OTX IN DER PRAXIS

Prüfabläufe barrierefrei **austauschen**

Komplexität zu beherrschen, ist die Herausforderung der heutigen Zeit. Dies kann nur in einer bereichsübergreifenden Zusammenarbeit gelingen. Durchgängig standardisierte Prozesse sind hierfür Voraussetzung. Standards schaffen Transparenz und Austauschbarkeit. Sie sind Grundlage für die kontinuierliche Verbesserung, die Konsistenz der Leistungserbringung und die Vermeidung von Redundanzen.

Der Standard OTX (Open Test sequence eXchange) nach ISO 13209 ist eine domänenspezifische Sprache (DSL) zur prozesssicheren Beschreibung austauschbarer und ausführbarer Prüflogik im Bereich der Automobilindustrie. Diagnoseabläufe können graphisch erstellt und gleichzeitig so detailliert beschrieben werden, dass dieselbe Prüflogik barrierefrei in beliebigen unterschiedlichen Zielumgebungen ausgeführt werden kann. Der Standard hat die nötige Reife und ist umfangreich genug, um bestehende Lösungen in Entwicklung, Produktion und Werkstatt zu ersetzen. Die Einsatzmöglichkeiten von OTX reichen von der Beschreibung einfacher Funktionstests in der Entwicklung über Inbetriebnahme-Abläufe in der Produktion bis hin zu vollständig generischen Tester-Anwendungen mit geführter Fehlersuche im Kundendienst. OTX ist offen, stabil, plattform- und technologieneutral.

Austauschbarkeit und Standardkonformität

Damit die vielversprechenden Potentiale nicht nur Werbeaussagen sind, sondern auch in der Praxis belastet werden können, reicht der Standard nicht aus. Prozesse müssen angepasst, Tools entwickelt und übergreifendes Denken muss gefördert werden. OTX allein sichert noch keine Austauschbarkeit. Unter Austauschbarkeit versteht man hier,

die unveränderte Verwendung von OTX-Dokumenten in verschiedenen Zielsystemen. Anders gesagt: Es soll dieselbe Prüflogik in unterschiedlichsten Systemen ausführbar sein und funktional zu denselben Ergebnissen führen. Um diese Austauschbarkeit sicherzustellen, müssen die Prüfabläufe (OTX-Dokumente) folgende Kriterien erfüllen:

- Gültigkeit
- Vollständigkeit
- Zielsystemunabhängigkeit

Gültige OTX-Daten müssen syntaktisch korrekt sein – sie entsprechen dem Datenmodell – und verletzen keine kritischen semantischen Checker-Regeln. In den meisten Fällen ist dies kein Problem. Fehler, die hier auftreten sind eher grob und schnell zu finden.

Weiterhin ist ein Prüfablauf dann vollständig, wenn die komplette Prüflogik in OTX vorliegt. Problem: Die Definition, was Prüflogik beinhaltet, ist vom Einsatzfall abhängig. Meinungen können weit auseinander gehen. Letztlich stellt sich die Frage, was man mit OTX erreichen will.

Um dies zu verdeutlichen, zwei Extremfälle: Im Ersten wird aus OTX lediglich eine Funktion in einem externen System aufgerufen, welche die gesamte Prüflogik enthält. Im zweiten Fall liegt der komplette Tester, inkl. HMI, Rollenverwaltung etc. in OTX vor. Der erste Fall ist zwar gültig, jedoch nicht vollständig und damit auch nicht austauschbar. Der zweite Fall ist sowohl gültig als auch vollständig, jedoch kommt man hier an die Grenzen der Ausdrucksfähigkeit von OTX, wofür OTX nicht gemacht wurde. In der Praxis hat sich gezeigt, dass die in OTX gespeicherte Prüflogik in etwa dem fachlichen Wissen eines guten Bauteileverantwortlichen entsprechen sollte.

Wenn die Abgrenzung der Prüflogik klar ist, dann gehört zur Vollständigkeit auch, dass alle in OTX referenzierten Elemente vorhanden und zugreifbar sind. Man könnte beispielsweise eine Prozedur aufrufen, die es in OTX gar nicht gibt, sondern zur Laufzeit irgendwie an die externe Methode einer Prüfstands-Bibliothek gebunden wird.

Zielsystemunabhängigkeit

OTX darf keine zielsystemabhängigen Daten enthalten. Andernfalls ist es zwar gültig, aber nicht austauschbar. Zielsystemabhängigkeit beginnt dort, wo für die Ausführung des Ablaufs spezifische Umgebungen mit spezifischen Implementierungen benötigt werden. Da ein Ablauf immer auf einem Zielsystem ausgeführt wird, gibt es diese Abhängigkeit immer. Die dafür nötigen Konfigurationsdaten dürfen nicht innerhalb, sondern müssen außerhalb der OTX-Dokumente gespeichert werden, siehe OTX-Mapping unten.

Ein Ort, in dem zielsystemabhängige Daten gespeichert werden könnten, sind die so genannten Meta-Daten. In OTX kann man an nahezu allen Elementen Meta-Daten speichern und somit innerhalb des OTX-Dokuments beliebige zusätzliche Daten transportieren, z. B. Freigabeversionen oder Kennzeichnungen von Prozeduren. Diese Daten dürfen aber das Laufzeitverhalten der Prüflogik nicht beeinflussen. Einfacher Test: Nach Löschen aller Meta-Daten, muss der Ablauf weiterhin ausführbar sein und das Laufzeitverhalten darf sich nicht ändern.

Eine weitere potenzielle Tür für Zielsystemabhängigkeit ist die Schnittstelle von OTX zur Außenwelt (Prozedur-Parameter, Context- und Status-Variablen sowie Screen-Parameter etc.). Hier dürfen nur konvertierbare Datentypen wie Boolean, Integer, Float, String, ByteField, List, Map, Enumeration und Structure verwendet werden.

Zusammenfassend kann man sagen, dass eine OTX-Bedatung erst dann wirklich standardkonform nach ISO 13209 ist, wenn sie gültig, vollständig und zielsystemunabhängig ist.

Alle Tools der Firma EMOTIVE erzeugen und verarbeiten ausschließlich standardkonformes OTX. Dies bedeutet nicht, dass es nicht auch sinnvolle Fälle geben kann, in denen gültiges OTX unvollständig und zielsystemabhängig verwendet wird, die Austauschbarkeit über Prozessgrenzen mit unterschiedlichen Tool-Landschaften ist jedoch dann nicht mehr sichergestellt.

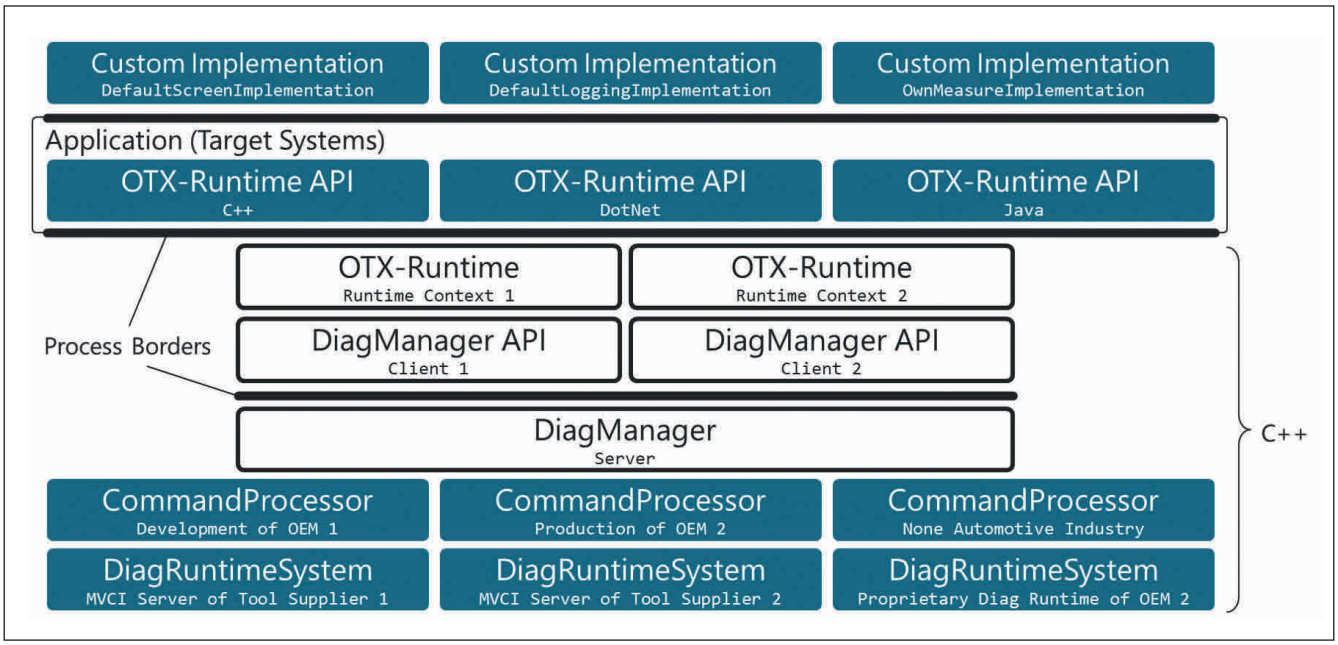

Bild 1: Austauschbare Ebenen der EMOTIVE OTX-Runtime. © emotive

Austauschbarkeit und Toolintegration

Die Firma EMOTIVE hat in den letzten Jahren große Anstrengungen unternommen, diesen Anspruch in die Praxis umzusetzen. So steht eine komplett neue Laufzeitumgebung für OTX (OTX-Runtime) zur Verfügung, die mit der Anforderung entwickelt wurde, Austauschbarkeit auf allen sinnvollen Ebenen zu ermöglichen, siehe Bild 1.

Um OTX auszuführen, muss die API der OTX-Runtime in eine Anwendung integriert werden. Über die API können OTX-Projekte (PTX-Dateien) geladen, deren Datenstruktur ermittelt (Browsing) und Prozeduren gestartet werden. Die OTX-Runtime API liegt in den drei Technologien C++, DotNet und Java vor und ist somit praktisch innerhalb jeder existierenden Technologie einsetzbar. Beim Starten einer Prozedur wird ein Runtime-Kontext erzeugt. Er ist das Herzstück der Laufzeitumgebung. Dort findet die eigentliche Ausführung von OTX statt. Er und alle darunter liegenden Schichten sind in nativem C++ geschrieben und somit performant und Ressourcen schonend in beliebigen Zielarchitekturen z. B. Desktop, Web oder Embedded einsetzbar.

Diagnose-Kommunikation

Für die Diagnose-Kommunikation arbeitet die OTX-Runtime auf dem auch stand-alone verfügbaren DiagManager. Er hat die Aufgabe, alle Diagnose relevanten OTX-Befehle in Befehle eines spezifischen Diagnose-Laufzeitsystems zu übersetzen. Dabei kann er als Server arbeiten, so dass eine parallele Diagnose aus beliebigen Prozessen oder Anwendungen möglich ist. Der Server serialisiert die Befehle der Clients und leitet sie an den Command-Processor weiter. Der Command-Processor hat die Aufgabe die Diagnose-Befehle zu optimieren und zu priorisieren. Beispielsweise findet im Command-Processor das Verwalten der offenen Diagnose-Kanäle statt. Der Command-Processor ist austauschbar. Somit könnte der Anwender seine eigene, spezifische Verwaltung der Kommunikationskanäle oder Diagnose-Dienste implementieren. Der Command-Processor sendet seine Befehle an das Diag-Runtime-System. Im Diag-Runtime-System findet die eigentliche Übersetzung von OTX auf die spezifischen Methoden eines Diagnose-Laufzeitsystems statt, z. B. nach ISO 22900–3 (MVCI). Auch diese Schicht ist komplett austauschbar. Der Anwender kann hier selbständig eine Anbindung an sein eigenes proprietäres Diagnose-Laufzeitsystem vornehmen.

OTX macht aber nicht nur Diagnosekommunikation. In OTX gibt es verschiedene Extensions zur Interaktion mit anderen externen Systemen:

- CommonDialogs
- Context und StateVariable
- ExternalServiceProvider
- HMI
- i18n
- Logging
- Measure
- SQL
- TestResultHandling

Die Implementierung dieser Extensions befindet sich nicht innerhalb der OTX-Runtime, sondern in einer externen Implementierung, der so genannten Custom-Implementierung. Hierfür werden an der OTX-Runtime API Schnittstellen bereitgestellt, die der Anwender selbst implementieren kann. EMOTIVE liefert für diese Schnittstellen Standard-Implementierungen aus. Es ist somit möglich, OTX in beliebige Zielsysteme nahtlos zu integrieren und auszuführen. Derselbe OTX-Ablauf kann innerhalb einer Web-Applikation mit einer HTML-Screen Anbindung ausgeführt werden oder im Fahrzeug im Infotainmentsystem.

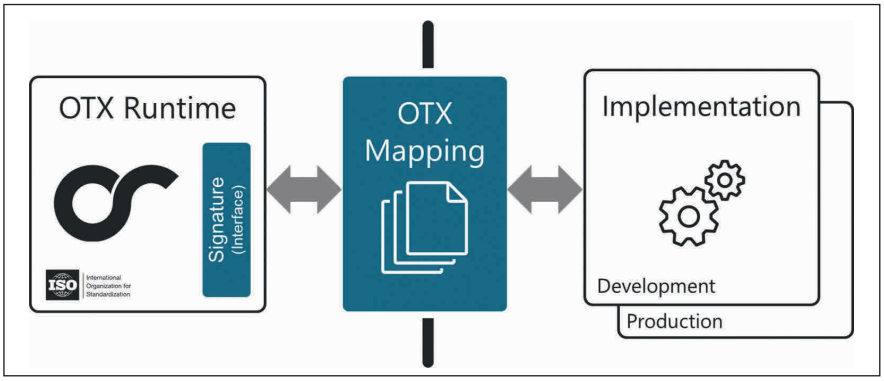

Bild 2: Funktionale Austauschbarkeit (OTX-Mapping). © emotive

Funktionale Austauschbarkeit

Damit OTX-Prüflogik auf einem Zielsystem ausgeführt werden kann, müssen externe Aufrufe von OTX an spezifische Funktionen im Zielsystem gebunden werden. Dies wird bei EMOTIVE als OTX-Mapping bezeichnet, siehe Bild 2. Alle für ein Zielsystem notwendigen Mapping-Informationen werden dabei in einer Datei gespeichert. Allein durch den Austausch dieser Datei kann dieselbe OTX-Prüflogik in verschiedenen Zielumgebungen laufen.

Beim OTX-Mapping werden Screens (Screens entsprechen in OTX einer so genannten Screen-Signatur und deren Parameter) beispielsweise an die passenden Klassen eines DotNet-Assemblies gebunden, die ein Fenster in WPF-Technologie darstellt; oder in einem anderen Fall an ein HTML-Template gebunden, das eine Seite im Internet-Browser beschreibt.

Fazit

Bei konsequenter Beachtung der Standardkonformität von der Erstellung bis zur Ausführung ist OTX wie kaum ein anderer Standard in der Lage, qualitätsgesichertes Prüfwissen über Prozess- und Toolgrenzen auszutauschen. OTX kann sicherstellen, dass dieselbe unveränderte Prüflogik zu jeder Zeit in beliebigen Zielsystemen ausführbar ist und zu denselben Ergebnissen kommt.
Die neue OTX-Laufzeitumgebung von EMOTIVE gewährleistet plattformunabhängige Austauschbarkeit auf jeder sinnvollen Ebene. ■ (oe)

www.emotive.de

Dr. Jörg Supke ist Geschäftsführer der emotive GmbH & Co. KG, 73760 Ostfildern.