In typical automotive development processes, not only are diagnostic descriptions and their software implementations handled by different people, in different roles, but they are also supported by different authoring tools. The consistency between the two sets of data must be verified manually, and this process can be both complicated and susceptible to errors. An integrated, model-driven approach to development makes it possible to describe diagnostics within a vehicle E/E system, including the relationships with the implementation software.

In projects related to vehicle diagnostics, the diagnostic specification and software development are, usually, largely decoupled. It may even be true that large parts of the functional aspects of software and other parts of the E/E system are specified long before any consideration is given to diagnostics and the needs of vehicle production-line stations or service workshops. Today, responsibilities for the diagnostic specification and application software development may be distributed among different people, roles and departments. The engineers involved complete their tasks with different tools and manage their work products in isolation from one another. There is no common database, meaning that data is manually worked through the various tools. The exchange of information between the different work areas is conducted informally by emails, meetings, telephone calls or sharing of documents. Often, essential changes are made ad hoc and without standardized data exchange or automatic synchronization.

These observations are not at all surprising when one considers how much the task areas of diagnostic and software development diverge in practice. A successful diagnostic design must consider the entire life-cycle of the system. Outside of the system development area, the diagnostic solution must also support the use cases of vehicle production and customer service areas. Experts in each of the areas view vehicle diagnostics from different perspectives. A key focus for production and customer service experts is on details related to the protocol for parameterizing the various testers in production and customer service. For system developers, on the other hand, the central focus is on the internal design and functionality of the vehicle software. Quite often, the areas consider one another’s needs to only be of passing relevance or even an annoyance.

Nonetheless, dependencies still exist between diagnostics and software! Inconsistencies can result, for example, if datatype definitions in the diagnostic specification do not match those of the implemented application software. The best approach is to be aware of these potential issues early in the configuration process for the basic software components defined in AUTOSAR, such as the Diagnostic Event Manager (DEM) and Diagnostic Communication Manager (DCM). In undesirable cases, this would not be done until
The start-up of an Electronic Control Unit (ECU) or even in the field. The old rule applies here: The later an error is detected, the more expensive and complicated it is to correct. If a single tool could be used to create the description of the application software together with the diagnostic contents, then it would be possible to avoid the mentioned inconsistencies and subsequent problems associated with them. An efficient method for overcoming the described challenges is now offered by the seamless exchange of data between two tools of the Vector product line-up: the model- and database-based tool PREEvision can be used for development of diagnostic and software concepts, and it exchanges all relevant data with CANdelaStudio, the Vector tool for creating and editing formal diagnostic specifications.

Advantages of Model-Based Tools
The advantages of the model-based PREEvision tool are being exploited more and more intensively by many automotive OEMs in their Electrical/Electronic (E/E) development processes. A higher-level E/E data model allows the manufacturers to assure technical consistency between system requirements, the architectural and design levels of the software, network and hardware components, and wire harnesses and geometry. They can describe all knowledge concerning the interrelationships between technical levels and elements in this shared data model (Figure 1).

If a technical change affects multiple levels, the change is immediately visible to all affected developers, and the individual user is automatically notified of the change. This makes it possible to recognize potential conflicts early on so that they can be corrected. Users can reliably track release points and interrelationships between them using versioning mechanisms. In addition, extensive support is available for managing variants. This lets users describe identical parts as well as the differences between different variants in the shared database. Individual variants can then be activated for display in the model view, for analysis and checking, or for data export.

Dependencies Between Diagnostic Modeling and Software Modeling
What dependencies even exist between the ECU software and the diagnostic description? In many authoring tools for diagnostics, an ECU’s diagnostic functions are described from the viewpoint of a diagnostic tester. In such a description, the focus is on the diagnostic functions that external test tools can use over the vehicle’s diagnostic interface (or interfaces). This includes, for example, information on which diagnostic trouble codes (DTC) a tester can read from an ECU’s fault memory and which data identifiers (DID) an ECU provides for reading and writing data. In addition, it describes detailed information that is very significant to what is displayed in the tester, such as datatypes, conversions and physical units of the individual data objects of DIDs and routines.
The diagnostic provisions of the application software must now be implemented via the specific design of software ports (diagnostic ports). For a data object of a DID, this is an individual data element of a diagnostic port. This data element is also given a specific implementation by assigning it a data type with a conversion formula and physical unit. However, in the software these details may be distributed among multiple objects by AUTOSAR-conformant modeling; this differs from the approach taken in many of the usual diagnostic description formats in which they are described by a single object.

For example, a single DID might be used to access several temperatures that are individually transferred by multiple software ports (possibly allocated to different software components) as a corresponding number of data elements. Despite the differences in granularity and description types, the informational content must be consistent. If diagnostics is defined together with the software in a tool, it is possible to describe software and diagnostics so that it is free of redundancies and inconsistencies. The datatypes are then described, as usual, in the software. The diagnostic description may then reference these datatypes; that is, it reuses the same objects. As an alternative, it is also possible to first describe the diagnostic contents and use the shared detailed information for the software implementation.

Along with the need to maintain consistent, shared data descriptions, the diagnostic specifications also result in requirements for the application software itself. For example, if the diagnostic specification defines a routine, there must be a software component in the application that implements this routine. If the diagnostic specification defines a DTC, the application must have monitoring functionality that can determine whether the associated error event has occurred. AUTOSAR sets specific, structural requirements for the implementing software components here. I/O control, for example, requires that the data that can be monitored must be represented as a record type. A routine, on the other hand, requires a client/server interface with start, stop and request result operations. Here too, it helps to define the structures just once and use them for both the software and diagnostics.

Specific Implementation in a Tool Chain

To overcome the described problems, PREEvision has been extended to include a separate abstraction layer for diagnostics. This diagnostic layer makes it possible to describe diagnostic objects system-wide and to link them to the application software. The user can define DiIDs here with their data objects, I/O controls and routines. Certain supplemental properties can also be described for these objects such as identifiers or the diagnostic services that are used. Moreover, relevant parts of event handling can also be modeled according to AUTOSAR. These include objects such as DTCs, DiagnosticEvents and Diagnostic OperationCycles to name just a few. Authoring tools like CANdelaStudio are essential for implementing a specific diagnostic description. A tool chain consisting of PREEvision and CANdelaStudio therefore offers bidirectional data exchange (Figure 2), and it supports various cooperative working models.
Return data flow to PREEvision is also possible. This allows users to take master data from existing diagnostic descriptions and use it as a starting point for their software implementation. A file-based approach was chosen for data exchange which enables exchanges across organizational boundaries. CANdleStudio’s import and export options open-up a broad range of options for working together with suppliers and for coupling to various tool chains. The exchange file contains both the actual diagnostic objects and associated details which are derived from the referenced software. Although the modeling of diagnostic objects is performed system-wide, the exchange file only contains the information for a single ECU. Which diagnostic ports are even available on an ECU will depend on the software modeling and hardware modeling. This information forms the basis for arranging the diagnostic objects for the data exchange (Figure 3).

There is one other aspect which is helpful for the downstream process: Information on the allocation of the diagnostic object to the diagnostic port in the application software is transported via a reference in the data exchange. In a configuration tool for basic software, the integrator no longer needs to make the connections between the ports of the basic software and those of the application software (a process known as port-mapping). The connections can be determined automatically based on the references, and this significantly reduces susceptibility to errors and saves time.

**Assuring Diagnostic Conformity in the Software**

The AUTOSAR standard defines how software components need to be described for diagnostics. If a diagnostic port is being generated, for example, it must be modeled in an AUTOSAR-conformant way. However, the AUTOSAR standard is extensive, complex and not always easy to internalize. The usual practice is to frequently look things up in the standard documents, and implementation of the standard is time-consuming and prone to errors.

Thanks to tool support within PREEvision, this data can now be created more efficiently and at a much higher level of quality: When a diagnostic port is generated directly from the actual diagnostic object, the tool support ensures that this port is generated properly and is AUTOSAR-conformant. If the user makes subsequent changes to the diagnostic port that lead to a deviation from the standard, the tool points this out.
Advantages in the Overall Development Process

Along with describing the software, hardware and communications in a model in PREEvision, diagnostic information for an E/E system is also described. The user manages all data centrally, and diagnostics is viewed system-wide. From the description of software, hardware and communications, an AUTOSAR-based system description is created (Figure 4). From the same model, the diagnostic contents for an ECU are exported, and then they are further refined and supplemented within CANdelaStudio. It is no longer necessary to manually adapt the diagnostic data to the software (or vice-versa). Changes made in CANdelaStudio may be written back to PREEvision by data exchange, and the software is updated automatically.

The AUTOSAR Diagnostic Extract (DEXT) or tester parameterization formats (such as Open Diagnostic-data Exchange [ODX]) are created in CANdelaStudio. Finally, to configure the AUTOSAR basic software, the system description and DEXT are read into tools like Vector’s DaVinci Configurator. Several advantages are realized here:

> There is no need to manually merge the diagnostic objects with the diagnostic ports. References transferred over the entire tool chain enable automatic allocation.
> The diagnostic ports are already modeled to be AUTOSAR-conformant, and they do not require any modifications afterwards.
> Properties of the diagnostic ports match the properties of the diagnostic objects, and no conflicts need to be resolved related to data types or software structures.

This results in lower coordination effort over the entire design process, less manual effort and a reduction in potential sources of error. Naturally, the advantages are not limited to linking diagnostics to the application software. Now, diagnostic data or sequences can also be derived automatically from other model levels in PREEvision, such as test sequences for wiring open-circuits that can be derived directly from the description of the wire harness. PREEvision’s model-based approach offers tremendous potential for the future here, promising to reduce work effort and sources of error even more.

Authors

Claudia Gettkandt
Vector Informatik GmbH
Product Manager PREEvision

Armin Cant
Vector Informatik GmbH
Product Manager CANdelaStudio

Originally published in "Elektronik automotive" magazine
Issue 12/2019 – December 2019