2021 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
VEHICLE ELECTRONICS & ARCHITECTURE TECHNICAL SESSION
AUGUST 10-12, 2021 - NOVI, MICHIGAN

ENABLING CUSTOM VEHICLE DIAGNOSTICS WITH A COMMON APPLICATION PLATFORM

Andrew Ludwig, Daniel Tagliente
U.S. Army DEVCOM Armaments Center, Picatinny Arsenal, NJ

ABSTRACT

This paper discusses the Diagnostics And System Health (DASH) embedded diagnostics software originally developed for use on the M109A7 / M992A3 Family of Vehicles (FoV). The history and background of work completed by the DEVCOM Armaments Center (AC) System Health & Interactive Future Technologies (SHIFT) Division in developing and managing the DASH program are described. The DASH software architecture and design details are also discussed in depth, with a focus on the more recent efforts to adapt DASH to use a generic core software application that can be integrated on a wide variety of current and future ground combat systems to more easily provide embedded diagnostics capability.

Citation: A. Ludwig, D. Tagliente, "Enabling Custom Vehicle Diagnostics with a Common Application Platform", In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 10-12, 2021.

1. INTRODUCTION

Onboard vehicle diagnostics for both light and heavy-duty military ground systems have quickly gained importance in order to decrease vehicle downtime and increase ease of vehicle maintenance. Vehicle diagnostics, health management, and overall logistics support have often been handled by original equipment manufacturers (OEMs) offering unique, and sometimes proprietary, software solutions for each vehicle platform and utilized complex methods of generating diagnostic solutions. This approach comes with unique challenges and costs, and often leads to increased soldier training due variations in maintenance strategies and equipment from system to system. [1] Organic maintenance is an alternative to CLS in which maintenance activity and diagnostic tools are developed and managed by a government organization rather than the OEM, and should theoretically be less expensive due to government's lack of profit motivation. [2]

Beginning with its involvement in the development of the M109A7/M992A3 Family of Vehicles (FoV), the System Health & Interactive Future Technologies (SHIFT) Division at Picatinny Arsenal's DEVCOM Armaments Center (AC) has developed an embedded diagnostics system to be used for ground vehicle and armament system fault detection, fault isolation, and general maintenance. This organic maintenance solution, known as Diagnostics And System Health (DASH) has been

UNCLASSIFIED
PAO Log # 397-21

matured over several software development cycles and has generated interest from other ground vehicle systems. This interest has led to the SHIFT Division further evolving DASH to offer a completely customizable solution that can easily be adapted to any vehicle with standard automotive or smart line replaceable units (LRUs).

This paper will go into detail explaining the considerations and steps taken to adapt DASH from a platform-specific embedded diagnostics program to a more generic cross-platform solution with platform-specific customization and integration options.

2. DASH OPERATIONS

DASH's primary objective is to help system operators and maintainers to quickly identify fault conditions, easily correct system failures, and to provide overall embedded diagnostics and maintenance capabilities.

The primary software feature implemented by DASH to accomplish these goals is a fault list that presents currently active faults on a system. The displayed faults are mapped from a LRU reported fault into a Universal Fault Code, which is guaranteed to be unique within a vehicle, and allows experienced maintainers to quicky visually recognize what issue is occurring on the vehicle. Each fault also has associated metadata such as description, possible causes, and links to interactive isolation and verification procedures. In addition to fault code information, the LRU repository definitions, diagnostic procedures, and support scripts are all defined in the Vehicle Configuration Package (VCP). This package, which is unique to a family of vehicles, supplies the data necessary for DASH to function in a way that allows meaningful fault reporting and procedure execution on each platform that is supported.

Once installed and configured for a specific vehicle or family of vehicles, DASH will record a fault history that allows operators and maintainers to view historical fault events including fault clearing and setting occurrences and system startup and shutdown times. The fault history is additionally used as part of the Prognostic and Predictive Maintenance (PPMx) logs to be offloaded for prognostic maintenance efforts. DASH will aid in performing Periodic Maintenance Checks and Services (PMCS) and conducting advanced maintenance operations by guiding a maintainer through step-by-step procedures through the user interface. Finally, DASH provides verification of hardware and software configuration data from supported subsystems and LRUs.

DASH uses a wide variety of communication interfaces to connect and share information with system LRUs. To date, DASH has been successfully demonstrated on fielded systems with Ethernet and Controller Area Network (CAN) bus communications, including those utilizing SAE J1939 messaging, but the DASH framework is adaptable to support other interfaces as necessary. The primary data contents transmitted over these interfaces consist of fault information, real-time data values, system and subsystem events, and hardware and software configuration data from LRUs.

In order to use and manipulate the LRU fault and diagnostic data received on a communication interface in a meaningful way, the DASH server is distributed with service plugins that can be enabled or disabled on each supported system. These services are loaded at runtime and use Python glue code, which can be made interoperable in custom and platform-specific ways. An example is the Repository Service which provides a central storage location for all LRU fault and runtime data as well as current system status variables. This service utilizes a publisher-subscriber pattern, allowing glue code to receive and operate on updates to the repository and pass these updates on to other services as needed. As part of the VCP, the Repository Service configuration defining the individual repositories and their fields are stored separate from the core DASH application. This ensures only fields relevant to the current vehicle

and its LRUs are stored. Additionally, these fields can be modified before and after being placed in the repository using pre- and post-write triggers. A common example is to convert LRU data from engineering or raw units to Metric and English units. Executing scripts based on repository field events allows a great deal of flexibility for triggering chains of events within DASH.

Meaningful information is displayed to the user through the DASH server Display Manager Service which connects to one or more DASH Client applications. The client is distributed as part of the core DASH application and can be run on most computers with a display running any major operating system such as Windows and Linux. The client can be used to display current fault information, run diagnostic procedures, check system software versions, and view LRU runtime data values in real time. All the data in the client is updated through the DASH server, making the client completely data-centric, not needing to know anything about vehicle-specific configurations.

3. SOFTWARE CONSIDERATIONS

The core functionality of the DASH application is designed to be communicating to and from LRUs within a vehicle and to display their status information to an end user. To provide this functionality, the DASH application consists of three main pieces, a server, one or more clients, and a vehicle configuration package. The vehicle configuration package is the key to providing DASH with the logic necessary to perform the diagnostics and communicate with LRUs on an individual vehicle. Without this package, the DASH client and server will function in a limited vehicle-agnostic way, but lack the necessary substance for meaningful or useful diagnostics. By having a hard separation between the application and the configuration, DASH can maintain a look, feel, and functionality that is common between every vehicle platform, while also providing tailored fault reporting and diagnostics.

To coordinate the vast amount of information within a vehicle platform that is maintained within DASH, the server application is developed using a proprietary framework call “pyFramework.” This framework is a Service Oriented Architecture (SOA) framework that utilizes components and plugins to perform specific functions. PyFramework was designed to be data-centric, net-centric, configurable, scalable, and scriptable architecture where specialized services are able to be developed for specific platforms and included with the DASH installation. This allows the flexibility to leverage the plugin and scriptable nature to develop extensive vehicle specific capabilities. Although pyFramework is a proprietary piece of software, the government has the right to use it for government purposes, and it is available for use by as a diagnostic solution on any government platform. This differs from the current software landscape where an OEM may see their vehicle software as a competitive advantage and not be willing to license it to competitors. Since SHIFT is not a vehicle OEM, there is no such conflict in allowing vehicles platforms to utilize DASH and its diagnostic capabilities. The following sections will outline the common vehicle components within pyFramework.

3.1. Core Service

The core service is responsible for initializing the pyFramework by loading each service within the framework and providing the service instance the interface to its configuration data. The definition of which services to load and their corresponding configuration data is defined in a configuration file as part of the vehicle configuration package. As seen in Figure 1, at runtime, the core service can selectively load different child services based on the vehicle configuration.

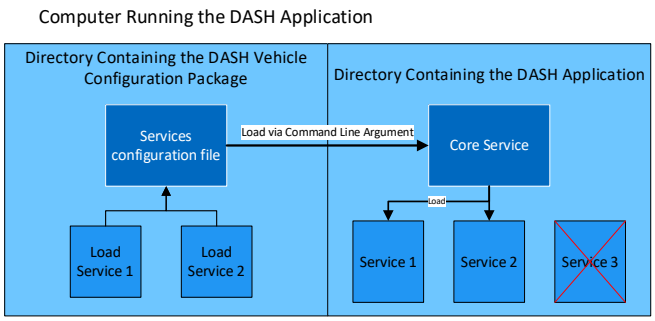

Figure 1

The Core service is also responsible for stopping each service when shutting down the DASH server and for monitoring each service for threading timeouts.

Finally, the core service framework is responsible for exposing all loaded child service interfaces to each other. Each service developed to function within pyFramework inherits from a common group of service interface classes and runs its own thread.

Each interface within pyFramework provides boilerplate functionality for building a pyFramework service to run within an instance of the DASH Server. As seen in Figure 2, the root class "IBaseService" exposes the service name and the method for the Core service to get the interface the service exposes to other pyFramework services.

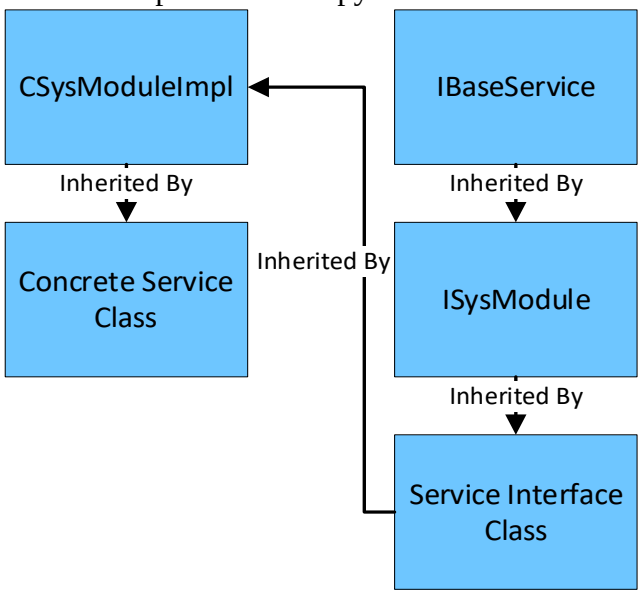

Figure 2

The "ISysModule" provides the functionality to run the service in its own separate thread. The "ISysModule" declares the virtual functions to allow a new service to handle initialization, starting, stopping, and cleanup when directed by the Core service. The "Service Class Interface" provides the interface functions that are assessable to other services when the instance of the new service is queried utilizing the "IBaseClass" function. This is the same interface too that will be exposed as a Python module at runtime making the service accessible to the Python glue scripts. The "CSysModuleImpl" class contains boilerplate implementations of "IBaseService" and "ISysModule" functionality. For most new services, this will suffice, but is overridable in the concrete implementation of the class. The concrete implementation defines the logic used to determine how the service will function. In the case of the repository service, the concrete class contains the data structures to maintain the individual repositories and their fields. Through the interface defined for the Repository service, other services may retrieve field values for reading and writing and access the publish functionality to set up their subscriptions to individual field values.

By default, the Core service automatically provides the interfaces for the common services outlined in the next sections. Through its interface however, a service can be searched for based on the name defined in the service configuration. Platform engineers can choose to define these new services for inclusion in the vehicle configuration package and provide their plugins for loading into the DASH server application at runtime while utilizing these boilerplate pyFramework interface libraries.

3.2. Repository Service

The Repository Service's function is to manage the data used internally in DASH and to provide a publish-subscribe interface to this data to the rest of the server application and to the external maintenance interface. Upon startup, this service loads a data schema for each repository, which

provides the data type, depth (historical values), persistence over power cycles behavior, initial value, conversion functions, and post- and pre-trigger functions for each field. If an LRU is defined to support different hardware variants that require different data and thus different repository structures, DASH can be configured to load a common repository until communication with the LRU and hardware version are determined. The service is designed to maintain multiple repositories, usually one per LRU in addition to some vehicle function repositories. Some fields present in all LRU repositories would include values for connection status, software versions, and current fault status. As with all other aspects of DASH, these repositories are completely configurable and individual repositories and fields can be added or removed via the vehicle configuration at the vehicle level or at the hardware specific LRU level.

For even more flexibility in how the repository functions, each field can be assigned a single pre-trigger and multiple post-triggers.

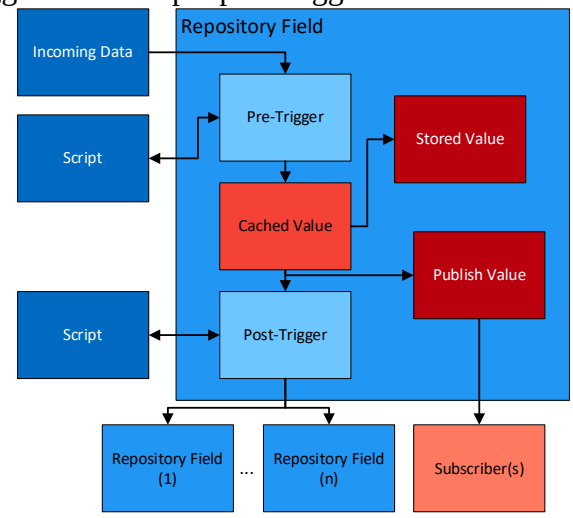

Figure 3

As shown in Figure 3, the pre-trigger is used to manipulate incoming data before being placed into a repository field in the DASH server. After the pre-trigger script is completed and the value returned, it is cached for post-triggers and publishing, as well as for storage in the repository data structure. Pre-triggers are especially useful when an LRU sends data that may be in engineering units to DASH. The pre-trigger allows DASH to automatically convert the incoming data to usable and maintainer understandable units.

Each post-trigger associated with a field is applied to the incoming data that has been cached in the DASH server after the pre-trigger has been applied. The other key difference is that there can be more than one post-trigger, as opposed to only one pre-trigger. This becomes especially useful when an LRU sends DASH a byte array of different data fields. The repository service facilitates the automatic parsing of each sub-field from the byte array and writing them into their appropriate repository field using a post-trigger. Note that these writes into sub-fields in the repository can come with their own pre- and post-triggers, further increasing the flexibility the engineers have when implementing their vehicle specific repository structures. Another example of the utility these triggers offer would be using a post-trigger on a field indicating an LRU's connection status to DASH. This trigger can be used to trigger a script that sends a request to the LRU for its software information, or any other information required to identify or otherwise negotiate the connection.

The repository service will also expose another fundamental piece of the DASH architecture, the publish-subscribe interface. Through the repository interface, defined for both C++ modules and Python scripts, components of DASH within the pyFramework will be able to subscribe to updates and changes in individual fields within the repository. This subscription interface not only allows the subscriber to select on what kind of field updates trigger its subscription, but also allow automatic conversion to any DASH supported units in the returned value. The ability to subscribe to fields extends to all scripts defined in a vehicle specific configuration as well. For instance, on the M109A7/M992A3 FoV, DASH does not currently communicate directly on the CAN bus, but rather through three gateway LRUs over Ethernet/TCP.

The algorithm to select a gateway through which to receive CAN bus messages is defined in the vehicle specific configuration using subscriptions to the three LRUs connected fields in the repository. This extends DASH's functionality in a vehicle specific way without affecting the Core DASH application allowing a vehicle specific engineering team to tailor DASH to their vehicle without the need for support from the DASH engineering team. Additionally, if the vehicle was ever re-architected to support direct DASH communication on the CAN bus, the DASH vehicle configuration is simply adjusted to add a CAN interface configuration and delete the triggers activating the gateway selection script.

3.3. Data Processing Service

The Data Processing Service's (DPS) function is to provide an interface for Python scripts defined both in the DASH application and in the vehicle configuration to be executed on a schedule or on demand via other services or scripts. Every service that is created in the DASH application defines a python extension which acts as an interface to Python scripts. This allows any Python script running within DASH to import a service and interact with its functionality and to potentially facilitate the logic required to bridge communication between two or more services.

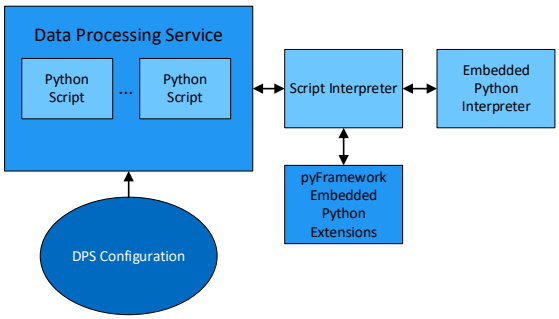

Figure 4

As an example, a script may analyze data from the Repository Service, produce results, and send the results back to the Repository service as an updated field value which may then be published to subscribers for further processing or updating of the client's display to the user. A script may also utilize the communication service to send messages to an LRU after a specific field has been triggered in the Repository service.

3.4. PPMx Data Logging Service

Recent priorities within vehicle fleets have shifted to have the need for not only periodic maintenance, but also Prognostics and Predictive Maintenance (PPMx) or Condition-Based Maintenance (CBM+). In fact, CBM+ is required as a proactive maintenance strategy for cost-effective lifecycle sustainment of military weapon systems.[3] The groundwork for achieving this in a fleet of vehicles is a robust collection of data from a large sample of vehicles over a period. To facilitate this data collection, the Core DASH application provides the PPMx Data Logging service. This service leverages the Repository service and the publish subscribe interface to receive value changes for the logged values according to the PPMx configuration. In addition to containing the repository fields to monitor, the configuration specifies channel rates, type, and metadata, along with vehicle metadata required for later analysis and algorithm development and execution. The PPMx service will aggregate all the repository values into appropriate groups based on the configuration and write them to an Army Bulk CBM+ Data (ABCD) formatted file in the Common Data Format (CDF) file type. The service will also handle writing all necessary ABCD format metadata creating a completed log that can be offloaded and consumed without any additional post processing. When paired with the DASH fault history log that is offloaded at the same time, the log files can be used to effectively analyze the complete vehicle status individually and at a fleet level.

The service will handle all its own file management as well; allowing for a predetermined number of log files to be preserved before rolling over older files with newer data, thus ensuring memory can be allocated between system

applications properly and limiting DASH on hardware where memory is scarce.

Over time, as new PPMx file formats are developed and implemented, the PPMx Service can have the CDF writing interface plugin exchanged with a vehicle specific plugin writing to the desired file output type with the appropriate internal structure and metadata to continue to offer the logged data without the need for post processing.

An example of the structure of the PPMx service can be seen in Figure 5. At startup, the service will determine the vehicle serial number it is currently executing on to load the appropriate vehicle metadata for the output files. The service will then begin its subscription to the necessary fields and will write placeholder data until an LRU connection is established and real LRU sensor values are received.

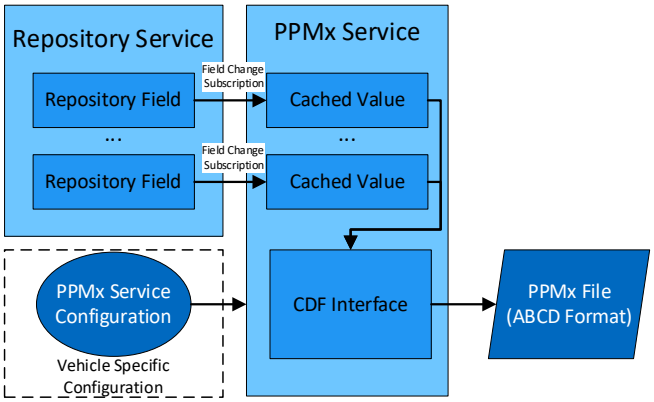

Figure 5

3.5. Network IO Manager Service

The function of the Network I/O Manager Service (DASH-NETIO) is to manage Ethernet and CAN data message traffic between DASH and connected LRUs. Ethernet messages are sent and received over TCP/IP connections with the LRUs, using a message format based on the Ethernet plugin used. On the M109A7, a DASH XML plugin is used, but this can be easily swapped for a JSON or Protocol Buffer based interface, for example. CAN messages can be sent and received directly over a CAN bus, or alternatively over an Ethernet connection with a gateway LRU that communicates on a CAN bus as is the case on the M109A7 FOV.

The DASH-NETIO service contains a list of the configured LRUs, each with the configuration and interface information needed to communicate with that LRU. One LRU can have any combination of defined interfaces, an example being both a CAN and an Ethernet interface. The physical connections and transmit/receive functionality are implemented using Network Interface Modules. These modules are loaded by DASH-NETIO at startup, based on the configuration file.

The DASH-NETIO component loads an XML configuration file on startup that contains LRU configuration information, such as: the expected LRUs and vehicle type identification, LRU Interface information, such as timeout values, fault codes for communication faults and addressing information. The configuration file also contains the list of Network Interface modules, along with the configuration data required by each one. The DASH-NETIO can be configured to only load the interfaces needed by the specific vehicle and can change device driver based on the current operating system.

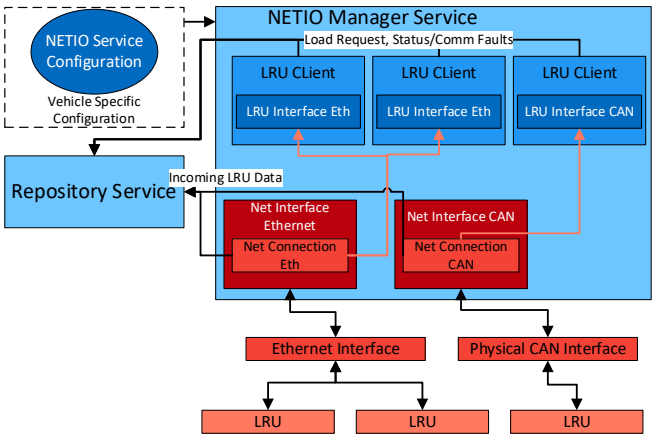

Figure 6

The DASH-NETIO contains a set of LRU clients, one for each LRU defined in its configuration as part of the vehicle configuration. Each of these instances will maintain the required status of the

individual LRU for reporting to the repository associate with the LRU. Within the LRU client, is an LRU interface which contains the parameters that define a particular logical network connection such as Ethernet or CAN.

The status of the connection to the LRU will also be maintained here and propagated to the Repository service as a field within the LRUs repository. Finally, the net interface modules are created based on the network interfaces that are defined, such as Ethernet, CAN, MIL-STD-1553, etc. These modules provide the interface to the actual hardware and manage the connections. Since these are created at runtime, the DASH-NETIO can be tailored to an individual vehicle's interfaces, keeping the instance of DASH as lean and efficient as possible.

3.6. DASH Client

To display current vehicle status and facilitate fault troubleshooting, the DASH application contains a client application that can be run on a wide variety of hardware and operating systems. As is the case with the rest of the DASH application, the client is completely configurable with a configuration being supplied after its initial connection to the server portion of the application.

Figure 7

This configuration is used to supply the client with the appropriate values for its user interface look and feel. Figures 7 and 8 both show two DASH clients running from the same source code. However, their supplied configurations are different, leading to a distinct look while still offering the same functionality and experience to operators and maintainers who are already familiar with DASH.

Figure 7 shows the DASH client used on the M109A7 / M992A3 FoV, which includes a black background with white text and yellow secondary color. The DASH client also displays the current fault count and system time in its top status bar.

Figure 8 shows the DASH client as adapted for the M2A4 Bradley vehicle, which utilizes a grey background with black text and a blue secondary color. This scheme better matches the rest of the Bradley graphic interface software to make DASH blend seamlessly into the tactical system.

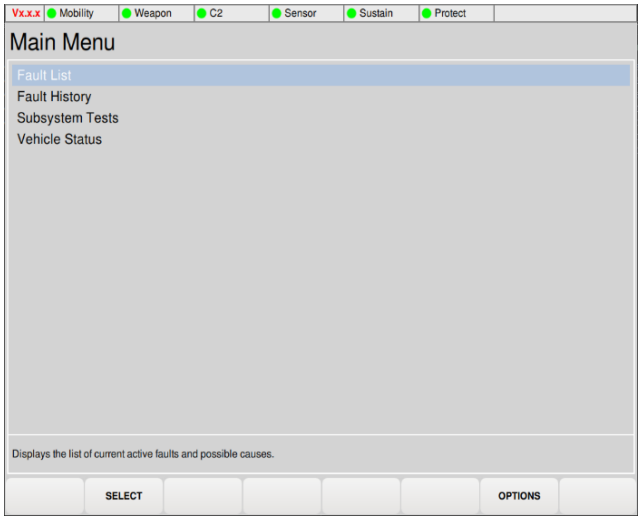

Figure 8

Another distinct difference is that the Bradley implementation of the DASH client does not show a fault count or system time in its status bar. This is because when DASH was first planned for integration into the Bradley software, it was embedded within a parent application which handles window management and already contained a status bar with a fault count indicator

and system time. DASH's flexibility in configuring its display, allowed engineering team to prevent redundant information from being shown in this instance.

Not only is the look and feel configurable, but the data shown to the user can also be changed per project and even per vehicle within a project. Since DASH is aware of which vehicle it is currently running on, the menu can be configured to show vehicle specific options, such as weapon calibration on an M109A7, while hiding them when running the same configuration on a M992A3. This change in menu content does not even require a restart of the application and is sent as a message to the client as part of the negotiation with the server. On the M109A7 / M992A3 DASH can maintain one configuration, allowing hard drives or smart display units to be swapped between vehicles, and the menu is dynamically updated with the correct items for the current vehicle.

Since the DASH client is written using the cross platform Qt framework, all the files used to describe the GUI are written in QML/QtQuick and stored in a Qt resource file that is embedded into the application executable at build time. However, this too can be overridden or added to at runtime from a resource file sent from the server to the client. For instance, the default fault list page in the DASH client can be seen in figure 9.

This page could be replaced with a newly designed QML page in a resource file sent over at runtime simply by editing the menu configuration file for the client to point to the new page. Engineers can completely customize the functionality of DASH in this way to limit or add to the features of the application to suit their vehicle's needs. At runtime, the DASH server will select the appropriate configuration for the connecting client, including any resource files.

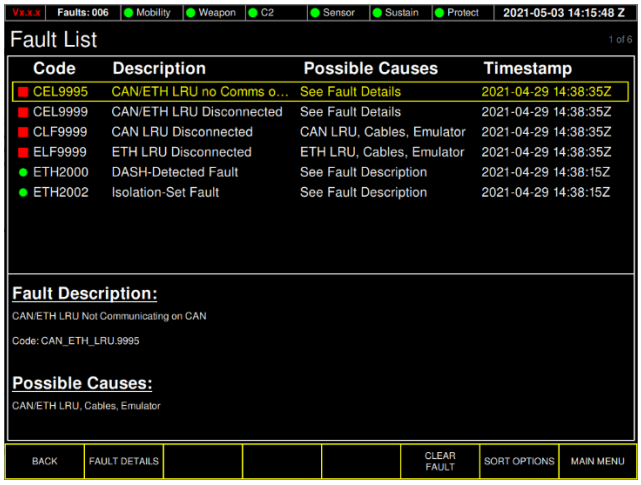

Figure 9

These resource files can be transmitted over the network to the client which is able to dynamically load it. Once this resource file is loaded, any resources it contains can be used by the client including new QML pages, icons, support files, etc.

Finally, the DASH server supports the ability to send an individual configuration to each client that connects, based on the IP address the client connects from, or how the client is configured to identify itself. With this flexibility, the on-vehicle DASH clients could be provided with a production configuration, but clients that connect from an engineer's development machine could be configured to provide application debug information, special screens to run tests, or additional screens showing detailed status of vehicle LRUs or communication buses, to make vehicle integration more efficient and less error prone.

3.7. DASH Diagnostic Procedures

Reporting each LRU's status and fault conditions is a piece of vehicle health monitoring, but without the ability to isolate faults to a single root cause, this functionality cannot fill the entirety of a vehicle's diagnostic needs. The DASH application gives engineers a palette of Visio shapes that perform specific actions and can be linked together in a flowchart used to guide a maintainer through

diagnosing and potentially isolating a fault to a single point of failure. This Visio document is then converted to XML through a DASH translator and loaded into the DASH server as part of the vehicle specific configuration. Form within the configuration, a procedure can be linked to individual fault codes to function as a user launchable way to isolate the fault code or to guide the user through verifying a repair if the fault code is not cleared by an LRU automatically.

The procedures can also be used as a DASH background procedure. These special procedures are loaded at startup and scanned for repository values in the shapes within the procedure. When a repository value is found, the background procedure will be subscribed to the corresponding field and will be executed whenever the field changes. Since a procedure can subscribe to any number of fields, complex logic can be executed automatically allowing engineers to dynamically hide multiple faults to set a root cause fault and clearly indicate a failure to the maintainer. [4]

3.8. External Maintenance Interface

After running a procedure, DASH may determine that the fault is unable to be isolated simply via non-intrusive testing. In this case, a diagnostics engineer can design a DASH procedure to set a link to an external diagnostic procedure associated with the fault code by using the "Set Maintenance Link" shape within a procedure. This will internally assign the link value to the DASH fault code and present a new softkey on the DASH fault list allowing a maintainer to advertise the link to an external diagnostic application. The advertisement of this link is done through the "External Maintenance Interface" which is an Ethernet TCP interface that supports a JSON messaging structure for external applications to communicate with DASH. The purpose of this interface is to allow the external application which would guide the user through more intrusive testing, to query DASH for LRU sensor values, subscribe to a field in the DASH repository, or leverage DASH's ability to command an LRU to execute an Initiated Built in Test (IBIT). After completing the necessary testing, this interface can be used to clear fault codes that were associated with the issue and to set a log message indicating the final status of the procedure.

3.9. Program Management Considerations

Creating a software item that can be deployed on a wide variety of systems causes unique program and configuration management concerns. Different platforms and programs have different requirements and controls that can have an impact on the number and types of documentation and processes that must be followed. In order to effectively meet these requirements while also remaining flexible, DASH uses a document hierarchy that allows for supplemental documents to be added as necessary. There is a core set of DASH documentation, including a Software Requirements Specification (SRS), Interface Requirements Specification (IRS), Software Design Description (SDD) and Interface Design Description (IDD) that can be used as a starting point and be supplemented with requirements and design details for a specific target system. For example, the DASH SRS includes all of the requirements supported by the core DASH application. A specific platform may have additional requirements that need to be included in a requirements document, but are allocated to that platform's configuration. In this instance, a supplemental SRS will be written for that platform that specifies the additional requirements that will be used on that implementation of DASH. Similarly, if a specific platform intentionally chooses to not implement a requirement included in the core DASH application, the supplemental SRS can include a statement that certain higher-level requirements are not included on the specific system. This approach can be applied to other documents as well.

The DEVCOM AC SHIFT Division has also developed a series of documents that can serve as an aid in creating or maintaining DASH

configurations on additional systems. Included in this document set are a "Developer's Implementation Guide for the DASH Application Program" and an "IDD for the DASH System Diagnostic Procedure Shape and XML Format." Together, these documents allow system OEMs and other developers to quickly develop a vehicle-specific DASH configuration that can be used with the latest released version of the core DASH application.

4. PLANNED FUTURE WORK

The DEVCOM AC SHIFT Division has partnered with several Program and Product Managers within Program Executive Office (PEO) Ground Combat Systems (GCS) to create a DASH solution on a number of systems. As mentioned earlier, DASH is currently fielded to the M109A7/M992A3 FoV and software sustainment and maintenance efforts for these platforms are ongoing. DASH is also fielded as the embedded diagnostics system within the overall software suite included with Mortar Fire Control Systems (MFCS). The M2A4 Bradley Infantry Fighting Vehicle (IFV) is currently transitioning its diagnostic solution to DASH. The DEVCOM AC SHIFT Division continues to support these efforts, and also performs traditional software sustainment and maintenance activity to keep DASH and its inputs as up-to-date as possible.

At the forefront of these efforts is the inclusion of Deep Learning, utilizing artificial intelligence, as either a service within DASH or as a complementary application running concurrently with DASH, sending information through the DASH-NETIO interface. This module would, in theory, accept an updated vehicle model created at a remote location as part of a system's automated log retrieval and collection process, allowing each vehicle to remain in sync with the newest data for determining the useful life of components. The end goal of this effort would be to allow DASH to identify operational and maintenance tasks prior to a system component failing or fault condition being realized. If successful, this would enable a true prognostic solution across vehicle and weapon system fleets.

A current effort in the prototype phase explores the use of integrating DASH with other established SHIFT technologies to create a dedicated PPMx data recorder and wireless log off-loader. The goal of this effort is to leverage DASH's robust communication, data management, and logging capabilities paired with a secure, automatic, wireless offload application within one ruggedized LRU. This LRU could then be configured and mounted on any vehicle in order to implement PPMx logging easily and efficiently. The primary benefit of wireless automatic offloading is that it reduces the likelihood of lost logs due to negligence and eliminates the need to train users how to operate new hardware or perform new periodic maintenance tasks.

5. CONCLUSION

With each new DoD vehicle or weapon system that is developed, new or changed communication standards, and hardware that implements these standards, will be introduced into fleets. Each of these systems will need a means of addressing faults and performing diagnostics and maintenance on this hardware and will require an adaptable system to communicate on these standards. Due to the wide variety of operating systems, computing resources, and the infinite customizability of the DASH Server and Client through VCP, DASH would be adaptable to fill maintenance requirements. As DASH gains time in the field and is placed on more vehicle platforms, program and product managers would realize reduced maintainer training burdens by having a common diagnostic system that is already in use on several vehicle platforms while simultaneously reducing development costs by utilizing a modern and intuitive diagnostic development process. While DASH may not be deployed to every vehicle platform, it remains an organic maintenance option for system developers who are seeking a diagnostic solution for in development vehicles, or current

platforms looking to upgrade to a modern diagnostic solution. DASH is mature and field-proven diagnostics solution that provides commonality with every other vehicle that implements it to the benefit of system maintainers.

6. REFERENCES

[1] B. D. Coryell, “Performance-based Logistics, Contractor Logistics Support, and Stryker,” Ph.D. Thesis, US Army Command and General Staff College, Fort Leavenworth, Kansas, Jun. 2007.

[2] P. H. Porter, “Organic Versus Contractor Logistics Support for Depot-Level Repair: Factors That Drive Sub-Optimal Decisions,” Research Report, Air War College, Montgomery, Alabama, Feb. 2016.

[3] DoD Instruction (DoDI) 4151.22, “Condition-Based Maintenance Plus for Materiel Maintenance,” 14 August 2020.

[4] D. A. Tagliente, M. D. Lospinuso and F. DiRosa, "Dynamic Fault Monitoring and Fault-Based Decision Making in Vehicle Health Management Systems," 2017 IEEE AUTOTESTCON, 2017, pp. 1-5.