Data Analytics and Visualization Application for Asset Health Monitoring

Richard Carley[1], Sara Fuller[2], Glenn Bond[3], Parker Jones[4], David Allen[5], Adam Jordan[6], and T.C. Falls[7]

[1,4,5,6,7]*Mississippi State University: Institute for Systems Engineering Research, Vicksburg, Mississippi, 39180, USA*
rcarley@iser.msstate.edu
parkerj@iser.msstate.edu
allend@iser.msstate.edu
tcfalls@iser.msstate.edu
adamg@iser.msstate.edu

[2]*Mississippi State University: Center for Advanced Vehicular Research, Mississippi State, MS, 39759, USA*
sfuller@cavs.msstate.edu

[3]*US Army Engineer Research and Development Center, Vicksburg, MS, 39180, USA*
William.G.Bond@erdc.dren.mil

ABSTRACT

Much of the research on predictive maintenance has focused on statistical and machine learning techniques, while there has been significantly less focus on the human computer interaction or visualization aspects of PHM. Human computer interaction and visualization techniques can quickly help identify interesting data sub-domains from assets, time periods, and sensors provided the data can be queried, retrieved, and displayed in a timely manner. Augmenting visualization and interaction with a visual, aggregative fleet-based query system adds a further dimension, highlighting the ability of the fleet to carry out its mission. Visualizing data from an asset with a multitude of sensors in a way that fosters human understanding and decision making is challenging from the standpoint of dimensionality. That difficulty is significantly compounded as overwhelming numbers of assets of varying type are added to comprise a hyperdimensional dataset.

In this paper, we propose a scalable framework that is capable of visualizing past, current, and prognosticated health from the individual sensor up to the fleet or group level. In addition to viewing near real time sensor data, maintenance logs, fault information, and data aggregations will be merged with the sensor data to make the analysis and visualizations more valuable. This framework is scalable regarding how much data can be collected, stored, and processed, and the different organizational levels within a fleet of assets. The framework is built as a web-application primarily using the following visualizations: a collapsible tree structure for asset information; 2D charts for temporal sensor data, fault data, and maintenance data; and 3D digital twins of critical components. These components combine to optimize human-computer interaction for decision making across several phases of operations and support for Army ground vehicles.

The dataset used to build and demonstrate the capability of the web-application contains sensor readings from over 3000 vehicles and comprises approximately 9TB of data. Vehicle information such as model, make, sub-component, and fleet organization are presented in a configurable, collapsible tree structure. This allows the user to visualize the fleet and to select the asset and sensor combinations needed to display temporal sensor data to answer a nearly infinite number of questions using individual or combined statistics. Information regarding each vehicle's health status is displayed then aggregated and displayed for each of the higher tree nodes. A 3D digital twin also highlights sensor locations and current health status of assets and components. These component models can be viewed and manipulated with or without a virtual reality headset to provide diagnostic and repair support. As health status monitoring for asset subcomponents are developed, they can be added to the system, allowing for complete health status reporting.

1. INTRODUCTION

While there has been a lot of research into various statistical and machine learning techniques for predicting Remaining Useful Life (RUL), very little has focused on optimizing decision support from a visualization or human-computer interaction aspect. In this paper we propose a visualization framework capable of displaying past, present, and prognosticated asset health information at different organizational levels.

The primary focus of the framework is to support fault diagnosis for individual assets and higher-level overviews that provide aggregated health information for groups of assets. The goal is to be able to provide managers, technicians, and mechanics with the information they need to make informed decisions about how best to deploy assets, manage maintenance schedules, order replacement parts, visualize repairs, and more. Users should be able to instantly view high level fleet health statistics and dive down to individual sensor data, timestamped at the sub-second level.

Along with sensor and health data, the tool is also able to integrate output from predictive models, diagnostic trouble codes, and maintenance records into visualizations, aiding in fault diagnosis. Fault prognosis information is provided in the form of estimated time to failure based on RUL predictions. The data used to demonstrate the framework represents different assets, each with approximately 100 sensors. While the data and use cases here represent the management of a fleet of vehicles, this framework could easily be applied to any setting where managing groupings of assets is required, such as an industrial setting.

Three visualization types are used within this tool:

- Tree Structure
- 2D Time-Series Graphs
- 3D/Virtual Reality digital twins

Three webpages, Unit Status, Asset Details, and Status Reports provide a hierarchical overview, including visualizations for the selected data. The Unit Status page provides visualizations displaying vehicle status and aggregated unit information in support of unit level operations. The Asset Details page provides an overview with a user defined structure which supports more rigorous data analysis. Both pages contain a tree visualization with interactive nodes capable of opening interactable 2D time-series graphs or a new tab with 3D/VR visualizations. The Status Reports page displays automated reports for individual assets and aggregated data for the unit. This paper focuses mainly on the visualization aspects and the tools used to build these visualizations.

2. WEB-BASED APPLICATION ARCHITECTURE

Based on the fleet management use cases, the framework is required to process and display large amounts of data and to communicate technical information at multiple organizational levels to users in different roles. Large datasets present challenges in processing data and displaying information in a timely manner. Given the volume and complexity of both the data and questions asked of it, displaying that information in a way that does not obscure trends or overwhelm the user is also a challenge. Providing solutions to these challenges is essential to address the needs of fleet managers on a daily basis.

Our most significant obstacle was displaying data with enough informational value without overwhelming users cognitively. Our attempt to improve the understandability of such a large dataset involves using summaries and interactivity. A good summary informs users where they should focus their interest, while interactive elements allow them to refine their questions and look at more detailed information. Our summary takes the form of a user-configurable tree since we are displaying organizational data which already has a pre-defined hierarchy.

The web-based application architecture was chosen so that the large dataset resides on a server, eliminating the need to disseminate large datasets and updates that occur regularly for the selected use cases. It also has the advantage of being able to run on most operating system using web browsers, eliminating the need to install specialized software and any software updates only need to be applied to the server. The web-based application was developed using Blazor for the webpages and a combination of Unity, WebXR, and Vega/Vega-lite

Our framework is unique in that it provides a unified way of looking at high level fleet information along with detailed technical information about singular assets. Another unique addition is the VR/3D views. These views provide opportunities for training or pre-maintenance discussions where technicians can better visualize and point out areas of interest. No current software that we are aware of has this level of interactivity along with the combination of previously mentioned features.

2.1.1. Blazor Web Assembly

Selection of a web-application development framework was crucial to the development of our tool. A web-application development framework handles the complex operations of serving web pages and responding to user input, allowing programmers to focus on developing the application. Microsoft's Blazor Web Assembly (BWA) (Roth, 2019), an open-source web application framework, was chosen since it offers a seamless connection between web languages (HTML, JavaScript, CSS) and C#, and allows for modular production of pages and components. BWA also allows for computation on the client instead of a server. This increases startup time but importantly increases responsiveness once the page is loaded, distributing computational load across networks for scalability.

2.1.2. Vega/Vega-Lite

The 2D graph visualizations and tree structures are powered using an open-source, high-level grammar library called Vega-Lite. (Satyanarayan et al., 2017) This library uses a declarative JSON syntax to create specifications to describe how visualizations should be rendered. It's built on top of the Vega visualization grammar but is a much more concise language. Vega-Lite allows for graphs to be directly embedded into web pages using Vega's JavaScript runtime and can display graphs in any modern browser. The library was chosen since it provides many features for creating interactive graphs, data loading, transformations, scales, map projects, and can respond to input streams. It also has several different application programming interfaces for many popular, general-purpose programming languages like Python (Satyanarayan et al., 2017) (Wongsuphasawat et al., 2022). Interactivity was an especially important factor in choosing a visualization library due to the large amount of data the system addresses and its hierarchical nature. This feature allows the user to control how much and what data to view, and to focus on more detail about a specific data point without adding unnecessary information.

2.1.3. Unity/WebXR

The Unity 3D game engine is a software application that allows for the creation of three-dimensional environments combined with scripting logic which can then be exported as executable applications for end users (Goldstone, 2009). Unity was chosen as the main development tool for the eXtended Reality (XR) portion of the project due to its integration with WebXR and a WebXR exporter software library.

WebXR is an Application Programming Interface (API) used to communicate with VR and AR devices (MacIntyre, Smith, 2018) and the WebXR exporter allows a Unity created application to be exported as JavaScript files that can then be used as a web application. Unity and WebXR are used together to display 3D models of critical components to users either in a browser or using VR devices.

WebXR Device API support to Unity is provided by Mozilla's WebXR exporter library, which is compatible with Unity version 2021.2.5f1 (Weizman, 2022). To enable player movement, two C# classes were created to handle 3D movement and these two classes monitor the keyboard, mouse, and the Meta Quest Touch controllers for user inputs and adjust the user's in-scene position accordingly. Since WebXR is hardware agnostic, HTC Vive and other VR systems can also be used instead of a keyboard and mouse and the Meta Quest 2 ("WebXR", 2022). We have tested the VR capabilities with the HTC VIVE and Meta Quest 2. As of now, Chrome does not support VR, while Firefox and Edge require VR to be enabled in the browser's settings. The browsers must be configured to use the computer's discrete GPU within Windows Graphics Settings.

2.2. Data Analytics and Visualization System

Dataset operations for the application are provided by the Data Analytics and Visualization System (DAVS), the result of a joint research program, sponsored by the US Army Engineering Research and Develop Center (ERDC), between Mississippi State University, and Hottinger, Brüel and Kjaer Solutions, LLC (HBK).

DAVS is a collection of software designed to provide an end-to-end solution (loading, cleaning, analyzing, and visualizing data) for scalable analytics and visualization. It is focused primarily on the efficient processing of massive amounts of temporal data collected from various types of assets. While DAVS currently utilizes MongoDB for data management, it has an API for interacting with data and provides database agnostic capabilities to DAVS applications. Apache Kafka (Apache Software Foundation, 2022), a high throughput, distributed, event streaming platform is used as the basis for the distributed processing capability provided by DAVS. While other open-source software tools are used in DAVS, MongoDB and Kafka provide the primary capabilities required by the framework presented in this paper.

The use of MongoDB and Kafka by DAVS provides several distinct advantages during the operations to import, clean, process, analyze and visualize temporal sensor data. Kafka provides a distributed, asynchronous, fast, and fault tolerant messaging system. This capability is used by DAVS essentially as a task queue to provide distributed processing. This functionality is heavily used during data loading, analytic, and querying operations. Scalability is achieved by adding hardware and increasing the "worker" processes allowing DAVS to be executed on platforms ranging from laptops to super computers. For this research, DAVS was configured using three hardware nodes for MongoDB and one node for Kafka where each node in the system had dual CPUs, each with 20 cores and 40 threads. MongoDB was configured with one node as the controller node and two nodes for data distribution. Even though the dataset size was over 9TB and contained over 40 million records, the configuration allowed queries retrieving specific sensor data for an asset during a given time period to be executed in less than a second. When the data throughput of DAVS is combined with asynchronous operations and software multi-threading the result is the ability to display multiple graphs within a few seconds of user selection. This provides very responsive web-page interaction to the user. Scalability for significantly greater volumes of data can be achieved by adding computational resources.

2.2.1. MongoDB

MongoDB is a NoSQL, distributed database management system designed for scalability, flexibility, and agility (MongoDB Inc, 2022). It uses JSON-like documents with optional schemas to perform queries that are optimized to retrieve data efficiently. DAVS utilizes many MongoDB features such as load balancing, replication, aggregation, ad hoc queries, and many more. It can also be used as a grid file system over multiple machines along with the load balancing and replication features as data volumes exceed the abilities of one machine. There are many MongoDB drivers available for the most popular programming platforms, including a .NET driver for C#. The .NET driver is used to query data on demand whenever the web app makes a request to the web server (MongoDB Inc, 2022). MongoDB is an essential element of DAVs, so it was necessary to include MongoDB as part of our framework

2.3. Dataset

The dataset used for this paper contains a record of over 3,000 vehicles, each with over 100 sensors. The dataset is over 9 TB in size, and is broken up into channels, including startup, operational, and fault data. Each channel represents a sensor, and multiple sensors can be combined, as inputs to functions whose outputs represent the health of various components. We define these components as parts, or groups of parts, that serve a specific purpose in the vehicle, such as an engine, battery, or transmission.

Assets, or individual vehicles, are identified by a unique Vehicle Identification Number (VIN), and can be grouped by vehicle group and subgroup, as well as location, unit identification code, and squad. Maintenance data for these assets is included to help determine what maintenance has been performed and how often it occurs.

2.4. Reasoning/Comparison to Other Tools

Alves et. al. (2020) created a system that used a web application to connect sensors and a database to a webpage dashboard and a Microsoft Hololens for Augmented Reality (AR) visualization. Their aim was to be able to visualize current data and aid in earlier detection of failures for a specific machine. Our system aims to address some of the same issues but differs in three main ways: it uses Virtual Reality(VR) instead of AR, it handles thousands of assets with over 100 sensors each and focuses on RUL as well as finding existing failures. The number of assets and volume of data we are dealing with creates substantial hurdles from a data processing, networking, and human cognition perspective, all of which are addressed in this paper. Furthermore, VR-based systems can reduce the time to learn tasks, and aid in training, and are already being utilized to aid in maintenance for manufacturing environments (Buettner et. al. 2022).

Based on the use cases defined for our framework, we knew it had to be able to process and display large amounts of data and be able to communicate technical information at multiple organizational levels to users in different roles. Large datasets present challenges in processing data in a timely manner, as well as in being able to display that information in a way that does not obscure trends or overwhelm the user. With a proper mongo database set up, DAVS is able to handle data processing quickly despite the large dataset size, therefore our biggest hurdle was in displaying data to the user. Our attempt to improve the readability of such a large dataset involves using summaries and interactivity. A good summary can tell users where they should focus their interest, while interactive elements can allow them to look at more detailed information. Our summary takes the form of a tree since we are displaying organizational data which already has a pre-defined hierarchy.

A web-based platform was chosen since the data would have to reside on a server due to its size. It also has the advantage of being able to run on most browsers, eliminating the need for a user to install specialized software. Modularity is simple with a web page, and any updates only need to be applied to the server. However, this does mean that users will need a stable connection, either directly on the server's network, or through a VPN.

Our framework is unique in that it provides a unified way of looking at high level fleet information along with detailed technical information about singular assets. Another unique addition is the VR/3D views. These views provide opportunities for training or pre-maintenance discussions where technicians can better visualize and point out areas of interest. No current software that we are aware of has this level of interactivity along with the combination of previously mentioned features.

3. FEATURES AND VISUALIZATIONS

This section will discuss and describe the various visualization methods employed by our tool. This includes the motivation and implementation of each visualization, as well as how to interpret and interact with them.

3.1. Tree Structures

The primary interactive visualization for our framework is a tree where each level represents an organizational level of a specific unit or the entire fleet of vehicles. The root node represents the highest organizational level of the selected data, and the bottommost leaf nodes represent sensor or asset data. Some of the specific implementations of trees will be discussed in greater detail in the Demo/Webpage Overview section, but an example can be seen in Figure 1. Our system is currently set up to use RUL values as a measure for vehicle health, but future versions will incorporate other health indicators or indices to be used in place of or alongside RUL.

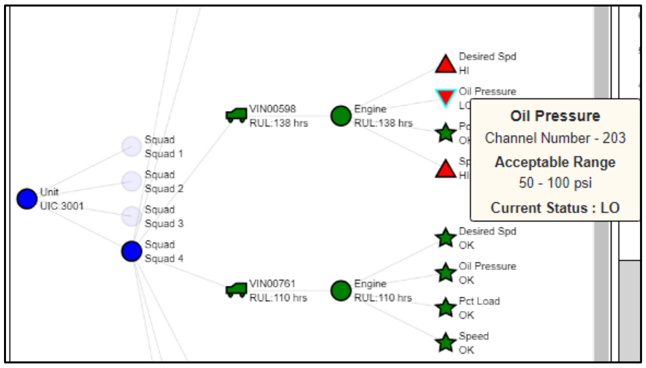


Figure 1. Example of tree visualization with mouse hover over the engine's oil pressure sensor

The nodes of each of the trees are coded by both color and symbol to aid human cognition with easier to recognize indicators of health status. Green indicates healthy status with a relatively long estimated RUL. Yellow indicates that potential problems will need to be addressed, reflecting a lower estimated RUL value. With this indicator, users should be wary and observant when tasking or operating the asset and components, but the asset is usable if the mission requires. Red indicates that problems are imminent based on estimated RUL, and action should be taken, or maintenance performed before the asset can be returned to mission-capable status. Lower-level nodes such as those representing sensors are also coded by symbol to indicate health. Star patterns mean healthy sensor readings, and up and down arrows are used to indicate that a sensor is detecting higher or lower than normal output.

Tree visualizations are used in the webpages to provide relevant information and links to child nodes and other information. When the user hovers the mouse over a node, the system will provide relevant health information. At higher levels in the tree, this will include aggregated health information such as the number of healthy units or assets within an organization, while information from lower-level nodes will be more specific to assets or components. Only specific components critical to the operation of the vehicle are displayed as nodes to limit the amount data displayed at this level.

Since the tree visualizations can get large, all nodes except the lowest levels of the tree are collapsible. Clicking on the bottom most nodes will always open graphs in the right side of the page. Clicking component nodes in the Unit Status page will also allow the user to open a 3D/VR visualization of the component in another browser tab. These interactive elements allow the user to better customize their visualization by either minimizing unnecessary information or linking to more specific charts and data. The following sections provide more information concerning the 2D time-series graphs as well as the 3D/VR visualization capability.

3.2. 2D Time-Series Graphs

2D time-series graphs are created using the Vega-Lite library. They contain sensor, fault, and maintenance data. Sensor data is represented by a blue line, while the fault and maintenance data are represented by red circles and diamonds, respectively. Multiple graphs can be opened at one time, in one frame.

There are two aspects to each graph. The top graph shows the entire timeline of the sensor, which can be months or years, while the bottom graph shows a selection of the timeline as defined by the user. The red vertical bars in the upper graph represent the time period displayed in bottom graph. Time periods can be selected in the upper graph by either using shift and the mouse scroll wheel, or by left clicking and dragging the mouse. Similarly, the buttons on the right side of the graphs can be used for zooming and panning as well. Graphs of sensor data from the same vehicle can be synchronized as seen in Figure 2, allowing the user to view simultaneous drops in "Voltage" and "Oil pressure", for example.

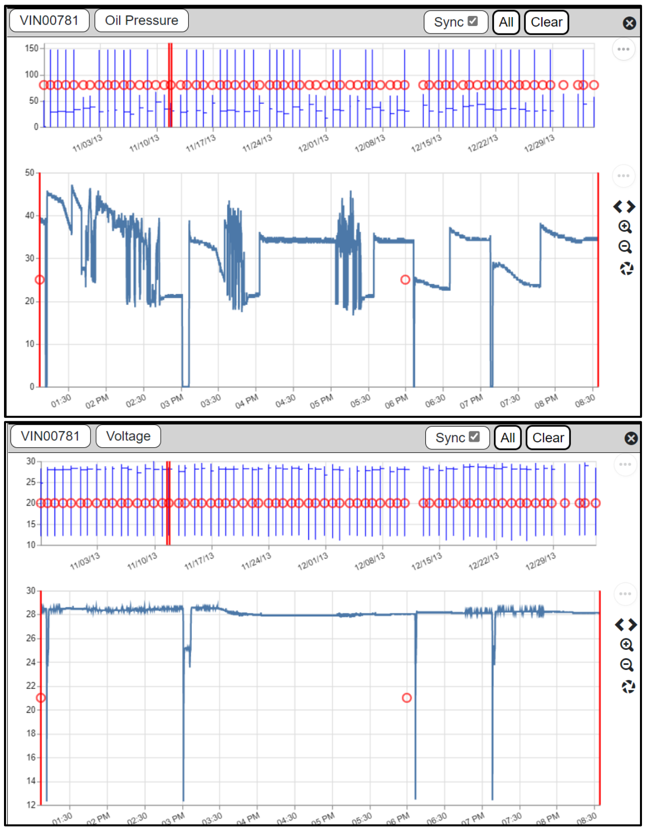


Figure 2. Example of multiple 2D time-series graphs synced

3.3. Status Reports

The status report page is a PDF display page containing a PDF generated during the loading of the page based on data in the database. Currently it displays the total of one unit grouped by a tabbed list of Squad, Asset, Component, and then Sensor. Each Squad has an assessment of how many assets are at or above a predetermined RUL estimate. The asset, component, and sensors display their associated RUL value next to their icon to represent their current condition. The condition of the child objects, in aggregate, determine the readiness of the parent that they are attached to. For example, a component or components attached to an asset may affect the RUL estimate of the asset. These reports provide a standardized method for communicating a unit's vehicle readiness that can be used for reporting and briefings.

3.4. 3D/VR Digital Twins

This section discusses the implementation and use of 3D models to display health information. This includes the reasoning for the use of the 3D models, their implementation into the web framework, and basic instructions for user interactions.

3.4.1. Overview of Features

A Digital Twin is a virtual representation of a physical object along with associated information that could be considered meta-data or generated operational data. With two dimensional, visual digital twins, multiple profile views must be created to display sensors within the component. This would increase the cognitive load on the user by requiring interpretation and mapping profile view layers together to understand and gain knowledge from the holistic view. To avoid this situation, our system uses interactable three-dimensional objects to be significantly more intuitive than two dimensional representations. 3D rendering represents a departure in terms of knowledge discovery, rapid identification of conditions, and annotation of digital twins of specific components with annotated health information aggregated from different sensors. This improvement in information synthesis allows users to monitor the status of the physical object in a concise user-friendly digital format (Liu, 2022). These visualizations can be viewed with or without AR/VR hardware.

3.4.2. 3D/VR Model Preparation

The four example models used to show the 3D/VR capability of the software: an Abrams Tank, a Honeywell AGT1500 Gas Turbine Engine, a generic vehicle transmission, and a vehicle battery, were downloaded from grabcad.com and sketchfab.com. These models were in the FilmBox (.fbx), Digital Asset Exchange (.dae), and object (.obj) file formats. These three file formats are compatible with the Unity 3D game engine. The four models were imported into Unity 2021.2.5f1, and material (.mat) file types were extracted from the imported version of the models. The texture component of these material files was then remapped to separately downloaded Portable Network Graphics (.png) files so the models and their textures appear correctly within the scene. From the imported models, prefabricated objects, or prefabs, were created for use within the scene. Box colliders were attached to each prefab and oriented so that the collider center point matched the model center point and boundaries expanded to allow the user to grab any region of the prefab. Rigid bodies were also attached to the prefab and set to simply follow the user's hand or mouse cursor when selected. When released they are locked in place until picked up again or the reset button is pressed, allowing the user to concentrate on the model and its condition according to its specific data.

3.4.3. 3D/VR Integration with Web Dashboard

Sensor data is represented in a hierarchy such that the vehicle is the root object, components are child branches from the vehicle, and sensors are child branches of the components. When a user clicks on a vehicle component, the component is shown in the VR/AR environment. The URL to display the component also contains the information regarding current sensor information.

As an example, looking at Figure 3, the information sent to unity contains the 3D model name of the critical component, the Unit Identity, Vehicle Identification Number, Remaining Useful Life, Transmission Oil Temperature value, and the Crankcase Pressure value. The information containing the string "transmission" is the first parsed value and is used to toggle on the 3D transmission object and update text boxes within the scene based on the other values in the provided query string. In this example, the query string sent two values, 0 and 2, for the Crankcase Pressure and the Transmission Oil Temperature sensors respectively, which corresponds to the "Low" and "OK" messages shown in Figure 3.

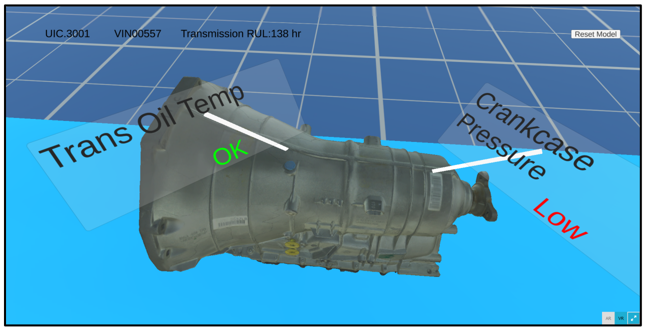


Figure 3. Viewing a digital twin in 3D space

3.4.4. Instructions for Navigation Within 3D/VR

For a non-VR user, pressing "W" on the keyboard moves the user in the positive Z direction (forward) within the scene. In VR, when the user moves their left controller's thumb stick more than 10 percent in the real-world positive Y direction, they move in positive Z direction of the head mounted display within the VR scene. Selected components can also be moved within the scene by using a mouse and keyboard. When the user clicks the left mouse button, a ray is cast in the positive z-axis direction using the mouse cursor's x and y positions within the scene's viewport. If the ray hits a valid object, the object's x and y positions are continually updated to match the user's movement with the mouse cursor until they release the left mouse button, allowing the user to intuitively manipulate components on a flat display.

If the user is in VR mode, like Figure 4 below, the scene monitors the user's controller to determine if the controller and loaded model are colliding. If a valid collision is occurring, and the user presses the "grip" button on the controller, then the model's position will be continually updated to match the position of the controller until the user releases the grip button.

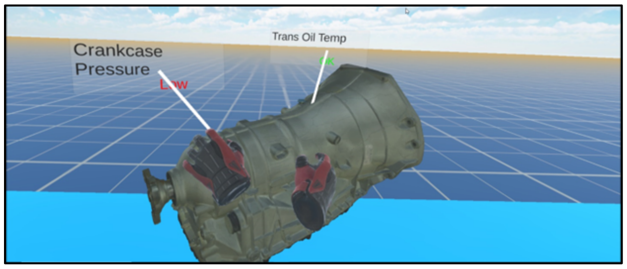


Figure 4. Viewing and manipulating a digital twin in VR

4. DEMO/WEBPAGE OVERVIEW

This section provides a basic walkthrough of each of the webpages. Specifically, we show how we use the visualizations described in earlier sections in a couple of different ways.

4.1. Unit Status

Figure 5 shows the unit status page with a tree visualization and sensor data graphs. This page combines data for each sensor with the tree structure, providing a complete view of current health status along with the historical data, which includes faults as well as maintenance information.

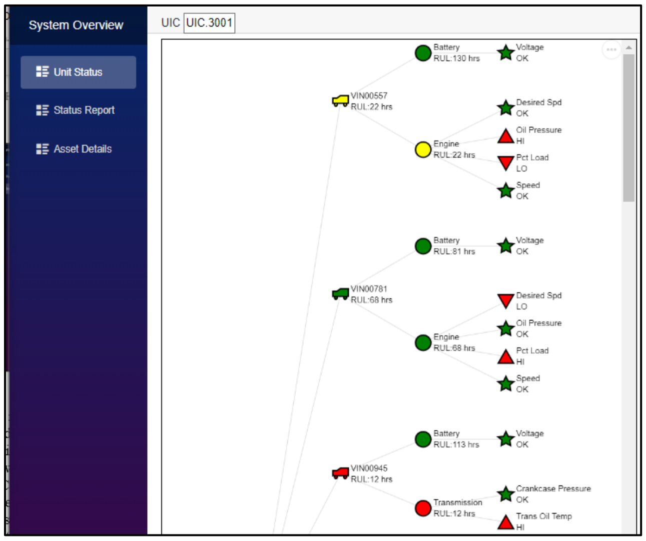


Figure 5. Unit Status View

The unit status page is built to handle all the assets for a unit. The tree follows a fixed structure with the Unit Identification Code (UIC) being the top or root node, followed by assets, components, and finally sensors as the bottom leaf nodes. Specific units are chosen by entering the unit's ID number in the text box labeled "UIC" at the top of the page. Clicking on the components will open a separate tab with a 3D/VR visualization and clicking on the sensors will open graphs on the right-side of the existing page.

Multiple graphs can be opened in the right half of the page by clicking on the sensor nodes. An example of these graphs can be seen in Figure 2. The graphs can be synchronized and interacted with as discussed in section 3.2, while the 3D objects can be viewed and manipulated as in section 3.4.4.

4.2. Asset Details

The asset details page, Figure 6, much like the Unit Status page combines graphs with an interactive tree. However, there are a few differences.

The first is that the user can specify the order that the tree is organized by, using the drag and drop menu at the top of the page. This menu allows the user to choose which groupings the tree will be built on. The choices are group, subgroup, site, UIC, and squad. The bottom nodes of this tree will always be the asset itself. The choice of a dynamic hierarchy allows users to more easily create a tree that works for the visualization or querying needs, as opposed to the fixed hierarchy of the Unit Status page.

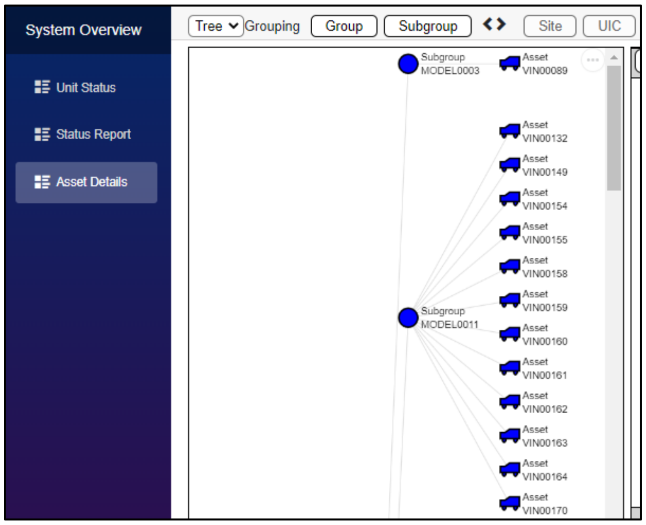


Figure 6. Asset Details page

A second difference from the Unit Status page is that by clicking on the asset, graphs will open in the right side of the page, however with these graphs, users can select from all the individual sensors related to the asset, as shown in Figure 7. The Unit Status page only shows the sensors for whatever the user has deemed a critical component. The asset details page allows a deeper dive into all of the data available for that asset.

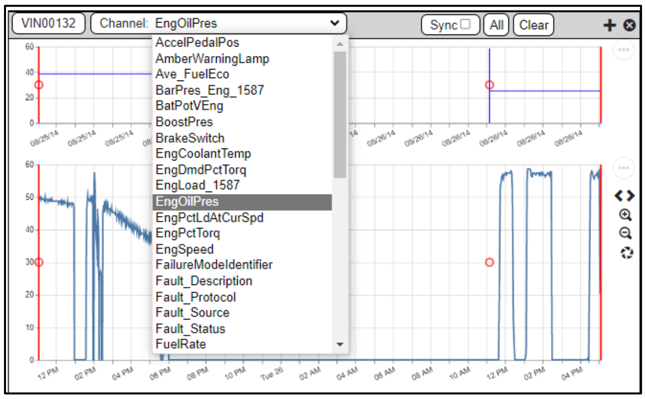
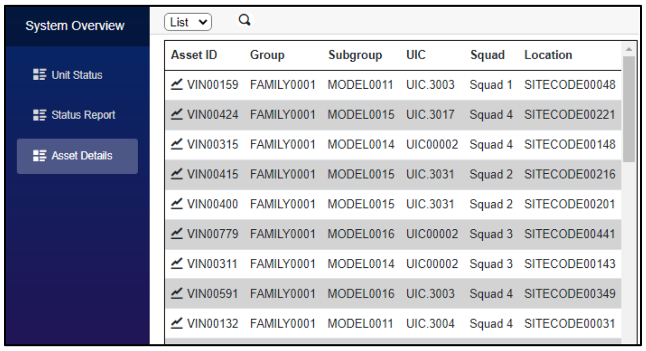


Figure 7. Selecting from an asset's sensors This figure shows on the right side of the Asset Details page

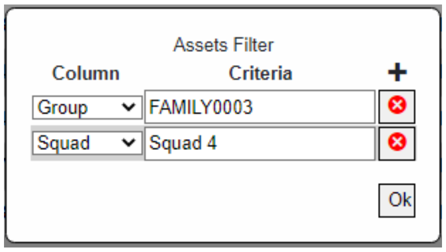
The default view for the left side of the page is the tree view, but users can select a list view from the drop-down menu on the top left of the page. The list view shown in Figure 8, lists the individual components along with the same groupings from the tree view.



Asset ID	Group	Subgroup	UIC	Squad	Location
VIN00159	FAMILY0001	MODEL0011	UIC.3003	Squad 1	SITECODE00048
VIN00424	FAMILY0001	MODEL0015	UIC.3017	Squad 4	SITECODE00221
VIN00315	FAMILY0001	MODEL0014	UIC00002	Squad 4	SITECODE00148
VIN00415	FAMILY0001	MODEL0015	UIC.3031	Squad 2	SITECODE00216
VIN00400	FAMILY0001	MODEL0015	UIC.3031	Squad 2	SITECODE00201
VIN00779	FAMILY0001	MODEL0016	UIC00002	Squad 3	SITECODE00441
VIN00311	FAMILY0001	MODEL0014	UIC00002	Squad 3	SITECODE00143
VIN00591	FAMILY0001	MODEL0016	UIC.3003	Squad 4	SITECODE00349
VIN00132	FAMILY0001	MODEL0011	UIC.3004	Squad 4	SITECODE00031

Figure 8. Table View

The magnifying glass icon opens a pop-up window, Figure 9, which allows users to filter the data by column values.


Assets Filter

Column	Criteria
Group	FAMILY0003
Squad	Squad 4

Figure 9. Query selection for table view

This filter can have multiple criteria and uses AND operations to combine the criteria. For example, if a user selects “FAMILY0003” for the Group and “Squad 4” for the Squad, the query will find all the vehicles in the vehicle group “FAMILY0003” that are attached to “Squad 4”. This filtering will persist if the user switches back to the tree view.

5. CONCLUSION

The framework proposed in this paper provides an intuitive means for visualizing assets at different organizational levels for industrial, commercial, or military applications, by querying significantly large, complex data. At higher levels, the tool displays aggregated health information for the asset or unit. The system’s interactivity allows for the tree to be more readable and situationally manipulable, as well as providing links to other visualizations. The use of DAVS and MongoDB provides responsive user interactions as querying and visualizing data in response to a user’s request typically takes no more than a few seconds.

The next revisions of the framework will include improvements such as adding the ability to view more information about maintenance and fault events We will also further improve the usability of status reports by making them more customizable and giving the user the ability to add graphs and figures. Additional future work aims to better integrate output from machine learning techniques used for anomaly/fault detection as well as RUL prediction. We are especially interested in finding ways to better communicate fault probability instead of a single RUL value as this is one path to a health assessment suite. Specifically, we are looking at integrating confidence and prediction intervals along with cumulative probability of failure into the prognosis module. These methods will better reflect the uncertainty and imprecision inherent in RUL predictions.

ACKNOWLEDGEMENT

This research was conducted by Mississippi State University under contract to the U.S. Department of Defense (DoD) High Performance Computing Modernization Program, through the US Army Engineering Research and Develop Center (ERDC) Contract #W912HZ21C0014. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Army ERDC or the U.S. DoD.

REFERENCES

Alves, F., Badikyan, H., Moreira, H. A., Azevedo, J., Moreira, P. M., Romero, L., & Leitao, P. (2020). Deployment of a smart and predictive maintenance system in an industrial case study. 2020 *IEEE 29th International Symposium on Industrial Electronics (ISIE)*, 493–498. doi: 10.1109/ISIE45063.2020.9152441

Apache Software Foundation (n.d.) *Apache Kafka* https://kafka.apache.org/

Buettner, R., Breitenbach, J., Wannenwetsch, K., Ostermann, I., & Preil, R. (2022) A systematic literature review of virtual and augmented reality applications for maintenance in manufacturing. 2022 *IEEE 46th Annual Computers, Software, and Applications*

Conference(COMPSAC), 545-552. doi: 10.1109/COMPSAC54236.2022.000

Goldstone, W. (2009). *Unity Game Development Essentials*. Packt Publishing Ltd. 2009.

MacIntyre, B. and Smith, T.F. *Thoughts on the Future of WebXR and the Immersive Web. 2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct).* 2018. pp. 338-342. doi: 10.1109/ISMAR-Adjunct.2018.00099.

MongoDB Inc. (n.d.). *What is MongoDB* https://www.mongodb.com/what-is-mongodb

Liu, Y. K., Ong, S. K., & Nee, A. Y. C. (2022). State-of-the-art survey on digital twin implementations. *Advances in Manufacturing*, *10*(1), 1-23.

Roth, D. (2019). *Blazor server in .NET Core 3.0 scenarios and performance.* ASP.NET Blog. Microsoft. https://devblogs.microsoft.com/dotnet/blazor-server-in-net-core-3-0-scenarios-and-performance/

Satyanarayan, A. Moritz, D. Wongsuphasawat, K. Heer, J. (2017). *Vega-Lite: A Grammar of Interactive Graphics IEEE Transactions on Visualization and Computer Graphics* doi: 10.1109/TVCG.2016.2599030

WebXR Device API Explainer. (n.d.). GitHub repository.https://immersive-web.github.io/webxr/explainer.html#xr-hardware.

Weizman, O. (2022). *Unity WebXR Export*. GitHub repository https://github.com/De-Panther/unity-webxr-export.

Wongsuphasawat, K. Moritz, D. Satyanarayan, A. Heer, J. (n.d.). *Vega-Lite A grammar of interactive graphics* https://vega.github.io/vega-lite/

BIOGRAPHIES

Richard Carley is a Research Engineer with the Institute for Systems Engineering Research (ISER) and a PHD candidate in the Department of Computer Science and Engineering at Mississippi State University. His primary research interests deal with big data analytics and visualization.

Sara Fuller is the Strategic Programs Manager at the Center for Advanced Vehicular Systems (CAVS) at Mississippi State University. Her research interests include PHM and advanced maintenance applications.

Parker Jones is a Research Engineer with the Institute for Systems Engineering Research (ISER) at Mississippi State University.

David Allen is a Research Engineer with the Institute for Systems Engineering Research (ISER) at Mississippi State University.

Adam Jordan is a Research Engineer with the Institute for Systems Engineering Research (ISER) at Mississippi State University.

T. C. Falls is the Associate Director, Software Design and Development for the Institute for Systems Engineering Research (ISER) at Mississippi State University. He provides technical leadership to the ISER for software design, development, architecture, implementation, and integration. His research interest includes data fusion, management, analysis, and visualization primarily in support of advanced maintenance.

Dr. Glenn Bond is a computer scientist at the US Army Engineer Research and Development Center's Information Technology Laboratory. His research interests have centered on High Performance Data Analytics, particularly for US Army wheeled ground vehicles, for the past few years. Glenn collaborates with the MSU team in this capacity.