled, the raw data collected by those unmanned systems are simultaneously sent to and stored in a cloud infrastructure. Cloud technologies enable the collected data to be stored long term, shared among multiple users, distributed among geographically separate locations, and easily retrieved at a later time as required. This system utilizes the Ozone Widget Framework (OWF) as the common display interface to present the tactical controls and tools to analyze and view the collected data to both operator and analyst users.

*kim.b.nguyen1@navy.mil; phone 1 619 553-9456

The three main components include unmanned vehicle control, cloud infrastructure, and the web browser presentation layer. This paper provides a high level description of how each component was designed and implemented.

2. UNMANNED VEHICLE CONTROL

The Remote Operator Control Services (ROCS) unmanned-vehicle-control software is designed as a service-oriented architecture to provide operators the tools and framework required when tactically controlling unmanned systems. In addition to vehicle control, this same component provides the mechanisms needed to share and distribute raw data collected from unmanned vehicles among users, both local and geographically separated. The following subsections describe the design and implementation behind this component at a high level.

2.1 Service-Oriented Architecture (SOA)

ROCS follows a SOA design pattern where each capability of the system is designed and implemented as an independently run service. Together these services compose the unmanned-vehicle-control component in the *UxV to the Cloud via Widgets* system. These services collaborate with one another by communicating with a common databus to retrieve and share vehicle-status updates and commands. These services enable operators to control multiple, heterogeneous unmanned systems, as well as share the data received from the unmanned systems to the cloud for other users to access.

Below is a list of key SOA benefits.

1. Service Reuse: Capabilities common to all unmanned vehicle protocols, such as operator hand-off and data ingestion to the cloud, are abstracted into separate services. This approach allows the system to be configured to reuse those same services for multiple applications (i.e. controlling both unmanned surface vehicles (USV) and unmanned aerial vehicles (UAV)).
2. Extensibility Ease: SOA emphasizes that each software component be designed to be loosely coupled. All services are self-contained and run independent from one another, with each service implemented to fulfill a single function or capability. With loosely coupled services, the system can easily be configured to restrict and/or extend other features and capabilities. New features can be added without modifying the existing system by implementing them as isolated services that interface to the common databus.
3. Flexibility: Since each service in ROCS is designed to run independently, the system can be tailored and configured to run only the services required to provide the functionality needed by the operator. For example, if the operator only needs to control a USV, then ROCS can be configured to run only the required services to provide USV control.

2.2 Data Centric Model

ROCS is built around a data-centric model where a centralized databus stores and shares the current system state. DDS is chosen as the solution for the databus for its real-time data exchange, high performance and scalability capabilities. The databus is fully compliant with the Open Management Group (OMG) Data Distribution Service (DDS) standard. ROCS currently uses the Real-Time Innovations (RTI) Connext as the data centric middleware, or DDS, but has also used the PrismTech Opensplice DDS implementation in previous development.

As the ROCS middleware, DDS is solely responsible for storing and distributing the state of the unmanned vehicle at real time, but not how the data collected from the unmanned vehicle is stored long-term. The services in ROCS are designed to trust DDS to hold the current state of the entire system. This state is defined as the information needed to tactically control an unmanned system such as its current geographic location, which operator is controlling the vehicle, navigation mode, etc. The state of the system is stored and distributed by the DDS, and is trusted to be the most accurate and up-to-date. This assumption holds validity because the services are designed to offload and publish any collected data from the unmanned vehicles to the DDS. Any changes a service may make to the unmanned-vehicle data that affects the state of the system are also published to the DDS. With this approach of using DDS, the need to synchronize data between the different services is eliminated.

Unmanned vehicles tend to vary in sizes, purpose, and platform type, typically resulting in an array of “languages” required to communicate to the various vehicles. In order to design a system that supports multiple heterogeneous

unmanned vehicles in an efficient and extensible manner, a common data model that captures the general status, commands, and state for any unmanned system is required. General system capabilities, such as cloud ingestion, could be designed around the common data model and implemented once. If a new vehicle protocol is introduced, only a translation service that maps the vehicle protocol to the common data model needs to be implemented.

Figure 1 shows the system architecture for ROCS. All services retrieve data by subscribing to the DDS, and all modifications made to the data are published back to the DDS. The "protocol" services are designed to publish the live status from the unmanned vehicles and process the commands sent to the unmanned vehicles. ROCS currently supports the following protocols: Joint Architecture for Unmanned Systems (JAUS), AeroVironment Raven and Puma. Translation services transform data from the native protocols, e.g. JAUS, over radio communications often using User Datagram Protocol (UDP) to the common data model on the wired network using DDS. The common data model in ROCS was designed around the Mine Countermeasures (MCM) Data Model. This enables other services (such as the "Cloud Ingest" services) to only support the common data model while providing access to data from multiple vehicle protocols. With the state of the system stored on the DDS, any service that can subscribe and retrieve data from the DDS also has access to the current system state.

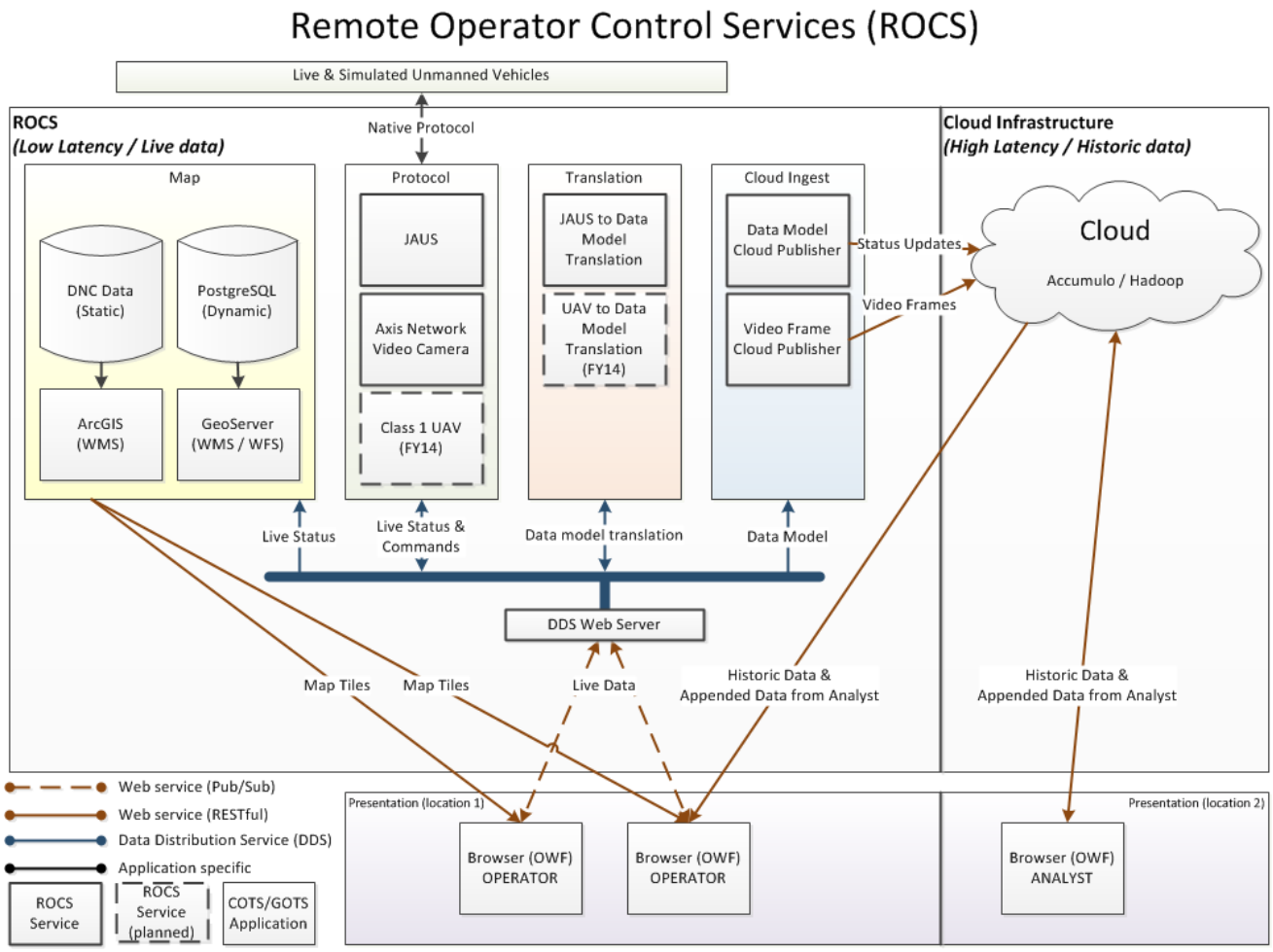

Figure 1: The system architecture of ROCS and how the vehicle-control software interfaces to the cloud infrastructure and presentation layer.

2.3 Real-Time Unmanned Vehicle Control within a Web Framework

When designing a system to control unmanned vehicles in real-time, there are three fundamental factors that need to be addressed: performance, asynchronous data support, and bi-directional communication between the operator and unmanned system.

1. Performance: The system needs to provide the operator with low-latency access to the unmanned-system data. The amount of time it takes to transport messages to and from unmanned systems needs to be minimal.
2. Asynchronous Data Support: Data collected by unmanned systems is asynchronous by nature. Unmanned systems typically contain several sensors, each of which can transmit data independent from one another, thus resulting in asynchronous data streams.
3. Bidirectional Communication: There are always two lines of communication between an operator and an unmanned system. An operator needs to be able to send commands to the unmanned system and also retrieve data from the unmanned system.

The Web, however, is built around a request/response communication model. When a client visits a web page, "an HTTP request is sent to the web server that hosts the page. The web server acknowledges the request and sends back the response[1]." This framework works well for data that is not dynamically changing or time sensitive. In the case of unmanned-vehicle control, where the data constantly changes and the interaction with the data needs to be in real-time, the traditional HTTP request/response model fails to provide adequate performance.

There are existing web technologies such as Asynchronous JavaScript and XML (AJAX) that mimic asynchronous real-time communication between a client and server. However, these technologies are built as workarounds for the HTTP-client-request-and-response model. Since every HTTP request requires headers and cookie data, a lot of extraneous information gets transferred between the client and server, increasing memory usage and network bandwidth. As a result, these web technologies often yield high latency when used to stream real-time content, making it an inadequate communication mechanism for unmanned-vehicle control. To minimize memory usage and latency for ROCS, WebSockets is chosen as the underlying web technology for transmitting asynchronous data between a client and server in real-time.

WebSockets is the "emerging protocol [that] has the potential to address many challenges with implementing asynchronous communication over the Web[2]". It provides "socket-like communication mechanisms into the Web[2]", enabling low-latency and bi-directional communication between client and server. WebSockets work by first establishing a handshake between the client and server over HTTP. Once the connection is made, WebSockets uses the Transmission Control Protocol (TCP) to transmit messages between the client and server and any overhead associated with HTTP connections is eliminated. This makes WebSockets a favorable framework for low-latency asynchronous bi-directional data transmissions over a long-lived connection between a client and server.

A subset of the services in ROCS are dedicated to provide low-latency real-time data transmissions between an operator and the unmanned system. The type of data the system needs to support include asynchronous data streams from the vehicle, commands from the operator, and multiple video feeds. These services all utilize WebSockets as the underlying data-transmission protocol between the client and server.

The NettyDDS service in ROCS enables real-time data transmissions of asynchronous data streams from the vehicle to the operator, as well as commands from the operator to the vehicle. This service resides on the server side of the system. NettyDDS provides publish/subscribe access to DDS, from a web browser client. It is designed as a Socket.IO server that by default uses the WebSockets transport protocol when possible. NettyDDS listens on a specific port (7171 by default) and opens a Socket.IO connection to any client that makes a request. Once the connection is established, the client then has complete publish-subscribe access to the DDS. This feature enables a client to send commands and receive live status updates to and from the unmanned vehicle within a web framework.

2.4 Video within a Web Framework

In order for unmanned systems to fulfill their operational purpose in the Intelligence, Surveillance, and Reconnaissance (ISR) military community, full-motion video is often a common sensor-payload capability. Full motion video enables

basic situational awareness for the operator vehicle's navigation and mission-specific surveillance. It can be the largest bandwidth consumer, typically resulting in massive amounts of required storage when archiving as data.

Similar to how unmanned systems communicate using a variety of protocols to control the platform and receive status, there are multiple types of compression and transports for a video stream. A general-purpose decoding library enables a system to support a wide variety of the most common compression schemes. As the common standard in this arena is FFmpeg, many open source video transcoding solutions are built using FFmpeg's underlying libraries. ROCS uses GStreamer which provides a framework to retrieve a video stream, transcode into a new stream, and serve to a new port.

The initial system design included the use of the HTML5 video tag, however this functionality was found to be very unreliable with a considerable amount of added latency. The concept of operations for ROCS is to be on a wired network, where there is a hard-wire connection between the services, including the client. This assumption allows the primary design constraint of HTML5 and video latency to be addressed by transcoding the video into Motion JPEG (MJPEG), resulting in a fast spatial compression but forgoes the latency-adding temporal compression. The video is brought into GStreamer, transcoded into MJPEG, and served to the browser over HTTP using a multipart/x-mixed-replace content type. Initially, the stream was delivered by long polling (leaving the connection open and continuing to send data), but due to the nature of how this data stream worked, it was impossible to tell if the video had stopped or failed for any reason. Technological advances with web browsers have since made this method obsolete, and a much more stable option was developed using WebSockets. By passing the MJPEG frames into the browser using binary WebSockets, the data can be continuously updated in a standard image tag that creates the video stream. In addition to a standard 2D display of video, user interfaces were also generated to display stitched 360 degree video within a WebGL canvas.

DDS was not used to transmit video frames, but the representation of the available GStreamer URLs were published to enable the client to gain knowledge of the available video feeds. These URLs were also used by the Cloud ingest service to push frames into the cloud for reconstruction into video for the Analyst view of historical data, as well as historical cross referencing of video frames to vehicle location for all users.

2.5 Mapping within a Web Framework

Most existing systems made to control and operate unmanned vehicles that are aware of their geographical location, provide some type of map interface for the user. A map is critical for both operator and analyst users. It enables the unmanned vehicle's mission progress to be easily tracked and visualized, thus enhancing the user's situational awareness. Often times, these maps become very resource intensive due to the large amounts of data they need to be able to handle and display.

To minimize the amount of geospatial data that is loaded into the web browser, the map application is architected to utilize a back-end Open Geospatial Consortium (OGC) compliant database and server to handle the processing and rendering. The following open source technologies: GeoServer, OpenLayers, PostgreSQL, and PostGIS are chosen to implement the back-end database and server. A brief description of how each of the technologies is used is listed below.

1. GeoServer: A Java-based server that renders geospatial data stored in a database to various mapping formats following the Web Map Service (WMS) standard. This technology minimizes the resources that the web-based map application would otherwise require by off-loading the processing to the back-end server side. With this, the map application is capable of displaying large amounts of data in the web browser.
2. OpenLayers: A JavaScript-based mapping library that enables a web-based application to display map data. Capable of supporting multiple open standards, OpenLayers is very flexible and provides an array of capabilities needed to visualize and to edit the geospatial map data.
3. PostgreSQL: An SQL-compliant object-relational database that easily scales to accommodate large amounts of data - upwards to over 4 terabytes, and many concurrent users. Many extenders are written for PostgreSQL enabling it to support geospatial data making it an appropriate fit.
4. PostGIS: An extender that enables PostgreSQL to support spatial data. This allows the database to store OGC compliant geographic objects, enabling GeoServer to render the appropriate mapping formats automatically.

Data first comes off of the vehicles in its native protocol and is published onto the DDS. A translation service converts that data into the common data model, which is then published onto the DDS. Another service subscribes to the common data model and converts that data into a geospatial data type, which is then written to PostgreSQL database. The spatial

database extender, PostGIS, allows the PostgreSQL database to support geospatial data types. These geospatial data types can then be rendered into a Web Map Service (WMS) layer and served to a web page using GeoServer. Once the WMS tile is served, the web-map application uses the OpenLayers API to display it in the web browser.

3. CLOUD INFRASTRUCTURE

Unmanned vehicle control systems are typically not built with the infrastructure to store and archive data collected from unmanned-vehicle sensors. An unmanned vehicle can have multiple sensors onboard each collecting data at various frequencies. On the completion of a single mission a significant amount of data can be collected; combined with numerous missions and multiple unmanned vehicles the amount of collected data can grow rapidly and exponentially. The lack of infrastructure in supporting data storage and data distribution leads to two problems: 1) sensor data not archived cannot be analyzed for identification of conspicuous objects or events; 2) situational awareness is not maintained when one operator hands off control of an unmanned vehicle to another operator. In the interest of designing a system that is capable of storing and distributing the unmanned-vehicle collected data, cloud technologies (e.g. Accumulo) is chosen as the underlying data storage infrastructure.

3.1 Accumulo

The cloud infrastructure is implemented using Accumulo, which is a distributed key-value based data storage and retrieval software originally developed by the National Security Agency (NSA), but is now maintained by the Apache Software Foundation. Accumulo is based on Google's BigTable design, which is a "distributed storage system for managing structured data to scale to a very large size; petabytes of data across thousands of commodity servers[8]." Accumulo is well suited for applications that need to aggregate large amounts of data. Accumulo is built on the following Apache technologies: Hadoop, Zookeeper, and Thrift; a brief description of the technologies is listed below.

1. Hadoop: A framework that enables large data sets to be stored and processed in a distributed manner across a cluster of computers. Hadoop consists of two primary subsystems: the Hadoop Distributed File System (HDFS) and Hadoop MapReduce. HDFS consists of multiple nodes that work together to produce a reliable, high-throughput data storage and processing system while MapReduce is the computational model that processes large amounts of independent data in parallel.
2. Zookeeper: A single centralized service that aims to coordinate and maintain the different applications of a system in a distributed manner. It tracks the configuration information, naming, and synchronization.
3. Hadoop: A software framework that provides a common definition for data types and service interfaces across various programming languages such as C++, Java, Python, Node.js, etc.

The motivating factors in selecting Accumulo as the cloud infrastructure are Accumulo's support for security, performance, and flexibility. Security is always a major concern within military applications. Data is marked with a specific classification (e.g. UNCLASSIFIED, DOD, FOUO, LAW ENFORCEMENT), which determines whether or not a consumer has the correct need-to-know basis to access the data. By design, Accumulo implements its own security mechanism giving it the advantage over other data-storage frameworks where security would have to be built on top of the data-storage software. Without affecting any access and retrieval performance of the data, Accumulo applies security at the cell-level, meaning each key-value pair that represents the data contains a security attribute. Accumulo is proven to perform with very fast reads and writes, "10,000s operations per second per node[9]", and is easily scalable to support large amounts of data (petabytes). The non-relational, no-SQL key-value structure makes Accumulo very flexible. Without the need to define a data model, Accumulo can be extended to store sparse, disparate, heterogeneous data easily, making it well-suited for collected unmanned-vehicle data where the type of data collected is continually evolving.

3.2 Unmanned-Vehicle Status Data Representation

Accumulo is a non-relational data store that does not require a data model to be defined. The data in Accumulo are represented as separate key-value entries where each key is encompasses the following five elements: row id, column family, column qualifier, column visibility, and timestamp. Accumulo is able to efficiently sort and categorize the key-value pairs by using any of those five elements. With this key-value pair model, the status information collected from the

unmanned vehicle is stored within the column family and column qualifier elements of the key. By structuring the key-value pairs this way, the unmanned-vehicle data is able to be quickly sorted and retrieved. Figure 2 shows an example of how the unmanned-vehicle status data is mapped into the key attribute for the key-value pair. Starting at the third row, each proceeding row represents the "key" for a single key-value pair stored in Accumulo.

Vehicle ID: USV1
Latitude: 36.478
Longitude: 25.340
Timestamp: 1349121798000

Vehicle ID: USV2
Latitude: 40.478
Longitude: 15.340
Timestamp: 1349121799000

Key				
Row ID	**Column Family**	**Column Qualifier**	**Column Visibility**	**Timestamp**
USV1	Vehicle ID	USV1	Unclassified	1349121798000
USV1	Latitude	36.478	Unclassified	1349121798000
USV1	Longitude	25.340	Unclassified	1349121798000
USV2	Vehicle ID	USV2	Unclassified	1349121798000
USV2	Latitude	40.478	Unclassified	1349121799000
USV2	Longitude	15.340	Unclassified	1349121799000

Figure 2: A diagram that depicts how incoming unmanned-vehicle status data get mapped into the Accumulo key-value pair.

3.3 Media Storage

A separate media server, utilizing OpenStack Swift as the underlying storage system, is used to store all "media" data, which is defined as image frames and video clips. Once an image frame is received, a URL that represents the location of where the image is stored is generated. A set of key-value pairs that represent the meta-data associated to that image frame along with the URL of where the image is stored on the media server are inserted into Accumulo. To demonstrate the processing capabilities of cloud technologies, *UxV to the Cloud via Widgets* contains a mechanism where raw images from the unmanned-vehicle cameras are processed and rendered into a series of video clips, known as historical video. This enables an analyst to watch a playback of the unmanned vehicle's mission. Figure 3 below shows the process of how video clips are rendered from raw images from the unmanned-vehicle cameras and stored into the media server and figure 4 shows the process that enables users to retrieve and watch these historical video clips from the unmanned vehicles.

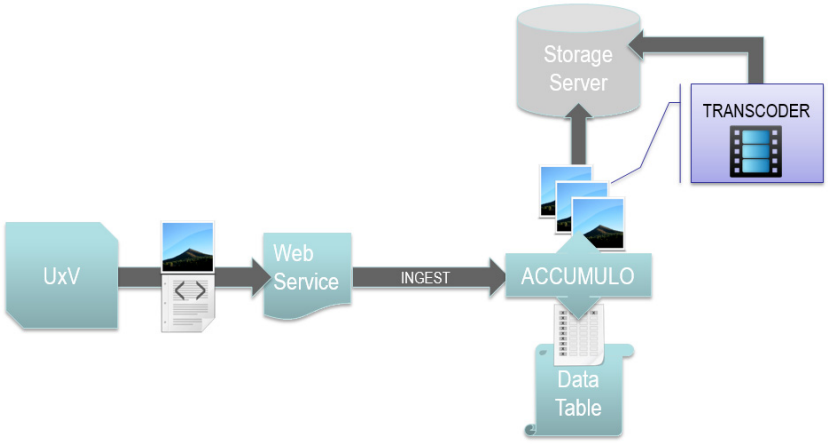

Figure 3: A diagram that shows the process of how image frames are stored and rendered into video clips.

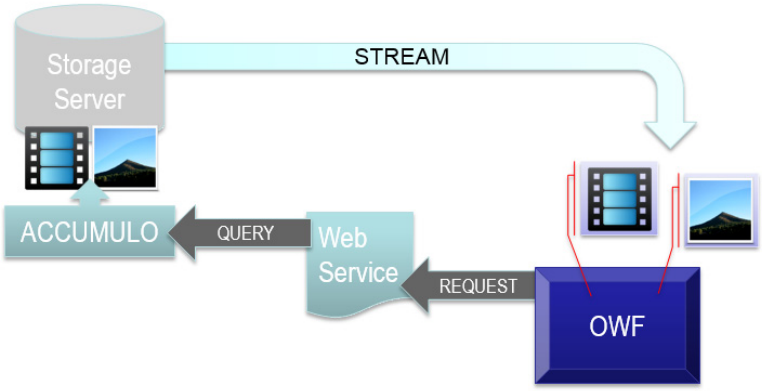

Figure 4: A diagram that shows the process of how image frames and video clips are retrieved from Accumulo and the media storage server

3.4 Limitations with Cloud Technologies

Military systems can be required to operate in a closed private network where there is no external connection or the network communication is in a disconnected, intermittent, low-bandwidth (DIL) environment. This constraint then requires each system to have its own cloud infrastructure, which results in a collection of disparate cloud stacks across multiple systems. Cloud technologies provide a solution in distributing data and processing power across multiple computers operating, however this is only applicable within a single cloud instance. What happens when data needs to be federated across multiple, disparate cloud instances?

4. PRESENTATION LAYER

A unified presentation layer is the key in pulling together the components that provide vehicle control, data visualization, and distribution as a single software system for the user. In the interest of provisioning a non-proprietary open-source lightweight architecture, Ozone Widget Framework (OWF) is chosen as a common computing environment. OWF provides a uniform look and feel, standardizing the user experience for all web-applications across disparate functional areas (e.g., analysis and control). These web-applications that run within OWF are classified as widgets, which are technology agnostic and can be designed and developed in myriad of programming languages. The only requirements to run OWF are a compatible web browser and a network connection to the back-end services. This approach eliminates the need for a user to install, configure, and maintain systems associated with thick-client applications.

4.1 Ozone Widget Framework (OWF)

OWF is a Government Open Source Software (GOSS) product that was initially developed for and by the National Security Agency (NSA). Currently OWF has achieved widespread adoption and utilization throughout the Department of Defense (DoD). In its essence, OWF provides a web-based framework that enables distributed users to consume, visualize, and interact with data and other back-end services via a web browser. Custom-made widgets can be easily built using common web technologies. Widgets are designed to be portable, lightweight, single-purpose HTML-based web applications that are launched within OWF. By using OWF, groups of widgets can be configured to launch together giving the user the “look and feel” of a desktop application within a web browser.

4.2 Client-Server Authentication

Security is a major concern for all military applications. As part of *UxV to the Cloud via Widgets*, an end-to-end, user-to-widget-to-cloud authentication and authorization mechanism was developed within OWF. The resulting security architecture leverages multiple server-side components, to include OWF, Central Authentication Service (CAS), OpenLDAP, a custom RESTful API and Accumulo. Figure 5 below depicts the authentication and authorization process from the top-down.

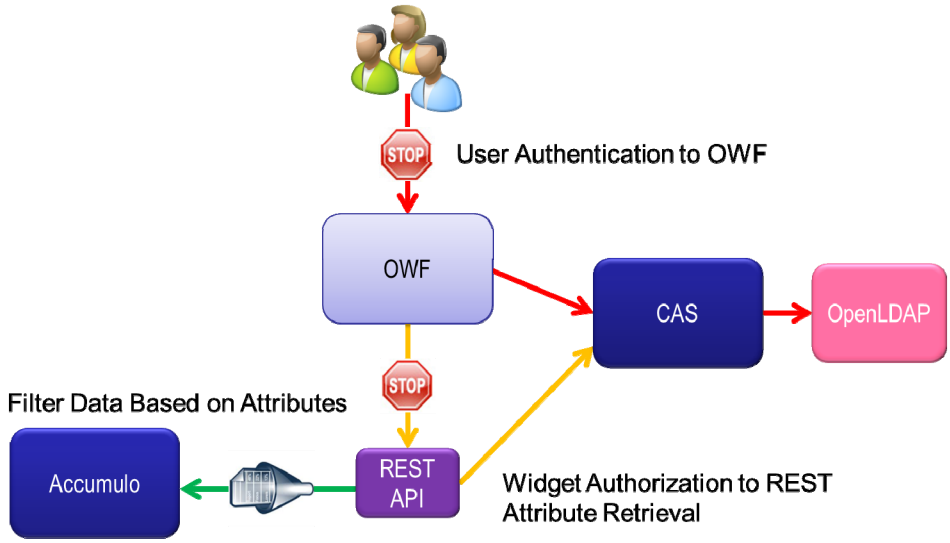

Figure 5: A diagram that depicts the process of how a user is authenticated and authorized in OWF when logging in.

A user browses to the OWF web URL, and in turn OWF redirects this request to the CAS service, prompting the user to enter their pin associated with a X.509 PKI certificate embedded on a Common Access Card (CAC). If a successful pin is entered, this request is sent to a Defense Information Systems Agency (DISA) provided Online Certificate Status Protocol (OCSP) responder, which performs a Certificate Revocation List (CRL) check. At this point, if the user presents a valid (i.e. not revoked) PKI certificate and enters the corresponding pin, the user is successfully authenticated (meaning the user is who he/she claims to be) to OWF.

CAS then forwards the user's PKI information to OpenLDAP to determine if the user belongs to the user store. Once verified, the user's attributes are then pulled from and returned to OWF, at which time the user has successfully been authorized to browse to OWF in their web browser. If the user's PKI information is not resident within OpenLDAP, even though they have presented a valid PKI certificate, they are not authorized to browse to OWF and will receive an error message.

This mechanism provides a way to protect access not only to OWF, but to individual widgets hosted within OWF. To go a step further, this security mechanism provides granular access (i.e. authorization) to the data displayed by widgets based on the user's attributes stored in OpenLDAP. A RESTful API provides a web service interface to abstract and protect all calls to the Accumulo cloud data store. When a widget calls on the RESTful API to query Accumulo, on behalf of the user, CAS will again intercept this call, pull the user's attributes from OpenLDAP, and pass them down to the Accumulo data-access layer. Accumulo can then filter, or otherwise make decisions upon which pieces of data (all the way down to the key-value cell level) the user can see. This feature provides a much finer granularity of access control than traditional "all or nothing" paradigms where the user is either authorized to see all of the data or none of it. In other words, the user may can view the data based on their need-to-know.

This security architecture supports the use case to filter query results based on the user's profile (e.g. clearance level, rank, nationality), the security labeling of data (e.g. UNCLASSIFIED, DOD, FOUO, LAW ENFORCEMENT), and business rules encapsulated within Accumulo to adjudicate data access. In fact, a Dynamic Security Banner service was built into OWF that would update as each query was run in order to show the 'classification' of currently displayed data, plus whatever that user was allowed to see.

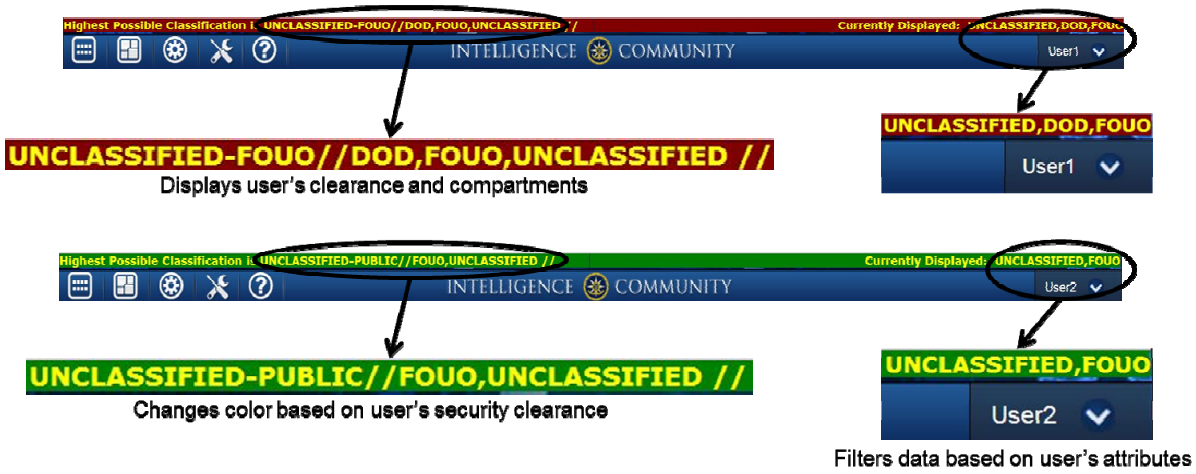

Figure 6: The security banners that are displayed at the top of OWF to every user. These banners are dynamically updated displaying the user's security clearance.

The envisioned use case for this capability is to support coalition and multi-agency operations where different users from different organizations may in fact have a need to know and see different types of data from disparate sources.

4.3 Widgets

The capabilities of this system target two types of users: analysts (those who examine and process the data collected from unmanned vehicles) and operators (those who command and control the unmanned vehicles). With OWF chosen as the common display environment, the system encompasses a collection of widgets created to fulfill the needs that both analyst and operator require. Most widgets are designed to fall into one of two categories; "analysis" widgets or "unmanned vehicle control" widgets, with the exception of the Common Map widget, which is shared between both users. Each category follows a different architecture as further explained below.

4.4 Common Map Widget

A graphical map enhances situational awareness and is often a necessity for both analysts and operators. As a result, the Common Map widget was designed to be a single map shared between both user groups. The widget tracks the current geographical location of the unmanned vehicle while an operator commands and control the vehicle. In addition, the widget can also retrieve the historical geographical locations of the unmanned vehicle from the Cloud and display the data as breadcrumb tracks. The current and historical tracks are rendered as separate layers on the map. This gives the user the flexibility to pick and choose what type of data to display.

4.5 Analysis Widgets

The analysis widgets provide the analyst with the capability to retrieve, display, and manipulate data collected from the unmanned vehicles. The purpose of these widgets is to render the unmanned-vehicle data in a form that is optimally useful to the analyst. In designing the architecture for these widgets, one key focus is to create a pipeline that enables these widgets to access and expose the unmanned-vehicle-collected data stored in the Accumulo Cloud. A custom RESTful API is built as the pipeline for the widgets to have access to the Accumulo Cloud.

A brief description of the analysis widgets created is listed below, along with screenshots of those widgets.

1. Video Widget: This widget displays playback of videos compiled from the images captured by cameras onboard the unmanned vehicle.
2. Thumbnail Gallery Widget: This widget displays the raw images captured by the cameras onboard the unmanned vehicle as a series of thumbnail-sized images.
3. 3D-Model Viewer Widget: This widgets displays the 3D model that was constructed from the raw 2D images captured by the unmanned vehicle. The 3D model is generated leveraging the server-side processing power of the Cloud. Users are able to zoom in/out and rotate the 3D model 360 degrees.

4. Generalized Ingest Widget: This widget enables users to import data from a spreadsheet into the cloud. The widget tags, labels, and classifies the data appropriately before ingesting it into the cloud.
5. Table View Widget: This widget displays sensor data received from the unmanned vehicle in tabular form.
6. Data Viewer Widget: This widget displays relevant position data (i.e. heading, course, longitude, latitude) along with the captured image from the unmanned vehicle at a particular time.

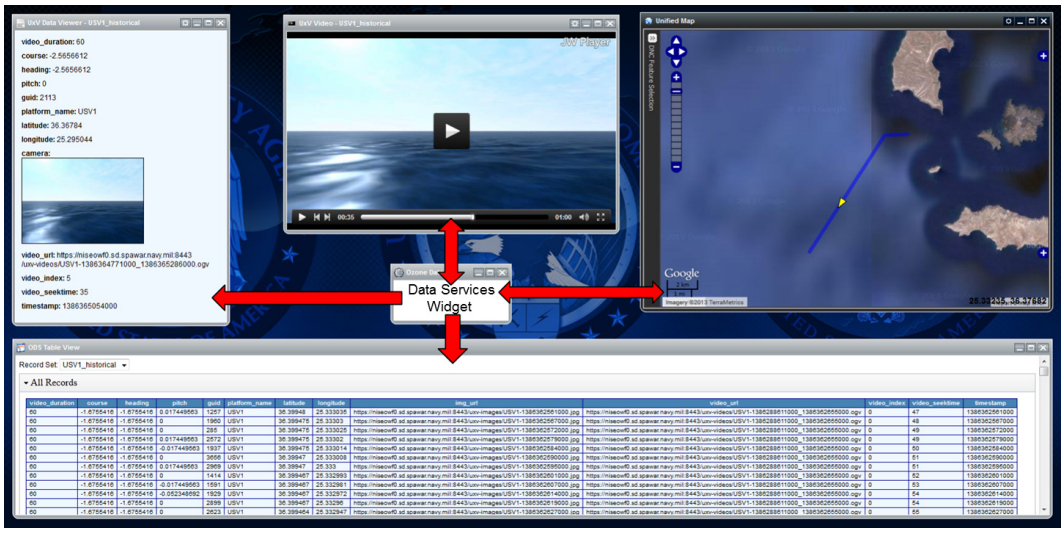

Figure 7: A screenshot of the OWF user interface for an analyst user. The following widgets are launched (going from top left to bottom right): Data Viewer, Video, Common Map, Data Services, and Table View. All the widgets are synchronized with one another where each widget's display gets dynamically updated.

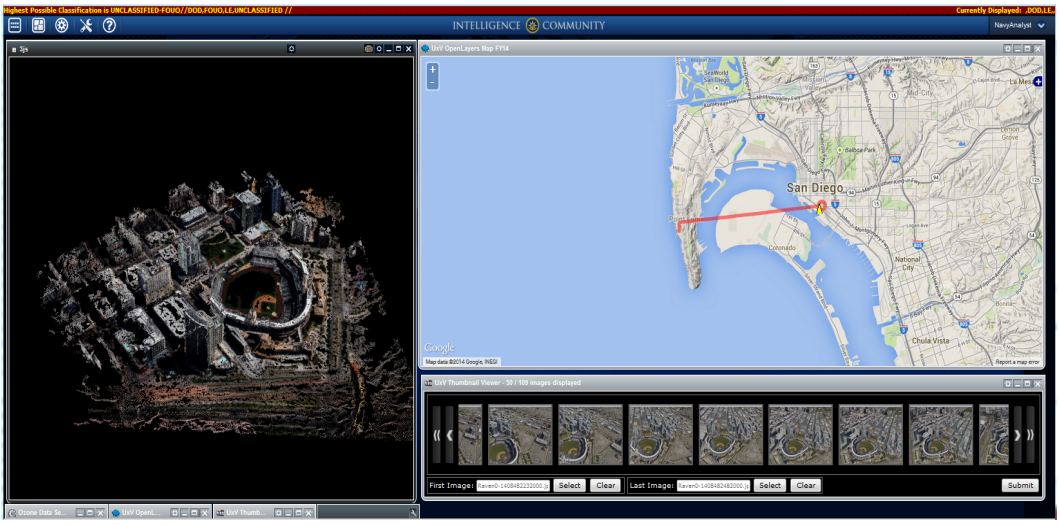

Figure 8: A screenshot of the OWF user interface for an analyst user. The following widgets are launched (going from top left to bottom right): 3D-Model Viewer, Common Map, and Thumbnail Gallery.

Since each widget is designed to provide a single functionality, the Data Services widget is built to connect the widgets together for a cohesive presentation for the user. The Data Services widget utilizes the inter-widget communication mechanism provided within OWF to achieve this goal. For the user, this means that once a track is selected on the Map widget, the other widgets auto-update to display the data correlated to the track selected. As an example, the Video widget would display the video clip that the unmanned vehicle captured from that geographical location of the track and

the Thumbnail Gallery widget displays the collection of images captured with the image associated to that specific track highlighted.

4.6 Unmanned Vehicle Control Widgets

The control widgets provide the operator with the capability to command and control the unmanned vehicle. The purpose of these widgets is to provide operators with the toolset necessary to tactically operate one or more unmanned vehicles. The system is composed of separate control widgets, developed and customized to command USVs and UAVs. All control widgets are built to communicate to the same back-end server running ROCS. Much of the user interface design for these widgets were pulled from another system Multi-robot Operator Control Unit (MOCU)[3].

A brief description of the control widgets is listed below along with screenshots of those widgets.

1. USV Video Widget*:* This widget displays multiple real-time video streams from the unmanned vehicle. The video streams are laterally stitched together to mimic the wider view of a windshield. Below the video are gauges that reflect the current USV status.
2. Video Quad View Widget*:* This widget renders and arranges multiple real-time video streams captured from the unmanned vehicle in the form of a square split into four quadrants. Each quadrant shows a single video feed.
3. Gamepad Widget*:* This widget listens for commands from a joystick (i.e. Gamepad controller) and maps it to commands to send to the unmanned vehicle. This feature enables a user to control the unmanned vehicle using a joystick.
4. UAV Video Widget*:* This widget displays a single real-time video stream along with gauges customized for an UAV. The gauges reflect the current UAV status.
5. UAV Status Report and Control Widget*:* This widget displays the current UAV status as text and provides the operator the ability to command the UAV. A user is able to set the following: radio channel, throttle, altitude, and loiter point (a geographical location around which the UAV will fly).

Figure 9: A screenshot of the OWF user interface display for an operator controlling a USV. The widget launched (top left to bottom right) are as follows: USV Video, Common Map, Gamepad, and Video Quad View.

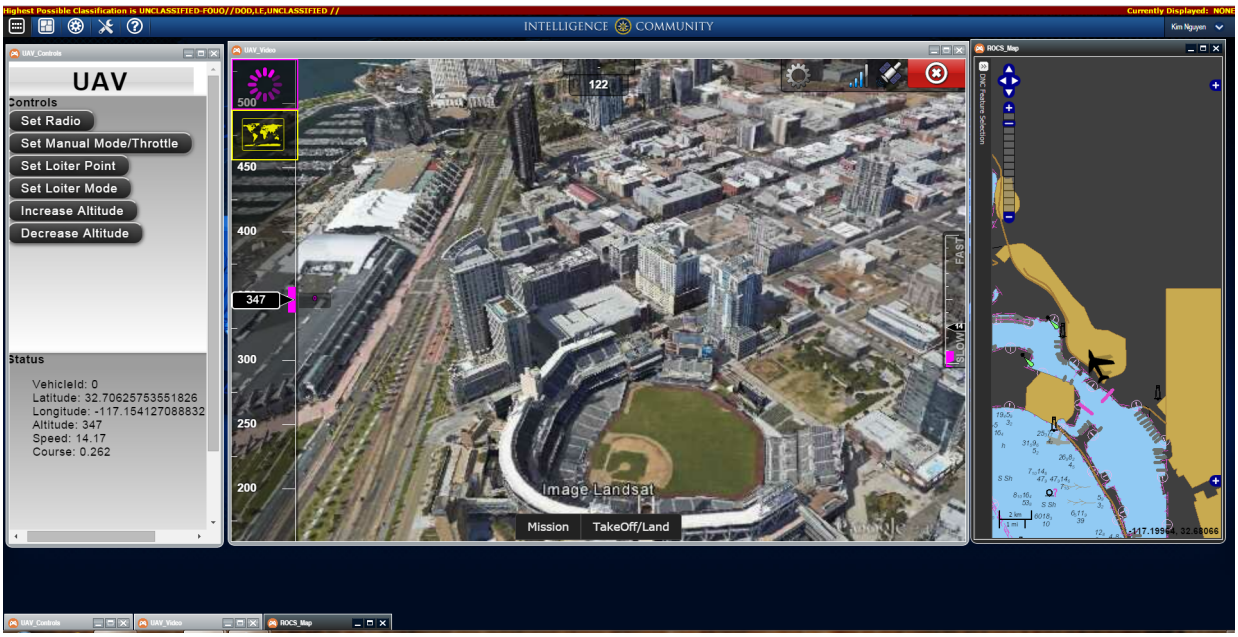

Figure 10: A screenshot of the OWF user interface for an operator controlling a UAV. The widgets launched (top left to bottom right) are as follows: UAV Status Report and Control, UAV Video, and Common Map.

5. SUMMARY

Incorporating cloud technologies with unmanned vehicle control software broadens the opportunities and uses for unmanned systems in military applications. By utilizing a cloud infrastructure, the data collected from unmanned systems can be stored in the cloud, thus making it easier to distribute the appropriate data to various consumers. With current unmanned-vehicle control systems, history of the data collected is only made available to the operator on that local hardware instance running the software. Analysts and strategic decision makers are restricted in their ability to obtain the unmanned vehicle data and share any new findings discovered after processing that data. The *UxV to the Cloud via Widgets* system developed at SSC Pacific aims to demonstrate the added benefits of using cloud technologies to distribute unmanned vehicle data.

REFERENCES LINKING

[1] Lubbers P. and Greco F., "HTML5 Web Sockets: A Quantum Leap in Scalability for the Web," (2013). http://www.websocket.org/quantum.html

[2] Puranik, D. G., Feiock D.C., Hill, J.H., "Real-time Monitoring using AJAX and WebSockets," ECBS '13 Proceedings of the 20th Annual IEEE International Conference and Workshops on the Engineering of Computer Based Systems (2013).

[3] Powell, D., Barbour, D., Gilbreath, G., "Evolution of a common controller", Proc. SPIE 8387, 0277-786X (2012).

[4] Yetman, C. F., Raney C. J., Morris M., George A., "UxB Data to the Cloud via Widgets," 18th International Command and Control Research and Technology Symposium (2013).

[5] Next Century Corporation. "Ozone Widget Framework," (2014). www.ozoneplatform.org

[6] "Data-Centric Middleware Modular Software Infrastructure to Monitor, Control and Collect Real-Time Equipment Analytics," RTI Whitepaper v. 50019, 0514 (2014).

[7] The Open Group. "Service Oriented Architecture: SOA Features and Benefits," (2013). www.opengroup.org/soa/source-book/soa/soa_features.html

[8] Chang, F., Dean J., Ghemawat S., Hsieh W. C., Wallach D. A., Burrows M., Chandra T., Fikes A., Gruber R. E., "Bigtable: A Distributed Storage System for Structured Data," OSDI'06: Seven Symposium on Operating System Design and Implementation (2006).
[9] Sqrrl Data. "Accumulo," (2015). www.sqrrl.com/product/accumulo
[10] The Apache Software Foundation. "Welcome to Apache Hadoop," (26 February 2015). www.hadoop.apache.org
[11] The Apache Software Foundation. "Welcome to Apache ZooKeeper," (2010). www.zookeeper.apache.org
[12] The Apache Software Foundation. "Apache Thrift," (2014). https://thrift.apache.org