Editor: Christof Ebert ■ Vector Consulting ■ christof.ebert@vector-consulting.de

The Autosar XML Schema and Its Relevance for Autosar Tools

Uwe Honekamp, *Vector Informatik*

Autosar (Automotive Open System Architecture) is an emerging technology in automotive software engineering that contributes to reuse and increased flexibility while preserving interfaces and system-level integrity. The Autosar approach has much to offer software engineers working on embedded systems or model-driven development. In this issue's essay, Uwe Honekamp briefly explains this approach and shows how to practically use models and XML configuration schemes.

I look forward to hearing from readers and prospective column authors about this column and the technologies you want to know more about. —Christof Ebert

Contemporary automotive embedded software, despite sporting a complex structure, usually runs on hardware whose performance and resource utilization are orders of magnitude lower than those of a typical PC. Consequently, this software has extraordinarily high requirements regarding efficiency and a minimized memory footprint.

To meet these requirements, developers commonly generate this software automatically, partly or completely according to specific configuration settings. To do this, they often use model-based development tools and code generators.

Autosar represents the first widespread approach to establish a standard covering virtually all aspects of the development workflow of automotive embedded software. The goal is to ensure the interoperability of development tool implementations by using a standardized XML-based format for information exchange among the tools.

The Autosar XML Schema

To promote interoperability, the Autosar consortium (an international group of automobile manufacturers, suppliers, and tool developers) has defined an XML schema against which all Autosar XML documents are supposed to be validated. The Autosar XML schema covers the formal structural description of relevant modeling artifacts and configuration descriptions exchanged among project partners. Table 1 lists the schema's abstraction levels.

This XML schema results from a complex model transformation based on a UML metamodel. The metamodel contains all the model elements plus annotations that a dedicated software tool interprets to perform the transformation. For example, the metamodel clarifies that a `ComponentType` can have required or provided `PortPrototypes` that in turn are typed by a `PortInterface` (see Figure 1a). Figure 1b shows the XML schema fragment resulting from transformation of the upper two levels of the metamodel elements in Figure 1a.

Some important aspects, however, can't be