ONE FOR ALL

THE SOLUTION TO SERVICING COMPLEX VEHICLE SYSTEMS – EACH OF WHICH MAY HAVE ITS OWN ECU – LIES IN STANDARDISING COMMUNICATION AND ENABLING TECHNICIANS TO SUPPORT A MACHINE WITH JUST ONE DIAGNOSTIC TOOL

The complexity of agricultural and other off-highway vehicles continues to increase, enabling them to accomplish more tasks and improve efficiency. Very often this added complexity is the product of an increasing number of subsystems in the main vehicle. Unfortunately, as the complexity of these vehicles increases, so too do their electronic control functions. This poses a growing challenge for service technicians, who provide support for a machine throughout its life. One way to facilitate this task is for the vehicle OEM to create a diagnostic tool capable of carrying out all the diagnostic subfunctions related to the various subsystems on the machine.

To integrate the numerous subsystem diagnostics into a single tool, a systematic approach is taken. The key is the use of standardised communication data formats and interfaces. The methodology for the implementation of these interfaces will be explored here through different use cases, which serve as examples of real-life issues that machine manufacturers face.

One for all or all for one

The first use case deals with a vehicle manufacturer who obtains one kind of subsystem from several suppliers. For example, a manufacturer of agricultural tractors might have a range of tractors of varying sizes and capabilities that use engines from different suppliers. In this example, the engines represent one type of subsystem.

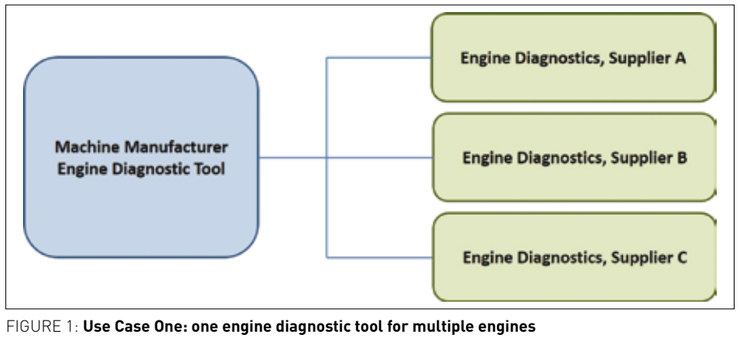

FIGURE 1: **Use Case One: one engine diagnostic tool for multiple engines**

The goal of the manufacturer should be to create one diagnostic tool that will be the same to the end user (technician) for every machine that is equipped with this type of subsystem, irrespective of the supplier. In this example, the OEM wants every vehicle to have the same engine diagnostic tool independent of the engine supplier, as illustrated in Figure 1.

Each of the engines contains a set of diagnostic information that is common among the suppliers, brands and models. This is the information that will used by the engine diagnostic tool created by the tractor manufacturer. This concept can be further generalised – each type of subsystem contains a set of information used for diagnostics that is common among the suppliers. Therefore, in theory, a general diagnostic tool for each type of subsystem is possible regardless of the supplier.

The second use case deals with an OEM who requires different components from different suppliers. For example, the tractor manufacturer might use an engine from supplier A, a joystick from supplier B and a steering column from supplier C. Each of these subsystems is controlled by its own ECU, and each contains its own set of diagnostic subfunctions. The goal for the manufacturer is therefore to integrate all the diagnostic subfunctions into one diagnostic tool so that the end user will not be required to access each subsystem individually using a different tool. This concept is illustrated in Figure 2.

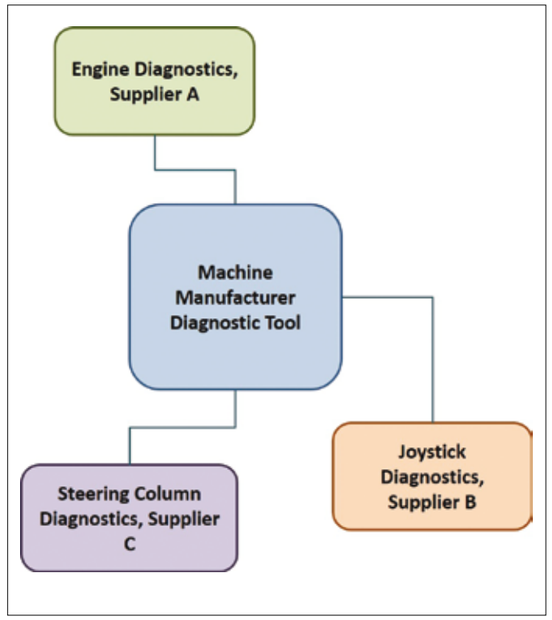

FIGURE 2: **Use Case Two: one diagnostic tool for different subsystems**

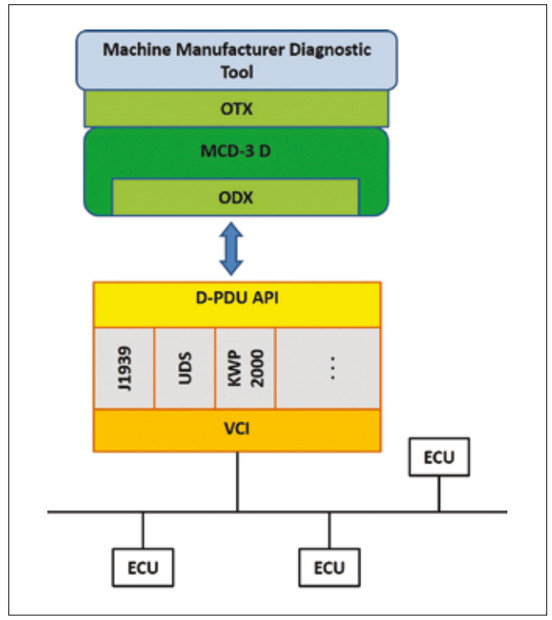

FIGURE 3: **Structure for communication of diagnostic information using standards**

Best case scenario

These two use cases illustrate real challenges faced by off-highway manufacturers today. This is especially true as the complexity of industrial vehicles increases.

One commonality in the two cases is the manufacturer's desire to combine all diagnostic functions into a single tool. This tool should be able to handle diagnostic subfunctions related to a single subsystem regardless of its supplier, as well as the diagnostic subfunctions of the various subsystems in one vehicle.

The key to achieving this goal is the use of standardised communication data formats and interfaces. These include D-PDU API, ODX and OTX, as illustrated in Figure 3 (above).

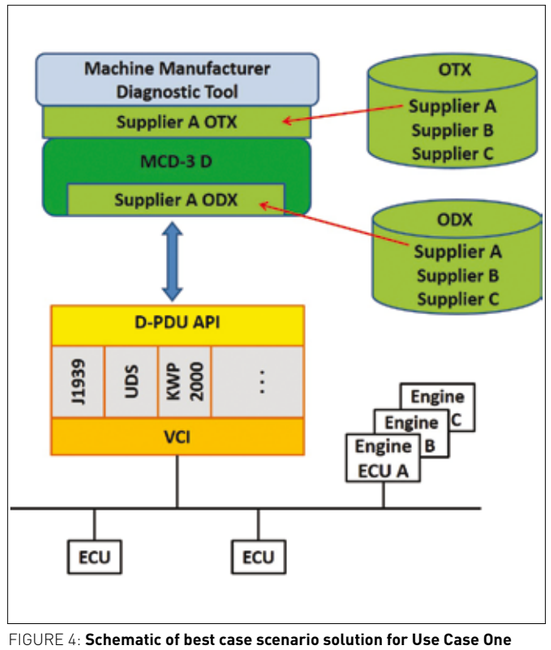

FIGURE 4: **Schematic of best case scenario solution for Use Case One**

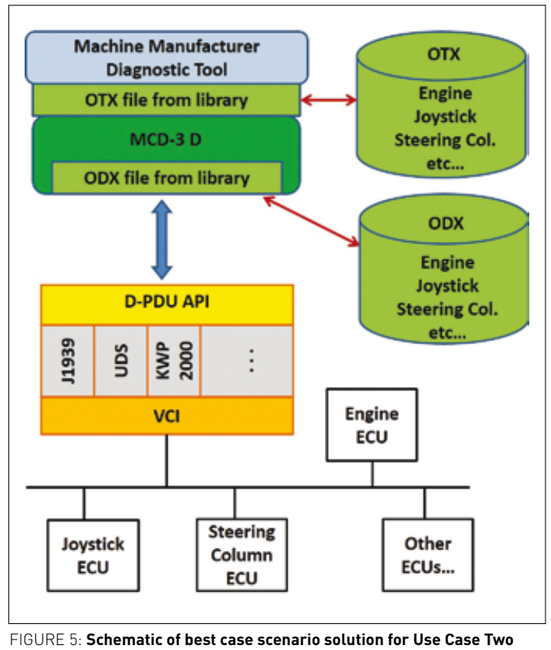

FIGURE 5: **Schematic of best case scenario solution for Use Case Two**

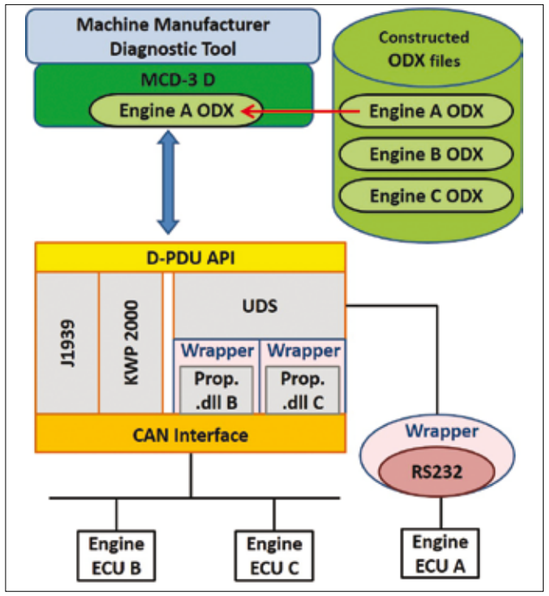

FIGURE 6: **Schematic of real world solution for Use Case One**

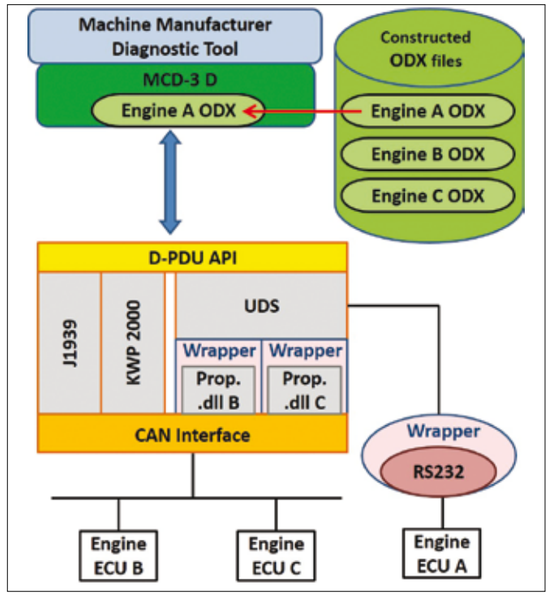

FIGURE 7: **Schematic of real world solution for Use Case Two**

The Diagnostic Protocol Unit application programming interface (D-PDU API), as specified in ISO 22900, is a generic software interface that can provide plug-and-play functionality for various communication protocols. This interface connects all the diagnostic information gathered through the protocols to a standard D-server.[1] This server then uses information regarding the data descriptions found in the Open Diagnostic Data Exchange (ODX) files and provides 'translated' information to the diagnostic application.

The ODX specification is used to describe and exchange vehicle and ECU diagnostic information, such as diagnostic trouble codes, identification data, input/output parameters and communication parameters. ODX is a machine-readable data format that is based on XML, and is independent of specific vehicle diagnostic protocols.[2,3]

Finally, Open Test Sequence Exchange (OTX) files are used to connect the raw data to the diagnostic application. The OTX format, described in ISO 13209, is an XML-based exchange format for diagnostic test sequences. The structure of these sequences is described in schemas that make it easy to follow, understand, and implement.[4]

Using these standards, a modular software architecture is created to communicate diagnostic information from the lowest level all the way up to the diagnostic application surface. This structure enables the integration of diagnostic information from different subsystems.

The sequence of operation with this model is as follows. Information requested by the diagnostic tool is matched to a specific diagnostic sequence (OTX). The test sequence then passes through the D-server where the required information is obtained from the ODX file(s). Next, the transformed request passes down into the D-PDU API layer, where it is formatted to communicate to the correct ECU on the bus through the appropriate protocol.

A response from an ECU is similarly communicated in reverse order through the layers up to the application surface.

Library of progress

Applying this methodology to the use cases yields very similar models. In the first use case, the manufacturer wants to integrate the diagnostic functions of one type of subsystem obtained from various suppliers. Figure 4 illustrates the solution to this use case. In this model, the suppliers of the engines used by the manufacturer (suppliers A, B and C) provide the corresponding OTX and ODX files for their engine ECUs. These files are placed in libraries as part of the overall diagnostic software architecture. They are then retrieved from the libraries as needed, depending on which engine is being used. In this way the OEM can have a single software architecture that can handle multiple variations of the same subsystem.

In the second use case, the manufacturer wants to be able to integrate the diagnostic subfunctions which correspond to the different subsystems into a single diagnostic tool. For example, the vehicle might include an engine, a joystick and a steering column from different suppliers, all equipped with their respective ECUs.

Similar to the first use case solution, the solution to this use case involves the suppliers of the subsystems providing the corresponding ODX and OTX files, as illustrated in Figure 5.

The ODX and OTX files are again stored in libraries that become part of the software architecture of the diagnostic tool. From these libraries, the ODX and OTX files associated with each ECU are retrieved when required. The use of libraries in this use case builds on the concept described in the first solution, in that these libraries not only contain ODX and OTX files related to different subsystems, but also contain the ODX and OTX files for the variations of each subsystem. In this way, the OEM is able to create one diagnostic tool that can be used for the entire machine. For example, by using ODX and OTX files provided by the suppliers, a manufacturer of agricultural tractors is able to create a single diagnostic tool that can be used across its entire range of tractors.

Real world case

In the real world, however, suppliers do not always provide ODX and OTX files, for several reasons. Some suppliers do not use these formats because they do not want the OEM to have access to their proprietary knowledge; instead, they use their own proprietary communication protocol DLLs, which are not supported by the D-PDU API. Also some subsystems might use specialized communication formats, such as RS232 instead of CAN.

Furthermore, suppliers sometimes fail to use standards because they run out of time during the development process or because they prefer to use their own methods of communication. And finally, suppliers are interested in preserving the quality of their products as well as their corresponding diagnostics. They are concerned that, by using standards, the machine manufacturer will create diagnostic systems that are not consistent with their own levels of quality.

Regardless of the reason, when suppliers do not provide ODX and OTX files, the task of subsystem integration will become more difficult. The solution involves the OEM constructing ODX files from the data descriptions provided by the supplier, and by constructing 'wrappers', or specialized interfaces, that enable the proprietary or non-standard communication formats to interface with standard protocols such as UDS. Hardcoded within these wrappers is information regarding diagnostic test routines that can be used with the built-in UDS function routine controls.

To demonstrate this methodology, Use Case One is modified to encompass two engine ECUs that communicate using proprietary protocol DLLs, and one that communicates through a serial interface, as illustrated in Figure 6.

The ODX files are created by the OEM from the description data provided by the suppliers of the different engines. Wrappers are additionally created for each proprietary protocol .dll, as well as for the RS232 interface, which will allow for these non-standard formats to interface with a standard protocol – i.e. UDS. Furthermore, licensing and user-level management mechanisms can be integrated into the wrapper. This enables the suppliers to set licensing options that can help to safeguard their subsystem quality standards.

The solution for real-world Use Case Two, as shown in Figure 7, further builds on the same concept from Use Case One. In this scenario, the OEM's diagnostic tool has to integrate the subfunctions of a variety of subsystems without having been provided with ODX or OTX files. Some of the ECUs might use proprietary protocol DLLs, while others might use other forms of communication such as RS232.

The solution for their integration is based on the same principles as in the solution for Use Case One. ODX files are constructed from the information provided for each ECU and stored in an ODX library. Then wrappers that include hardcoded information regarding diagnostic test routines are created to interface proprietary protocol DLLs and other communication formats with the UDS standard protocol.

Standardising communication

The real key to the integration of different subfunctions into one diagnostic tool is to use standardised communication interfaces and data formats. With these standards, a modular architecture can be created that allows the manufacturer to incorporate subsystems from multiple suppliers into one diagnostic application.

This architecture is capable of being adapted to different situations such that the OEM is not forced to completely rework the solution each time a new ECU is added. Not only is this system much easier for the manufacturer to work with, but it is also robust and futureproof. Suppliers will also benefit from such a modular architecture as it enables them to hide their knowledge; provide proven technologies; and even integrate appropriate licence and user management mechanisms to safeguard their quality standards. **iVT**

Juan Aguilar and Gerd Bottenbruch are business development & application engineer and software group leader at Sontheim Industrie Elektronik

References:
[1] ISO 22900-2
[2] http://www.asam.net/nc/de/home/standards/standard-detail.html?tx_rbwbmasamstandards_pi1%5BshowUid%5D=523&start=
[3] http://automotiveiq.wordpress.com/2011/06/03/automotive-diagnostic-systems-from-obd-to-open-diagnostics-exchange-format-odx/
[4] http://www.otxcentral.com/about-otx/

Configure your own service tool

Built on this concept of standardized communication interfaces and data formats, Sontheim Industrie Elektronik's Modular Diagnostic Tool (MDT) empowers manufacturers of complex systems to create their own, complete, custom-built applications such as vehicle diagnostics, vehicle service, Flash tools, and end-of-line programming. Incorporated into MDT are all the tools necessary for every aspect of diagnostic application design, such as data definition (using ODX), creation of test sequences, and custom graphical user interface design. MDT's modular runtime system architecture is capable of supporting different network types in addition to standardized and proprietary communication protocols. This architecture design allows for very robust yet flexible and futureproof diagnostic systems while maintaining low development times and test efforts. Sontheim also produces various vehicle communication interfaces which synergize seamlessly with MDT to meet all ranges of your diagnostic and communication applications. Whether OEMs require handheld or onboard, wired or wireless, Sontheim is the one-stop-shop for complete, high-performance and reliable electronics!