Anwendungen für Messen, Kalibrieren und Diagnose (ASAM AE MCD)

6

6.1 Einführung

ASAM, die Association for Standardization of Automation and Measuring Systems (ursprünglich ASAP Arbeitskreis zur Standardisierung von Applikationssystemen), ist eine Initiative der europäischen Automobilhersteller und ihrer Zulieferer, um die Applikation und den Test elektronischer Systeme in der Entwicklungs- und Fertigungsphase zu vereinfachen. In der Applikationsphase werden bei der Erprobung des Fahrzeugs und seiner Komponenten auf verschiedenen Prüfständen sowie im Fahrversuch Messwerte aus den Steuergeräten erfasst (*Measure*) und die steuergeräteinternen Parameter angepasst (*Calibrate*). In der Fertigungsphase wird der korrekte Einbau und die korrekte Funktion der Komponenten auf Fertigungsprüfständen erprobt und gegebenenfalls Feinjustage der Steuergerätedaten vorgenommen. In beiden Fällen werden Messdaten gesammelt (*Measure*), Diagnoseinformationen ausgewertet (*Diagnose*) und Steuergeräteparameter verstellt (*Calibrate*). Die Steuerung der Abläufe sowie die Datenauswertung und Datenhaltung erfolgt dabei durch eine Reihe von vernetzten Rechnersystemen, die unter anderem mit den Steuergeräten im Fahrzeug kommunizieren müssen. Innerhalb ASAM wird versucht, die Schnittstellen zwischen den einzelnen Prüfständen sowie die Datenaustauschformate zu standardisieren.

ASAM ist unterteilt in eine Reihe von Standards für die unterschiedlichen Ebenen des Gesamtsystems (Abb. 6.1) von den Steuergeräten und deren Applikations- und Diagnoseschnittstellen (MCD, neuerdings auch als Automotive Electronics AE bezeichnet), den Schnittstellen zwischen den Prüfstandskomponenten (GDI) bis zur Leitebene der übergeordneten Prüfstandssteuerung (ACI), Datenhaltung (ODS) und Datenanalyse (CEA). Im Folgenden soll lediglich die AE MCD-Ebene aus Sicht der Steuergeräte näher betrachtet werden.

In der Begriffswelt der Informatik würde man ASAM-MCD als *Middleware* bezeichnen, d. h. als eine Zwischenschicht, die eine Kopplung zwischen den Steuergeräten auf der untersten Ebene und den übergeordneten Test- und Diagnoseanwendungen herstellt

W. Zimmermann und R. Schmidgall, *Bussysteme in der Fahrzeugtechnik*, ATZ/MTZ-Fachbuch, 229
DOI 10.1007/978-3-658-02419-2_6, © Springer Fachmedien Wiesbaden 2014

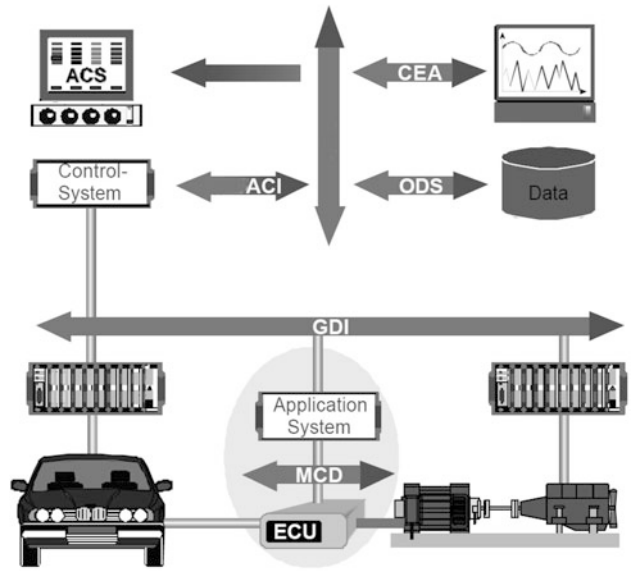

Abb. 6.1 Überblick über ASAM (Quelle: ASAM-Dokument: Introduction ASAM-MCD); *MCD*: Measure, Calibrate, Diagnose (AE … Automotive Electronics); *GDI*: Generic Device Interface; *ODS*: Open Data Service; *ACI*: Automatic Calibration Interface; *CEA*: Components for Evaluation and Analysis; *ECU*: Electronic Control Unit

(Abb. 6.2). Die Aufgabe der Zwischenschicht besteht darin, nach oben eine einheitliche, abstrahierte Schnittstelle zur Verfügung zu stellen und Implementierungsdetails der Steuergeräte zu kapseln. Die ASAM-MCD Middleware-Software läuft dabei auf dem Testsystem oder Diagnosetester, d. h. einem PC-artigen Rechnersystem mit den üblichen Schnittstellen zu anderen Rechnersystemen (Ethernet, USB usw.) und kommuniziert mit den Steuergeräten über die Kfz-üblichen Bussysteme (K-Line, CAN usw.). Häufig findet man aber auch für Applikationsaufgaben modifizierte Steuergeräte, bei denen der Steuergeräte-Flash-ROM-Speicher erweitert und durch RAM-Speicher ergänzt wurde (*Emulations-Tastkopf ETK*), um die Programme und Daten leichter ändern und schnell zwischen unterschiedlichen Werten umschalten zu können. Eine wesentliche Rolle bei ASAM MCD spielt außerdem die einheitliche, herstellerunabhängige Beschreibung der Steuergeräteeigenschaften in einer Datenbank mit standardisiertem Datenaustauschformat. Die Datenbank enthält Informationen über die im Steuergerät verfügbaren Funktionen und Variablen, z. B. deren Speicheradressen, die Parameter der Busschnittstelle sowie zur Umrechnung der steuergeräteinternen Daten in physikalische Werte. Insgesamt definiert ASAM-MCD drei Schnittstellen

- ASAM 3: Schnittstelle zu den übergeordneten Test- und Diagnoseanwendungen,
- ASAM 2: Schnittstelle zur Steuergerätedatenbank,
- ASAM 1: Schnittstelle zu den Steuergeräten.

Leider widerspricht die real existierende Situation in der MCD-Welt der sauberen Grundstruktur nach Abb. 6.2 und spiegelt sehr stark die historische Entwicklung sowohl in der Kfz-Elektronik als auch in der Informationstechnik seit Mitte der 90er Jahre wieder.

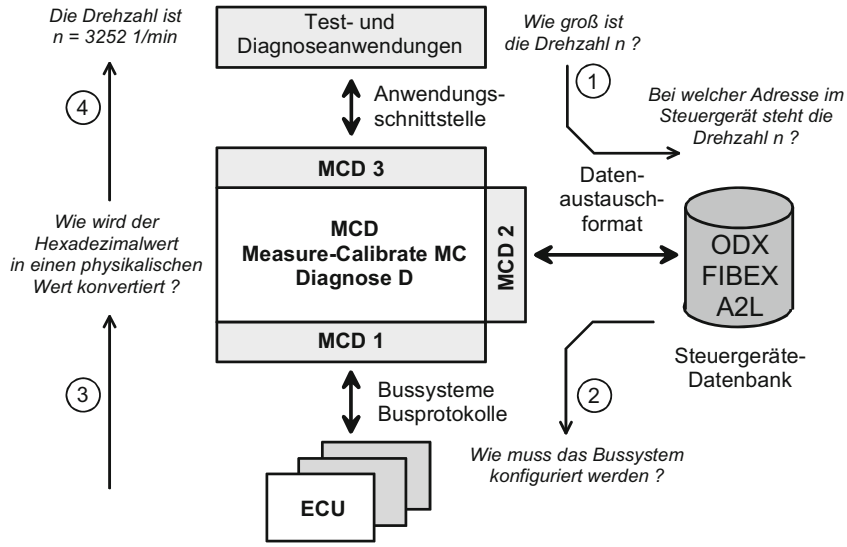

Abb. 6.2 ASAM AE MCD-Schnittstellen

Die Initiatoren der ASAM-Vorgängervereinigung ASAP beschäftigten sich zunächst lediglich mit Applikationssystemen und damit dem Bereich Messen und Kalibrieren (MC). Der Bereich Diagnose (D) war ausgeklammert. Die Diagnoseschnittstellen wurden parallel zur Arbeit von ASAP in ISO-Gremien normiert. Als ab Anfang 2000 versucht wurde, die Diagnoseaspekte mit zu berücksichtigen, musste auf die auf beiden Seiten existierenden, nur teilweise kompatiblen Lösungen Rücksicht genommen werden. Daher existiert heute für jede der drei Schnittstellen ASAM 1, 2, 3 jeweils eine -MC und eine -D Ausführung, die zwar so konzipiert sind, dass sie koexistieren können, die aber noch keineswegs vereinheitlicht sind.

Gleichzeitig spiegelt sich in den Standards die Weiterentwicklung in der Informationstechnik wieder (Tab. 6.1). Das Busprotokoll zu den Steuergeräten (ASAM 1MC) für den Bereich Messen-Kalibrieren war ursprünglich das CAN Calibration Protokoll CCP. Dieses wird allmählich durch das XCP-Protokoll abgelöst, das neben CAN und FlexRay auch andere Netzwerkvarianten (USB, Ethernet mit TCP/IP und UDP/IP) berücksichtigt. Für Diagnoseaufgaben dagegen sieht ASAM keine eigenen Standards vor, sondern verweist auf die etablierten ISO-Normen für KWP 2000 bzw. UDS.

Als Datenaustauschformat (ASAM 2) wurde ursprünglich ein proprietäres Textformat (ASAM MCD 2MC) definiert, auch als ASAM2 Meta Language AML oder in neueren Dokumenten als ASAP Classic bezeichnet. Dieses könnte zukünftig durch ein XML-basiertes Format ersetzt werden. Bereits XML-basiert sind das Format für die Beschreibung der Diagnosedaten (ASAM MCD 2D), als ODX (Open Data Exchange)-Format bekannt, das OTX-Format zur Beschreibung von Diagnoseabläufe sowie der als FIBEX bezeichnete Standard für die Beschreibung der Steuergeräte-Onboard-Kommunikation.

Tab. 6.1 ASAM-Varianten innerhalb des AE MCD Standards

Busprotokolle für Messen-Kalibrieren (MC)	
ASAM AE MCD 1MC CCP	CAN Calibration Protocol CCP
ASAM AE MCD 1MC XCP	Universal Measurement and Calibration Protocol XCP
Busprotokolle für Diagnose (D)	
Modular Vehicle Communication Interface (ISO 22900-1, -2 MVCI, D-PDU)	Hardware- und Softwareinterface zum Bussystem
KWP 2000 (K-Line oder CAN)	Verweis auf die ISO-Normen, keine Normierung innerhalb ASAM
UDS (CAN, FlexRay, Ethernet)	
Datenbeschreibungsformate	
ASAM AE MCD 2MC ASAP2	Proprietäres textbasiertes Datenformat für Messen-Kalibrieren (*ASAM Classic*, ASAM2 Meta Language AML)
ASAM AE MCD 2MC CDF	XML-basiertes Datenformat für Messen-Kalibrieren *CDF Calibration Data Format*, soll mittelfristig AML ablösen/ergänzen
ASAM AE MCD 2D ODX ISO 22901-1, -2	XML-basiertes Datenformat für Diagnosedaten *ODX Open Data Exchange*
ASAM AE MCD 2NET FIBEX	XML-basiertes Datenformat für die Steuergeräte-Onboard-Kommunikation *FIBEX Field Bus Exchange Format*
Anwendungsschnittstelle	
ASAM AE MCD 3 (ISO 22900-3, nur D-Server)	Einheitliche Schnittstelle für Mess-, Kalibrier- und Diagnoseanwendungen
ISO 13209 OTX	Open Test Sequence Exchange

Inzwischen teilweise noch als ASAP bezeichnete Dokumente nach ASAM überführt und einige ASAM-Vorschläge als ISO-Standards genormt. So ist ODX mittlerweile als ISO 22901 standardisiert. Die ASAM MCD 3D-Schnittstelle inklusive der Hard- und Software-Schnittstelle zum Diagnose-Bussystem, dort als *MVCI Modular Vehicle Communication Interface* und *D-PDU Diagnostic Protokoll Data Unit Interface* bezeichnet, wurde als ISO 22900 genormt.

6.2 Busprotokolle für Aufgaben in der Applikation (ASAM AE MCD 1MC)

Die für Mess- und Kalibrieraufgaben verwendete unterste ASAM-Schicht wird nochmals in zwei Teile untergliedert (Abb. 6.3). Der Teil 1a definiert die Busprotokolle CCP und XCP, der Teil 1b die funktionale Schnittstelle (Interface) zur übergeordneten Schicht. Für beide Teile sind Konfigurationsdateien notwendig.

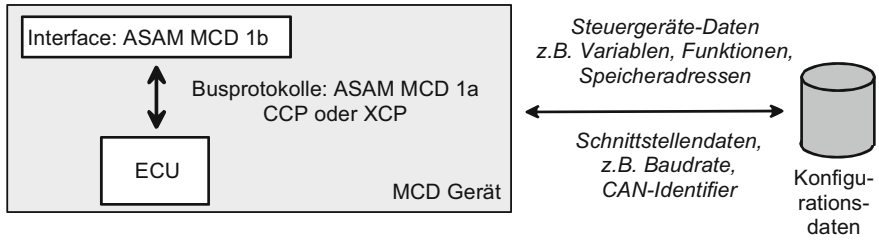

Abb. 6.3 ASAM-Schnittstelle zu den Steuergeräten für MC-Anwendungen

Das *Universal Measurement and Calibration-Busprotokoll XCP* basiert auf dem älteren *CAN Calibration Protocol CCP*, erweitert und modifiziert dieses allerdings so stark, dass es zu diesem nicht aufwärts kompatibel ist. Weiterhin erlaubt XCP außer CAN auch die Verwendung anderer Bussysteme von SPI (Serial Peripheral Interface, wird Steuergeräte-intern verwendet) bis zu Ethernet und USB. Diese Bussysteme werden heute zwar nicht von den eigentlichen Steuergeräten eingesetzt, in der Entwicklungsphase werden aber häufig modifizierte Geräte mit so genannten Applikationsadaptern verwendet. Die Applikationsadapter ersetzen den ROM-Speicher der Steuergeräte durch RAM-Speicher, um die Programme und Daten schneller ändern zu können, oder machen die Mikroprozessor-internen Register- und Speicherbereiche durch Emulator- oder Debugschnittstellen zugänglich. Um die übliche In-Vehicle-Kommunikation der Steuergeräte nicht zu stören, werden die Applikationsadapter in der Regel über ein zusätzliches unabhängiges Bussystem angebunden, wobei dann auch Ethernet, USB oder Firewire zum Einsatz kommen können. Aufgrund der historische Ableitung aus CCP und weil CAN bezüglich der möglichen Botschaftsgröße die stärkste Beschränkung aufweist, orientiert sich das XCP-Protokoll nahezu durchgängig an CAN. Die anderen Busvarianten stellen lediglich „Tunnel“ dar, die die CAN-typischen, max. 8 Byte großen Nutzdatenblöcke über ein anders geartetes Netzwerk durchleiten.

CCP und XCP verwenden eine verbindungsorientierte, logische Punkt-zu-Punkt-Verbindung zwischen dem ASAM-MCD-Applikationssystem und dem Steuergerät, wobei das Applikationssystem gleichzeitig mehrere Verbindungen zu unterschiedlichen Steuergeräten bedienen kann, während ein einzelnes Steuergerät nur genau eine Verbindung unterstützt. Das Konzept ist dem KWP 2000/UDS-Kommunikationsmodell nach Abschn. 5.1.1 und dem zugehörigen Single-Frame/Multi-Frame Transportprotokoll nach Abschn. 4.1 sehr ähnlich, doch verwenden die CCP/XCP-Normen leider andere Bezeichnungen (Abb. 6.4). Während das Steuergerät bei KWP 2000/UDS in der Kommunikation als *Server* und der Diagnoserechner als *Client* bezeichnet wird, ist das Steuergerät bei ASAM MCD der *Slave* und der Applikationsrechner der *Master*. Inhaltlich verbirgt sich dahinter exakt dieselbe Rollenverteilung: Alle Aktionen gehen vom externen Rechner (Diagnosetester, Applikationssystem) aus, der die Verbindung zum Steuergerät (*Slave*) aufbaut, Anfragen an das Steuergerät sendet und auf jede Anfrage eine Antwort erhält, bevor die nächste Anfrage gesendet wird (*Request-Response*-Verfahren).

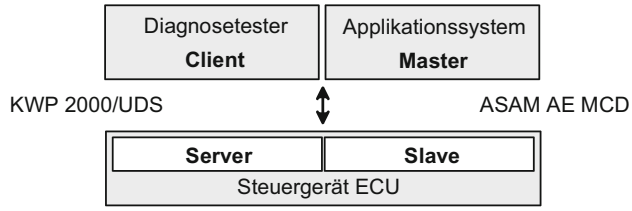

Abb. 6.4 Benennung der Kommunikationspartner bei Diagnose und Applikation

Vergleicht man CCP/XCP mit KWP 2000/UDS, so stellt man fest, dass KWP 2000/UDS bei Verwendung von CAN praktisch dieselbe Funktionalität bietet. Lediglich der Protokoll-Overhead bei KWP 2000/UDS ist höher, so dass die Messdatenerfassung bzw. Stimuluserzeugung mit CCP/XCP höhere Datenraten erreicht und die zeitliche Konsistenz von Datenblöcken besser gewährleistet ist. Historisch gesehen wurde CCP zu einem Zeitpunkt entwickelt, als KWP 2000 ausschließlich über die für Applikationszwecke viel zu langsame K-Line verfügbar war. Als KWP 2000 schließlich auch über CAN eingesetzt wurde, war CCP im Applikationsbereich so fest etabliert, dass den Betroffenen anscheinend eher eine Weiterentwicklung zu dem verwandten, aber auch zu CCP nicht kompatiblen XCP statt eine Modifikation von KWP 2000/UDS für Applikationszwecke sinnvoll erschien. Unter diesem Nebeneinander von in der Ausführung vollständig inkompatiblen, aber in der Grundfunktionalität stark verwandten Protokollen leidet die Zusammenführung der MC- und der D-Funktionen unter dem Dach von ASAM durchgehend.

6.2.1 CAN Calibration Protocol CCP

Bei CCP erfolgt die gesamte Kommunikation mit Hilfe von zwei CAN-Botschaften mit jeweils eigenem CAN-Identifier. Die Botschaften werden bei ASAM als Objekte bezeichnet:

- **Command Receive Object CRO** (Befehlsbotschaft) vom Applikationssystem zum Steuergerät; überträgt Befehle zum Steuergerät.
- **Data Transmission Object DTO** (Datenbotschaft) vom Steuergerät zum Applikationssystem; enthält die Steuergeräteantwort.

Eine *Befehlsbotschaft CRO* ist grundsätzlich 8 Byte groß, d. h. umfasst die maximal mögliche Nutzdatenlänge einer CAN-Botschaft, und hat ein festes Format (Abb. 6.5). Das erste Byte enthält den Befehlscode *CMD* (*Command Code*), das zweite Byte eine laufende Nummer *CTR* (*Command Counter*), die mit jeder CRO-Botschaft inkrementiert wird, sowie bis zu 6 Byte Daten bzw. Parameter des Befehls.

Für die *Datenbotschaft DTO* (*Data Transmission Object*) des Steuergerätes gibt es zwei Formate. Bei direkten Antworten *CRM* (*Command Return Message*) auf einen vorigen Befehl ist die DTO-Botschaft ebenfalls grundsätzlich 8 Byte lang. Als erstes Byte wird die CRM-Paket-Kennung (*Packet Identifer*) PID = FFh zurückgesendet, im zweiten Byte ein

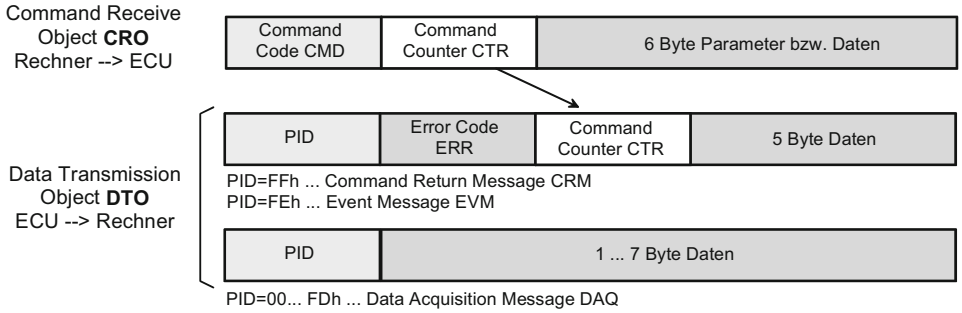

Abb. 6.5 CCP-Botschaftsformat

Fehlercode ERR (0 h bei fehlerfreier Ausführung des Befehls) und als drittes Byte die laufende Botschaftsnummer CTR der Befehlsbotschaft, zu der diese Antwort gehört. Die restlichen 5 Bytes enthalten Antwortdaten.

Dasselbe Format verwenden auch die *Event-Botschaften*, dabei wird aber PID = FEh gesetzt. Event-Botschaften werden vom Steuergerät asynchron gesendet, wenn Fehler auftreten, die sich nicht auf einen unmittelbar zuvor empfangenen Befehl beziehen.

Das zweite Format wird für die Messdatenübertragung *DAQ* (*Data Acquisition*) verwendet. Das PID-Byte bezieht sich dabei auf die Nummer einer so genannten Datenobjekttabelle *ODT* (*Object Descriptor Table*), die weiter unten beschrieben wird. Die weiteren Bytes enthalten eine variable Anzahl von Daten (max. 7 Byte). Nachdem die Messdatenübertragung vom Applikationssystem konfiguriert und gestartet wurde, erfolgt die Datenübertragung dann aus Sicht des Steuergeräts spontan, d. h. periodisch oder beim Eintreffen eines steuergeräteinternen Ereignisses, ohne dass weitere Anfragen vom Applikationssystem notwendig sind.

Die verfügbaren CCP-Befehle lassen sich in folgende Funktionsgruppen einteilen:

- Verbindungsaufbau und Steuerung,
- Zugriff auf den Steuergerätespeicher,
- Diagnose,
- Flash-Programmierung PGM,
- Kalibrieren CAL,
- Messdatenerfassung DAQ.

CCP-Verbindungsaufbau und Steuerung

CMD	Name	Zweck
01 h	CONNECT	Aufbau der logischen Verbindung zum Steuergerät durch das Applikationssystem. Die Botschaft enthält als Parameter eine 16 bit Kennziffer, die das Steuergerät eindeutig identifiziert. Das angesprochene Steuergerät antwortet mit einer positiven Antwort, alle übrigen Steuergeräte antworten nicht. Auf alle übrigen Botschaften reagiert das Gerät nur, solange eine Verbindung besteht.
05 h *	TEST	Diese Botschaft entspricht der CONNECT-Botschaft, baut aber selbst keine Verbindung auf, sondern dient nur der Abfrage, ob ein bestimmtes Steuergerät verfügbar ist. Auf diese Weise kann das Applikationssystem ermitteln, welche Steuergeräte am Bus angeschlossen sind.
07 h	DISCONNECT	Abbau der logischen Verbindung. Auch hier wird als Parameter die 16 bit Kennziffer mitgesendet, die das Steuergerät eindeutig identifiziert. Über einen weiteren Parameter kann die Verbindung auch nur zeitweise beendet werden, in diesem Fall nimmt das Steuergerät keine weiteren Befehle mehr entgegen (außer CONNECT), sendet aber weiterhin Messdaten.
12 h * 13 h *	GET_SEED UNLOCK	Anfordern eines Initialisierungswertes für den Schlüsselalgorithmus und Zurücksenden des ermittelten Schlüsselwertes als Login-Prozedur für den Zugriff auf geschützte Funktionen. Die Länge des Schlüsselwertes und der Schlüsselalgorithmus selbst sind wie bei KWP 2000 nicht in der Norm definiert (daher erfordert die MCD 1b-Schicht des Applikationssystems in der Regel die Einbindung eines herstellerspezifischen Schlüsselalgorithmus).
0Ch * 0Dh *	SET_S_STATUS GET_S_STATUS	Setzen und Abfragen von Statusinformationen, die anzeigen, ob das Applikationssystem im Steuergerät bestimmte Initialisierungsschritte, z. B. das Initialisieren der Kalibrierdaten oder das Initialisieren der Messdatenerfassung abgeschlossen hat. Außerdem kann das Steuergerät aufgefordert werden, diese Initialisierungsinformationen permanent zu speichern, so dass z. B. nach einem Steuergeräte-Reset automatisch eine Messdatenübertragung gestartet wird.
18 h	GET_CCP_ VERSION	Mit dieser Botschaft kann das Applikationssystem abfragen, welche CCP-Version, z. B. CCP 2.1, das Steuergerät unterstützt.
17 h	EXCHANGE_ID	Austausch von Konfigurationsinformationen. Das Steuergerät liefert eine 16 bit Bitmaske zurück, in der es anzeigt, welche der Funktionsblöcke Kalibrieren CAL, Messdatenerfassung DAQ und Flash-Programmierung PGM es unterstützt und für welche dieser Funktionen ein *Seed and Key*-Login-Vorgang notwendig ist. Der Befehl ermöglicht zusätzlich die Übertragung herstellerabhängiger Konfigurationsinformationen, evtl. mit Hilfe zusätzlicher UPLOAD-Botschaften.

(mit * markierte Befehle sind optional)

CCP-Zugriff auf den Steuergerätespeicher

CMD	Name	Zweck
02 h	SET_MTA	Setzen eines der beiden steuergeräteinternen Speicherpointer MTA0 bzw. MTA1. Die Adresse ist ein 32 bit + 8 bit-Wert, wobei der 8 bit-Teil zur Auswahl einer Speicherseite oder eines Segments dient. Vor jedem Datentransfer von oder zum Steuergerät muss zunächst mit diesem Befehl die Transferadresse gesetzt werden (Ausnahme: SHORT_UP). Bei weiteren Transfers auf aufeinander folgende Adressen ist dies nicht notwendig, da die Transferbefehle bei den Transfers den Pointer automatisch inkrementieren. MTA1 wird nur beim MOVE-Befehl verwendet.
03 h	DNLOAD	Schreiben von 1 bis 5 Datenbytes in den Steuergerätespeicher ab der Steuergeräte-Speicheradresse, auf die der Pointer MTA0 zeigt. Neben den Datenbytes wird die Länge des Datenblocks als Parameter übergeben. MTA0 wird nach der Übertragung automatisch um die entsprechende Anzahl von Bytes inkrementiert.
23 h *	DNLOAD_6	Schreiben von 6 Datenbytes, ansonsten wie DNLOAD. Durch die feste Länge muss keine Längenangabe übertragen werden, so dass ein Datenbyte mehr gesendet werden kann.
04 h	UPLOAD	Lesen von 1 bis 6 Datenbytes aus dem Steuergerätespeicher ab der Steuergeräte-Speicheradresse, auf die der Pointer MTA0 zeigt. MTA0 wird nach der Übertragung automatisch um die entsprechende Anzahl von Bytes inkrementiert.
0Fh *	SHORT_UP	Lesen von 1 bis 5 Datenbytes, dabei wird neben der gewünschten Länge die 32 bit + 8 bit Speicheradresse mitübertragen. Die Pointer MTA0 und MTA1 werden nicht verwendet, d. h. es ist kein vorheriger SET_MTA-Befehl notwendig.
19 h	MOVE	Kopieren einer im Befehl angegebenen Anzahl (32 bit Wert) von Bytes innerhalb des Steuergerätespeichers von der Adresse, auf die MTA0 zeigt, zu der Adresse, auf die MTA1 zeigt.

Im Gegensatz zu KWP 2000 und UDS verwendet CCP für den Zugriff auf Steuergerätedaten stets deren absolute Speicheradressen, d. h. das Applikationssystem benötigt exakte Informationen über die Speicherverteilung des Steuergerätes und der aktuellen Steuergerätesoftware. Für den Zugriff muss die CCP-Protokollsoftware im Steuergerät zwei Pointer MTA0 bzw. MTA1 verwalten, die über einen SET_MTA-Befehl vom Applikationssystem gesetzt werden, bevor über einen folgenden DNLOAD, UPLOAD oder MOVE-Befehl auf den Speicher zugegriffen wird.

CCP-Funktionsgruppe Flash-Programmierung PGM

CMD	Name	Zweck
10 h *	CLEAR_ MEMORY	Löschen einer bestimmten Anzahl von Bytes im Flash-ROM des Steuergerätes. Die Anzahl wird im Befehl übergeben, die Anfangsadresse des zu löschenden Speicherbereiches steht im Pointer MTA0 und muss mit einem vorherigen SET_MTA-Befehl initialisiert werden.
18 h *	PROGRAM	Programmiert 1 bis 5 Byte ab der Speicheradresse, auf die der Pointer MTA0 zeigt, in das Flash-ROM des Steuergerätes. Die Länge und die zu programmierenden Daten werden mit diesem Befehl übertragen. MTA0 wird anschließend um die entsprechende Anzahl von Bytes inkrementiert.
22 h *	PROGRAM_6	Wie PROGRAM, aber feste Länge von 6 Byte.
0Eh *	BUILD_ CHECKSUM	Berechnen und Auslesen einer Prüfsumme über einen Speicherbereich, dessen Länge im Befehl angegeben wird. Die Anfangsadresse steht in MTA0. Der Algorithmus zur Berechnung der Prüfsumme (1 bis 4 Byte) ist herstellerspezifisch. Checksummen werden üblicherweise verwendet, um die Konsistenz eines Datensatzes und den Erfolg des Flash-Programmiervorgangs zu prüfen.

CCP-Funktionsgruppe Diagnose

CMD	Name	Zweck
20 h *	DIAG_SERVICE	Starten eines Diagnosedienstes im Steuergerät. Mit dem Befehl wird die Service ID des Dienstes sowie 1 bis 4 Parameterbyte übergeben. Die Ergebnisse des Diagnosedienstes werden in den Steuergerätespeicher ab Adresse MTA0 kopiert, die Länge des Ergebnisses wird in der Steuergeräteantwort zurückübertragen. Die Ergebnisse selbst können über einen anschließenden UPLOAD-Befehl ausgelesen werden.
21 h *	ACTION_ SERVICE	Wie DIAG_SERVICE, nur dass eine herstellerspezifische Funktion im Steuergerät gestartet wird.

CCP-Funktionsgruppe Kalibrieren CAL

CMD	Name	Zweck
11 h *	SELECT_CAL_ PAGE	Umschalten auf eine neue Speicherseite. Die Anfangsadresse der neuen Speicherseite muss vorher über eine SET_MTA-Botschaft in den Pointer MTA0 geladen werden.
09 h *	GET_ACTIVE_ CAL_PAGE	Abfrage, welche Speicherseite gerade aktiv ist. Die Antwort liefert die 32 bit + 8 bit Anfangsadresse der Speicherseite zurück.

Für die Applikation werden häufig speziell umgerüstete Steuergeräte eingesetzt, deren ROM-Speicher durch RAM-Speicher ersetzt wurde. Um im laufenden Betrieb ganze Datensätze, z. B. den gesamten Parametersatz eines Reglers oder ein Kennfeld, in einem Schritt

aktivieren zu können, verwenden die Applikationssteuergeräte häufig so genannte Speicherseiten, von denen genau eine aktiv ist, d. h. für die Steuer- und Regelfunktionen des Steuergerätes verwendet wird, während die andere(n) Speicherseite(n) im Hintergrund vom Applikationssystem verändert werden können. Der Anfang einer Speicherseite wird wieder durch eine 32 bit + 8 bit-Adresse festgelegt.

Das Herunterladen der Datensätze erfolgt mit DNLOAD-Befehlen. Die Gültigkeit des Applikationsdatensatzes kann über BUILD_CHECKSUM geprüft werden. Als Anzeigemöglichkeit, ob der Applikationsdatensatz gültige Daten enthält oder nicht, kann das CAL-Bit im Steuergeräte-Statusbyte verwendet werden, das vom Applikationssystem mit SET_S_STATUS bzw. GET_S_STATUS geschrieben bzw. gelesen werden kann.

CCP-Funktionsgruppe Messdatenerfassung DAQ

CMD	Name	Zweck
14 h	GET_DAQ_SIZE	Abfrage, wie viele ODTs eine bestimmte DAQ Liste enthalten kann (Beschreibung von ODTs und DAQ Listen siehe unten). Die Nummer der Liste muss im Befehl angegeben werden. Durch die Abfrage wird die Liste automatisch gelöscht und eine gegebenenfalls laufende Übertragung gestoppt. Im Befehl wird zusätzlich angegeben, mit welchem CAN-Identifier die Messdaten gesendet werden sollen. Die Unterstützung unterschiedlicher CAN-Identifier ist optional. Neben der Anzahl der ODTs wird in der Steuergeräteantwort auch der PID-Wert des ersten Listeneintrags zurückgeliefert. Alle folgenden PIDs innerhalb der Liste müssen fortlaufend sein.
15 h	SET_DAQ_PTR	Dieser Befehl initialisiert einen steuergeräteinternen Pointer als Vorbereitung für WRITE_DAQ. Im Befehl muss die Nummer der DAQ Liste, die Nummer der ODT und die Nummer des Eintrags in der ODT angegeben werden.
16 h	WRITE_DAQ	Erstellt einen Eintrag in einer ODT. Die Auswahl des Eintrags erfolgt über einen vorherigen SET_DAQ_PTR-Befehl. Der Eintrag enthält eine 32 bit + 8 bit-Adresse eines Steuergerätespeicherbereichs sowie dessen Länge.
06 h	START_STOP	Startet oder stoppt die fortlaufende Messdatenübertragung für eine einzelne DAQ-Liste. Neben der Nummer der Liste und der Anzahl der ODTs in der Liste muss die Häufigkeit der Übertragung vorgegeben werden (Einzelheiten siehe unten). Statt die Messdatenübertragung direkt zu starten, kann der Start auch nur vorbereitet werden.
08 h *	START_STOP_ ALL	Startet die Messdatenübertragung für alle DAQ-Listen, die durch einen vorherigen START_STOP-Befehl für den Start vorbereitet wurden.

Mit Hilfe der Funktionsgruppe *Messdatenübertragung* werden in DAQ-Botschaften periodisch Daten aus dem Steuergerät zum Applikationssystem gesendet. Welche Datenbytes in einer CAN-Botschaft übertragen werden, wird für jede DAQ-Botschaft über eine Zuordnungstabelle, die *Object Descriptor Table ODT*, festgelegt (Abb. 6.6). Da in der Regel mehrere Datentabellen ODT verwendet werden, werden die Datentabellen durchnumme-

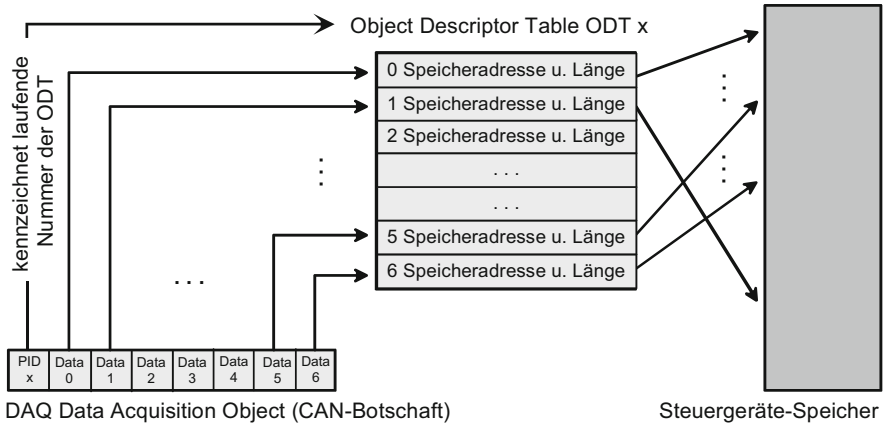

Abb. 6.6 Zuordnung von Messdaten in CAN-Botschaften zu Speicheradressen

riert und die Nummer der Tabelle im PID-Byte, dem ersten Datenbyte der CAN-Botschaft, mit übertragen. Die Einträge in einer ODT legen zu jedem der restlichen 7 Datenbytes der CAN-Botschaft die zugehörige Speicheradresse im Steuergerät fest. Für Daten, die mehr als ein Byte benötigen, kann die Tabelle auch so formatiert sein, dass statt der Adresse jedes Bytes die Anfangsadresse eines Datenwertes und dessen Länge eingetragen werden. Die Gesamtlänge aller in einer einzelnen ODT eingetragenen Daten darf maximal 7 Byte betragen. Da in der Regel mehr als 7 Datenbyte als konsistenter Messdatensatz gemessen und in mehreren DAQ-Botschaften übertragen werden müssen, lassen sich mehrere *Object Descriptor Tabellen ODTs* zu einem zusammengehörigen Datensatz, einer so genannten *DAQ-Liste* zusammenfassen (Abb. 6.7). Auch hier sind mehrere DAQ-Listen möglich, die

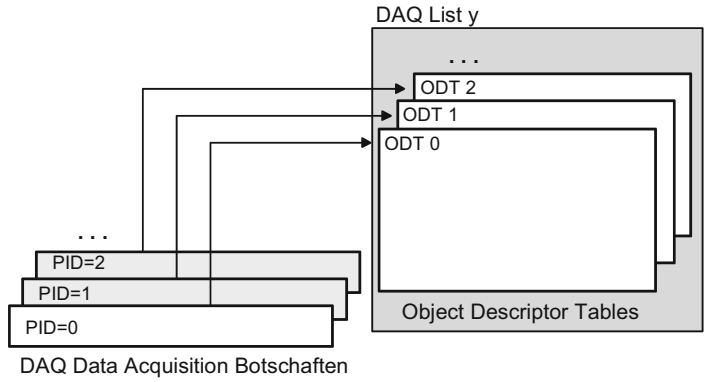

Abb. 6.7 Zusammenfassung von mehreren ODTs zu einer DAQ-Liste

ebenfalls durchnummeriert werden. Dabei kann optional für die Übertragung jeder DAQ-Liste ein eigener CAN-Identifier vorgesehen werden.

Bevor eine Messdatenübertragung gestartet wird, muss das Applikationssystem die DAQ-Listen und die ODT-Einträge (in einer Schleife) über die Befehle GET_DAQ_SIZE, SET_DAQ_PTR, WRITE_DAQ initialisieren.

Die eigentliche Datenerfassung und anschließende Datenübertragung im Steuergerät wird für jede DAQ-Liste durch ein Ereignis im Steuergerät ausgelöst. Als Ereignisquellen im Steuergerät können für periodische Übertragungen Zeitgeberinterrupts oder für spontane Übertragungen das Eintreffen von Signalen o. ä. festgelegt sein. Die CCP-Norm macht keinerlei Vorgaben, sondern schreibt lediglich vor, dass die möglichen Ereignisquellen (Event Channels) in den Steuergerätebeschreibungsdateien anzugeben und zu nummerieren sind. Mit dem START_STOP Befehl wird die Nummer der vom Applikationssystem ausgewählten Ereignisquelle an das Steuergerät übertragen. Zusätzlich wird ein *Vorteiler-Faktor* N übertragen, mit dem festgelegt werden kann, dass die Messdatenübertragung nur bei jedem N-ten Ereignis erfolgen soll.

Als Überwachung und Fehlerbehandlung definiert die Norm für alle Befehle Zeitschranken (Timeout) für die Steuergeräteantwort, in der Regel 25 ms. Größere Werte sind bei aufwendigeren Aufgaben wie der Flash-Programmierung oder dem Ausführen von Diagnosediensten möglich.

6.2.2 Extended Calibration Protocol XCP

XCP verwendet dasselbe verbindungsorientierte Request-Response-Kommunikationsschema und weitgehend dieselben Dienste wie CCP. Das Botschaftsformat und die Befehlscodes sind jedoch, wenn auch nur im Detail, unterschiedlich, so dass konkrete XCP-Implementierungen nicht aufwärts kompatibel zu CCP sind. Mit wenigen Ausnahmen sind die Botschaften unabhängig vom gewählten physikalischen Bussystem. In der CAN-Variante werden für jedes Steuergerät zwei feste, in einer Konfigurationsdatei vorgegebene CAN-Identifier verwendet. Einer dient zum Empfangen von Botschaften vom Applikationssystem zum Steuergerät, einer zum Senden von Botschaften vom Steuergerät zum Applikationssystem. Das heißt, im Gegensatz zu CCP verwendet XCP für jedes Steuergerät zwingend ein eigenes Paar von CAN-Identifiern. Über einen zusätzlichen Broadcast-CAN-Identifier kann das Applikationssystem mit Hilfe einer GET_SLAVE_ID-Botschaft die CAN-Identifier abfragen und optional für die Messdatenübertragung mit Hilfe von GET_DAQ_ID/SET_DAQ_ID-Botschaften zusätzliche CAN-Identifier dynamisch festlegen. Auf die wenigen spezifischen Botschaften für die anderen, alternativen Bussysteme (Ethernet, USB usw.) soll hier nicht eingegangen werden. Die XCP-Version für FlexRay wurde erstmals 2006 veröffentlicht. Im Vergleich zu anderen Bussystemen ist, wie am Ende dieses Kapitels dargestellt wird, ein höherer Aufwand notwendig, um eine ausreichende Bandbreite für die Applikationsaufgaben bedarfsabhängig zu garantieren, ohne diese dauerhaft für die eigentlichen Steuer- und Regelaufgaben zu blockieren.

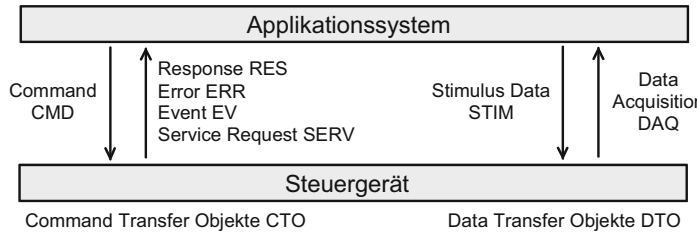

Abb. 6.8 XCP-Botschaftsgruppen

Wie bei CCP werden die Botschaften grob in zwei Gruppen unterteilt (Abb. 6.8):

- **Command Transfer Objekte CTO**
  Befehle (*Command Packet CMD*) vom Applikationssystem zum Steuergerät sowie Antworten (*Response Packet RES*) bzw. Fehlermeldungen des Steuergerätes (*Error Packet ERR*) sowie spontane Ereignisbotschaften (*Event Packet EV*) und Dienstanforderungen (*Service Request Packet SERV*) vom Steuergerät zum Applikationssystem.
- **Data Transfer Objekte DTO**
  Datenbotschaften (*Data Acquisition Packet DAQ*) für Messdaten vom Steuergerät zum Applikationssystem bzw. Stimulus-Daten (*Stimulus Data Packet STIM*) vom Applikationssystem zum Steuergerät.

Der STIM-Mechanismus wurde bei XCP neu eingeführt und stellt das Gegenstück zur schnellen Messdatenübertragung DAQ dar. Er ermöglicht die Realisierung von Bypass-Systemen, bei denen in der Entwicklungsphase Teile der Steuergerätefunktionalität in ein Applikations- bzw. Simulationssystem ausgelagert werden. DAQ und STIM zusammen ermöglichen dann im laufenden Steuergerätebetrieb Daten in beiden Richtungen schnell zu übertragen.

Die verwendeten XCP-Botschaftsformate (Abb. 6.9) enthalten im Gegensatz zu CCP (vgl. Abb. 6.5) keinen Botschaftszähler. Das erste Byte dient zur eindeutigen Unterscheidung des Botschaftsinhalts. ERR, EV und SERV-Antworten kodieren die Art des Fehlers, Ereignisses bzw. der Dienstanforderung im zweiten Byte. DAQ und STIM-Botschaften enthalten im zweiten Byte optional einen Zeitstempel. Dies ist im Vergleich zu CCP ebenfalls eine Neuerung.

Die XCP-Befehle lassen sich in dieselben Funktionsgruppen unterteilen wie bei CCP:

- Verbindungsaufbau und Steuerung sowie Zugriff auf den Steuergerätespeicher, zusammengefasst als Standardfunktionen STD bezeichnet,
- Flash-Programmierung PGM,
- Messdatenerfassung DAQ und Stimulation STIM,
- Kalibrieren CAL und Speicherseitenumschaltung PAG.

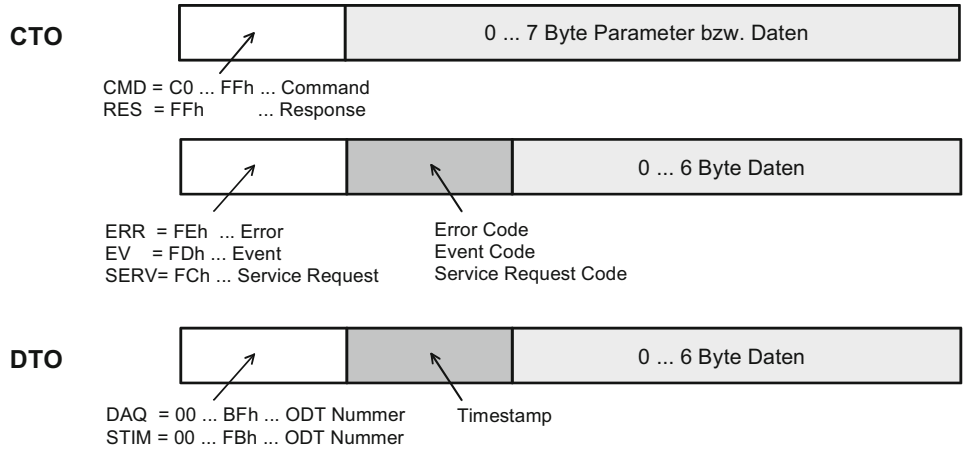

Abb. 6.9 XCP-Botschaftsformate

Die Konzepte entsprechen weitgehend den bereits im Abschn. 6.2.1 für CCP erläuterten Mechanismen, so dass in den folgenden Tabellen zwar alle XCP-Befehle aufgelistet, im jeweils anschließenden Text aber nur auf die Unterschiede zu CCP eingegangen wird. Die Funktionsgruppe Diagnose ist bei XCP entfallen, da hierfür mittlerweile die Standardprotokolle KWP 2000 und UDS verwendet werden.

Wiederum sind eine Reihe von Befehlen optional, die in den folgenden Tabellen mit * markiert wurden. Bevor einer dieser optionalen Befehle verwendet wird, muss das Applikationssystem mit Hilfe eines GET_…_INFO-Befehls für die zugehörige Funktionsgruppe zunächst abfragen, welche Optionen unterstützt werden.

XCP-Standardfunktionen für Verbindungsaufbau und -steuerung STD Teil 1

CMD	Name	Zweck
FFh	CONNECT	Aufbau der logischen Verbindung zum Steuergerät durch das Applikationssystem. Im Gegensatz zu CCP wird das angesprochene Steuergerät dabei durch die CAN-ID der Botschaft identifiziert, nicht durch eine in der Botschaft enthaltene Kennziffer. Auf alle übrigen Botschaften reagiert ein Steuergerät nur, solange eine logische Verbindung besteht. In der zugehörigen Antwort teilt das Steuergerät mit, welche Funktionsgruppen (Messdatenübertragung, Stimulation, Kalibrieren, Flash-Programmierung) unterstützt werden, welche XCP-Version implementiert ist, wie groß CTO und DTO Botschaften maximal sein dürfen (bei CAN stets 8 Byte), welche Bytefolge (Little bzw. Big Endian) und Adressgranularität (Byte, Word, Double Word) das Steuergerät verwendet und ob das Steuergerät nicht nur Einzelbotschaften sondern bei UPLOAD-Befehlen Blockübertragungen ähnlich dem Transportprotokoll ISO 15765-2 unterstützt.

CMD	Name	Zweck
FBh *	GET_COMM_ MODE_INFO	Abfrage, ob das Steuergerät für PROGRAM_NEXT und DOWNLOAD_NEXT-Befehle eine Blockübertragung ähnlich ISO 15765-2 unterstützt.
FDh	GET_STATUS	Setzen und Abfragen von Statusinformationen, die anzeigen, ob ein Speichervorgang der Kalibrierdaten bzw. der Konfiguration der Messdatenerfassung im Flash-ROM oder die Messdatenübertragung aktiv ist. Weiterhin wird abgefragt, welche der Funktionsgruppen Messdatenübertragung, Stimulation, Kalibrieren, Flash-Programmierung durch einen Seed and Key-Mechanismus geschützt sind.
F8 h * F7 h *	GET_SEED UNLOCK	Anfordern eines Initialisierungswertes für einen Schlüsselalgorithmus und Zurücksenden des ermittelten Schlüsselwertes als Login-Prozedur für den Zugriff auf geschützte Funktionen. Die Länge des Schlüsselwertes und der Schlüsselalgorithmus selbst sind wie bei KWP 2000/UDS nicht in der Norm definiert (daher erfordert die MCD 1b-Schicht des Applikationssystems in der Regel die Einbindung eines herstellerspezifischen Schlüsselalgorithmus). Der Schlüssel muss für jede Funktionsgruppe (Messdatenübertragung, Stimulation, Kalibrieren, Flash-Programmierung) separat ausgetauscht werden. Das Protokoll erlaubt das Austauschen von Schlüsselwerten mit mehr als 6 Byte mit Hilfe mehrerer aufeinander folgender CAN-Botschaften.
FAh *	GET_ID	Abfrage der Steuergeräte-Identifikation. Dabei kann es sich um einen ASCII-Text mit dem entsprechenden Inhalt oder um den Verweis (URL) auf eine entsprechende Beschreibungsdatei handeln. Falls die Informationen nicht in die Antwortbotschaft passen, wird ein steuergeräteinterner Pointer gesetzt, von dem aus die Daten mit Hilfe von UPLOAD-Befehlen abgeholt werden können.
FCh	SYNC	Neuinitialisierung nach einem Timeout
FEh	DISCONNECT	Abbau der logischen Verbindung
F1 h	USER_CMD	Übertragung herstellerdefinierter Befehle. Üblicherweise wird über das zweite Byte der Befehl spezifiziert.
F2 h	TRANSPORT_ LAYER_CMD	Bussystem-abhängiger Befehl. Der Befehl wird über das zweite Byte ausgewählt. Für CAN sind folgende Befehle definiert: GET_SLAVE_ID (FFh): Abfrage der CAN-Identifier aller am Bus angeschlossenen, XCP-fähigen Steuergeräte. Diese Botschaft wird mit dem Broadcast-CAN-Identifier gesendet. GET_DAQ_ID (FEh) und SET_DAQ_ID (FDh): Abfrage und gegebenenfalls Neusetzen des CAN-Identifiers für eine bestimmte Liste von Messdaten (DAQ-Liste).
F9 h *	SET_REQUEST	Starten eines Speichervorgangs der Kalibrierdaten bzw. der Konfiguration der Messdatenerfassung im Flash-ROM. Der Fortschritt des Speichervorgangs kann über GET_STATUS abgefragt werden.

(mit * markierte Befehle sind hier und in den weiteren Tabellen jeweils optional)

Zu den wesentlichen Neuerungen von XCP gehört, dass zusätzlich zum bekannten Request-Response-Schema, bei dem das Applikationssystem eine Befehlsbotschaft an das Steuergerät sendet und genau eine Antwortbotschaft zurückerhält, zusätzlich als Option der Blockmodus nach ISO 15765-2 (ISO-TP) unterstützt wird. Dabei wird unterschieden zwischen dem so genannten *Slave Block Mode*, bei dem das Steuergerät auf eine einzelne Befehlsbotschaft mit mehreren aufeinander folgenden Antwortbotschaften reagieren kann, und dem *Master Block Mode*, bei dem das Applikationssystem auf eine Befehlsbotschaft unmittelbar weitere Botschaften mit zugehörigen Befehlsparametern bzw. Daten senden darf. Der *Slave Block Mode* wird hauptsächlich in Verbindung mit dem Auslesen größerer Speicherbereiche des Steuergerätes (UPLOAD) verwendet, der *Master Block Mode* beim Herunterladen von größeren Blöcken von Kalibrier- und Programmierdaten (DOWNLOAD). Die Unterstützung der beiden Block-Modi durch das Steuergerät ist optional und wird dem Applikationssystem durch die Antworten auf die Verbindungsaufnahme (CONNECT) bzw. durch explizite Abfrage (GET_COMM_MODE_INFO) mitgeteilt. Dabei erfährt das Applikationssystem auch, welche der Funktionsgruppen CAL/PAG, DAQ, STIM und PGM das Steuergerät unterstützt. Über GET_ID gibt das Steuergerät Klartextinformationen über sich selbst oder die URL einer zugehörigen Beschreibungsdatei bekannt.

Über GET_STATUS erfährt das Applikationssystem, welche Funktionsgruppen über einen Seed and Key-Mechanismus geschützt sind und kann diese Funktionsgruppen über GET_SEED und UNLOCK freischalten.

Kommt es bei den Steuergeräte-Antworten zu einem Timeout oder antwortet ein Steuergerät auf eine Anfrage mit einer Fehlermeldung, so definiert der Standard, wie das Applikationssystem reagieren soll. Die Timeout-Werte selbst sind in der Schnittstellenkonfigurationsdatei des Steuergerätes festgelegt. Die Mehrzahl der Fehlermeldungen zeigt lediglich an, dass bestimmte Funktionen nicht unterstützt werden oder Parameter außerhalb des zulässigen Bereiches liegen. Bei Timeout-Fehlern wird versucht, die Kommunikation mit einer SYNC-Botschaft neu zu synchronisieren. Falls dies misslingt oder bei anderen schweren Fehlern wird ein neuer Verbindungsaufbau notwendig.

Über einen der Transport Layer Befehle kann das Applikationssystem mit Hilfe einer Broadcast-Nachricht alle am Bus angeschlossenen Steuergeräte auffordern, ihm die vom jeweiligen Steuergerät für die Kommunikation mit dem Applikationssystem verwendeten CAN-Identifier mitzuteilen. Diese Botschaften sind die einzigen Botschaften, auf die alle Steuergeräte ohne vorheriges CONNECT antworten.

Über USER_CMD können herstellerspezifische Befehle implementiert werden.

Wie bereits bei CCP kann das Steuergerät Kalibrierdaten und Konfigurationen für die Messdatenübertragung für einen folgenden Neustart (Reset) speichern. Die Speicherung erfolgt in der Regel im Flash-ROM bzw. EEPROM und wird über SET_REQUEST ausgelöst. Ob die Speicherung erfolgreich abgeschlossen wurde, wird über GET_STATUS abgefragt, bevor ein Reset ausgelöst wird.

XCP-Funktionen für den Zugriff auf den Steuergerätespeicher STD Teil 2

CMD	Name	Zweck
F6 h	SET_MTA	Setzen eines steuergeräteinternen Speicherpointers MTA (Memory Transfer Address). Die Adresse ist ein 32 bit + 8 bit-Wert, wobei der 8 bit-Teil z. B. zur Auswahl einer Speicherseite oder eines Segments dient. Vor jedem Datentransfer von oder zum Steuergerät muss zunächst mit diesem Befehl die Transferadresse gesetzt werden. Bei weiteren Transfers von/zu aufeinander folgenden Adressen ist dies nicht notwendig, da die Transferbefehle bei den Transfers den verwendeten Pointer automatisch inkrementieren.
F0 h EFh *	DOWNLOAD DOWNLOAD_NEXT	Schreiben von Datenbytes in den Steuergerätespeicher ab der Steuergeräte-Speicheradresse, auf die der Pointer MTA zeigt. Neben den Datenbytes wird die Länge des Datenblocks als Parameter übergeben. Wenn das Steuergerät nur Einzel-Botschaften unterstützt, können maximal 6 Byte geschrieben werden, bei Blockübertragung bis zu 255 Byte.
EEh *	DOWNLOAD_MAX	Schreiben von 7 Datenbytes, ansonsten wie DOWNLOAD. Durch die feste Länge muss keine Längenangabe übertragen werden, so dass ein Datenbyte mehr übertragen werden kann.
F5 h *	UPLOAD	Lesen von Daten aus dem Steuergerätespeicher ab der Steuergeräte-Speicheradresse, auf die der Pointer MTA zeigt. Wenn das Steuergerät nur Einzel-Botschaften unterstützt, können maximal 7 Byte gelesen werden, sonst bis zu 255 Byte.
F4 h *	SHORT_UPLOAD	Lesen von 1 bis 7 Datenbytes, dabei wird neben der gewünschten Länge die 32 bit + 8 bit Speicheradresse mitübertragen Der Pointer MTA wird nicht verwendet, d. h. es ist kein vorheriger SET_MTA-Befehl notwendig.
ECh *	MODIFY_BITS	Setzen oder Löschen von Bits innerhalb eines 32 bit-Datenwortes ab der Steuergeräte-Speicheradresse, auf die der Pointer MTA zeigt.

Der Zugriff auf den Steuergerätespeicher erfolgt wieder über Steuergeräte-Speicheradressen (32 bit + 8 bit), wobei über SET_MTA zunächst ein steuergeräteinterner Pointer gesetzt werden muss, bevor auf die Adressen zugegriffen wird. Die UPLOAD und DOWNLOAD-Befehle und ihre Varianten werden in der Regel in Verbindung mit den nachfolgend beschriebenen Funktionsgruppen PGM und CAL verwendet.

XCP-Funktionsgruppe Flash-Programmierung PGM

CMD	Name	Zweck
D2 h	PROGRAM_START	Schaltet das Steuergerät in den Betriebszustand zum Programmieren des Flash-ROMs bzw. EEPROMs. Erfordert in der Regel ein vorheriges Freischalten über den Seed and Key-Mechanismus. Das Steuergerät teilt in der Antwort mit, ob es für den Download bzw. Upload eine Blockübertragung nach ISO 15765-2 unterstützt.
CCh *	PROGRAM_ PREPARE	Falls der Programmieralgorithmus bzw. Treiber für das Flash-ROM nicht bereits im Steuergerät gespeichert ist, bereitet dieser Befehl das Steuergerät auf das Herunterladen der Programmierdaten vor. Das Steuergerät muss einen ausreichend großen Speicherblock als Zwischenspeicher reservieren. Das eigentliche Herunterladen erfolgt anschließend über DOWNLOAD-Botschaften.
CEh *	GET_PGM_ PROCESSOR_INFO	Abfrage, welche Optionen der Flash-ROM-Programmieralgorithmus des Steuergerätes unterstützt (Absolute oder Functional Address Mode, Kompression, Verschlüsselung, Zahl der Speichersektoren, wahlfreie Programmierung usw.)
CDh *	GET_SECTOR_INFO	Abfrage der Anfangsadresse bzw. Länge eines Flash-ROM-Sektors (kleinster lösch- und programmierbarer Speicherbereich) sowie Abfrage, in welcher Reihenfolge die Sektoren gelöscht bzw. programmiert werden müssen, falls der Flash-ROM-Baustein keine beliebige Sektor-Reihenfolge (wahlfreie bzw. sequenzielle Programmierung) zulässt.
CBh *	PROGRAM_ FORMAT	Informiert das Steuergerät, ob die Daten für den folgenden Programmiervorgang komprimiert und/oder verschlüsselt übertragen werden, wählt einen anwendungsspezifischen Programmieralgorithmus aus und schaltet zwischen Absolute und Functional Address Mode um (siehe unten).
D1 h	PROGRAM_CLEAR	Löschen einer bestimmten Anzahl von Bytes im Flash-ROM oder EEPROM des Steuergerätes. Die Anzahl wird im Befehl übergeben, die Anfangsadresse des zu löschenden Speicherbereiches steht im Pointer MTA (Absolute Address Mode) und muss mit einem SET_MTA-Befehl vorher initialisiert werden. Alternativ kann der zu löschende Bereich (Kalibrierdatensatz, Programmcode, EEPROM usw.) über eine Bitmaske ausgewählt werden (Functional Address Mode).
D0 h	PROGRAM	Programmiert 1 bis 6 Byte in das Flash-ROM bzw. EEPROM des Steuergerätes. Die Länge und die zu programmierenden Daten werden mit diesem Befehl übertragen. Die Anfangsadresse entspricht im Absolute Address Mode dem aktuellen Wert des Pointers MTA oder wird im Functional Address Mode implizit festgelegt. Welcher Modus verwendet wird, wird durch den vorherigen PROGRAM_CLEAR-Befehl definiert. MTA wird anschließend entsprechend inkrementiert.

CMD	Name	Zweck
CAh *	PROGRAM_NEXT	Wie PROGRAM. Diese Botschaft wird für die folgenden Programmierdaten verwendet, falls die Blockdatenübertragung verwendet wird.
C9 h *	PROGRAM_MAX	Wie PROGRAM, aber feste Länge von 7 Byte.
C8 h *	PROGRAM_VERIFY	Fordert das Steuergerät auf, einen ausgewählten Teil des Flash-ROM-Speichers zu verifizieren. Der Prüfalgorithmus ist herstellerspezifisch. Dies ist eine Alternative zur Checksummenprüfung.
CFh	PROGRAM_RESET	Beendet eine Programmiersequenz. Das Steuergerät bricht die Verbindung ab und führt im Allgemeinen einen Reset durch.
F3 h *	BUILD_CHECKSUM	Berechnen und Auslesen einer Prüfsumme über einen Speicherbereich, dessen Länge im Befehl angegeben wird. Die Anfangsadresse steht in MTA. Der Algorithmus zur Berechnung der Prüfsumme (1 bis 4 Byte) ist herstellerspezifisch. Vordefiniert sind Speicherwortsummen bzw. verschiedene CRC-Algorithmen. Checksummen werden üblicherweise verwendet, um die Konsistenz eines Datensatzes und den Erfolg eines Flash-Programmiervorgangs zu prüfen.

Mit PROGRAM_START wird das Steuergerät in den Programmiermodus geschaltet. Der eigentliche Programmieralgorithmus kann Teil des Steuergeräteprogrammes in einem Bereich des Flash-ROM-Speichers sein, der nicht gelöscht wird, oder vor dem Programmiervorgang mit PROGRAM_PREPARE und DOWNLOAD in das Steuergerät geladen werden. Über die GET_…_INFO- und PROGRAM_FORMAT-Befehle beschafft sich das Applikationssystem Informationen über die Anzahl und Länge der Speichersektoren und darüber, ob die Daten in komprimierter oder verschlüsselter Form in das Steuergerät geladen werden sollen. Ein Sektor ist die kleinste löschbare Einheit eines Flash-ROM-Speichers und wird entweder durch seine Speicheradresse (Absolute Address Mode) oder durch die Sektornummer (Functional Address Mode) identifiziert. Das Löschen eines Sektors erfolgt durch PROGRAM_CLEAR, bevor dann mit einer Reihe von PROGRAM_…-Botschaften die Daten heruntergeladen und sofort programmiert werden. Anschließend werden die programmierten Daten gegebenenfalls durch eine Checksummenprüfung oder einen steuergerätespezifischen Verifikationsalgorithmus überprüft und schließlich der Programmiervorgang mit PROGRAM_RESET abgeschlossen.

XCP-Funktionsgruppe Kalibrieren CAL und Speicherseitenumschaltung PAG

CMD	Name	Zweck
Ebh	SET_CAL_PAGE	Umschalten auf eine neue Speicherseite so, dass die Seite entweder für die Steuergeräte-Applikation oder für das Applikationssystem oder für beide zugänglich ist. Die Speicherseite wird durch eine Segment- und Seitennummer ausgewählt.
EAh	GET_CAL_PAGE	Abfrage, welche Speicherseite in einem vorgegebenen Speichersegment gerade für die Steuergeräte-Applikation bzw. für das Applikationssystem aktiviert ist. Die Antwort liefert die Seitennummer der Speicherseite zurück.
E9 h *	GET_PAGE_ PROCESSOR_INFO	Abfrage, wie viele Speichersegmente verfügbar sind und ob der Speicher eingefroren werden kann.
E8 h *	GET_SEGMENT_ INFO	Abfrage der Anfangsadresse und Länge eines Speichersegments sowie der Anzahl der Seiten des Segments.
E7 h *	GET_PAGE_INFO	Abfrage, ob das Steuergerät bzw. das Applikationssystem lesend bzw. schreibend auf eine Seite zugreifen können, ob das Steuergerät gleichzeitig mit dem Applikationssystem oder nur exklusiv auf eine Speicherseite zugreifen kann und ob es sich um die Seite mit den Initialisierungsdaten für diese Seite handelt.
E6 h *	SET_SEGMENT_ MODE	Einfrieren eines Speichersegmentes.
E5 h *	GET_SEGMENT_ MODE	Abfrage, ob ein Speichersegment eingefroren ist.
E4 h *	COPY_CAL_PAGE	Kopieren einer Speicherseite aus einem Speichersegment in ein anderes Speichersegment.

Für Applikationszwecke wird der Speicher eines Steuergerätes häufig ergänzt oder erweitert, indem in ein und denselben physikalischen Adressbereich (Segment) alternativ mehrere Speicherseiten (Page) eingeblendet werden können (Abb. 6.10). Dabei ist in jedem Segment genau eine Speicherseite in den physikalischen Adressraum des Steuergerätes eingeblendet, während die anderen Speicherseiten des Segments lediglich im Hintergrund liegen und nur für das Applikationssystem zugänglich sind. Häufig existiert für jedes Segment eine Seite im Flash-ROM, die das Standardprogramm bzw. die Standarddaten enthält, sowie eine oder mehrere Seiten, die vom Applikationssystem mit geänderten Daten geladen werden. Durch den SET_CAL_PAGE-Befehl kann das Applikationssystem für jedes Segment umschalten, welche Seite im Steuergeräteadressraum eingeblendet ist (Active ECU Page) und welche Seite für das Applikationssystem zugänglich ist (Active XCP Page), wobei zwischen *nur lesbar* und *les- und schreibbar* unterschieden wird. Abhängig vom Hardwareaufbau des Speichers ist es gegebenenfalls möglich, dass das Applikationssystem gleichzeitig mit dem Steuergerät auf dasselbe Segment zugreift. Hierdurch kann das Applikationssystem z. B. den Parametersatz eines Reglers im Hintergrund verändern und dann auf den neuen Parametersatz umschalten. Informationen über Segmente und Seiten er-

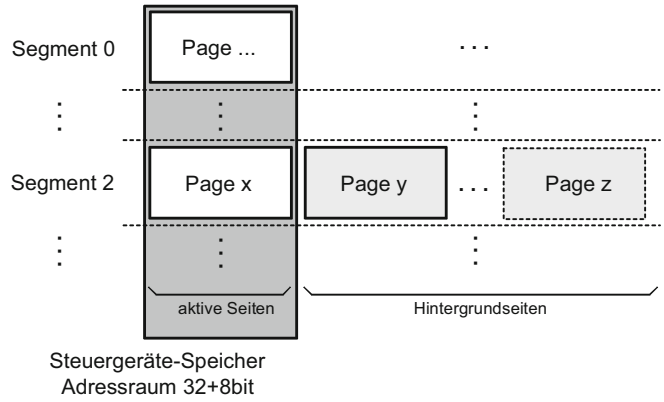

Abb. 6.10 XCP-Speicherstruktur für die Funktionsgruppe CAL/PAG

hält das Applikationssystem über die GET_…_INFO-Abfragen. Mit COPY_CAL_PAGE können Speicherseiten im selben Segment oder in ein anderes Segment kopiert und mit SET_SEGMENT_MODE eingefroren, d. h. gegen weiteres Verändern geschützt und nötigenfalls in die zugehörige Flash-ROM-Seite kopiert werden.

XCP-Funktionsgruppe Messdatenerfassung DAQ und Stimulation STIM

CMD	Name	Zweck
E2 h	SET_DAQ_PTR	Dieser Befehl initialisiert einen steuergeräteinternen Pointer als Vorbereitung für WRITE_DAQ. Im Befehl muss die Nummer der DAQ-Liste, die Nummer der ODT und die Nummer des Eintrags in der ODT angegeben werden (siehe unten).
E1 h	WRITE_DAQ	Erstellt einen Eintrag in einer ODT. Die Auswahl des Eintrags erfolgt über einen vorherigen SET_DAQ_PTR-Befehl. Der Eintrag enthält eine 32 bit + 8 bit-Adresse eines Steuergerätespeicherbereichs sowie dessen Länge. Nach dem Auslesen wird der DAQ-Pointer automatisch auf den nächsten Eintrag weitergeschaltet.
DBh *	READ_DAQ	Auslesen eines Eintrags in einer ODT. Komplementäre Funktion zu WRITE_DAQ.
DEh	START_STOP_ DAQ_LIST	Startet oder stoppt die laufende Messdatenübertragung für eine einzelne DAQ-Liste. Statt die Messdatenübertragung direkt zu starten, kann eine Liste auch nur zum Start oder Stopp vorausgewählt werden.
DDh	START_STOP_SYNC	Startet oder stoppt die Messdatenübertragung für alle DAQ-Listen gleichzeitig, die zuvor durch START_STOP_DAQ_LIST ausgewählt wurden.
E3 h	CLEAR_DAQ_LIST	Löschen einer DAQ-Liste, Stoppen der zugehörigen Datenübertragung, falls diese aktiviert war.

CMD	Name	Zweck
E0 h	SET_DAQ_LIST_ MODE	Setzt Parameter für die Datenübertragung: Übertragungsrichtung Messdaten (DAQ)/Stimuli (STIM), Zeitstempel ein/aus, Übertragung der ODT-Nummer ein/aus (nur möglich, wenn die Daten der DAQ-Liste mit einem eindeutigen CAN-Identifier übertragen werden). Auswahl des Ereigniskanals, der die eigentliche Datenübertragung triggert, des Teilerfaktors für die Wiederholrate und der Priorität, mit der die Daten übertragen werden, falls Daten aus mehreren DAQ-Listen gleichzeitig zur Übertragung anstehen.
DFh	GET_DAQ_LIST_ MODE	Abfrage der mit SET_DAQ_LIST_MODE gesetzten Werte für eine DAQ-Liste.
DCh *	GET_DAQ_CLOCK	Abfrage des aktuellen Standes des Steuergeräte-Timers, mit dem die Zeitstempel der DAQ-Botschaften erzeugt werden. Wird in der Regel zur Synchronisation mit der Zeitbasis des Applikationssystems verwendet.
DAh *	GET_DAQ_ PROCESSOR_INFO	Abfrage verschiedener DAQ-Optionen: Dynamisch eingerichtete DAQ-Listen, Resume-Modus (Speichern von DAQ-Listen im Flash-ROM und Start einer automatischen Datenübertragung nach Steuergeräte-Reset), Unterstützung von Zeitstempeln, Abschalten der Übertragung von ODT-Nummern, Anzahl der möglichen DAQ-Listen und Unterscheidung von statisch konfigurierten und dynamisch einrichtbaren Listen, Anzahl der verfügbaren Ereigniskanäle und andere.
D9 h *	GET_DAQ_ RESOLUTION_INFO	Abfrage weiterer DAQ-Optionen: Größe der ODT-Einträge (1, 2, 4 Byte), max. zulässige Länge des Speicherbereichs, auf den ein ODT-Eintrag zeigt, Länge und Zeitauflösung des Zeitstempels.
D8 h *	GET_DAQ_ LIST_INFO	Abfrage der Optionen einer DAQ-Liste: Statisch/dynamisch konfigurierte Liste, fester/konfigurierbarer Ereigniskanal, Anzahl der zugeordneten ODTs und Anzahl der Einträge je ODT.
D7 h *	GET_DAQ_ EVENT_INFO	Abfrage der Einstellungen eines bestimmten Ereigniskanals: Wiederholrate, Priorität, Anzahl der DAQ-Listen, die durch das Ereignis getriggert werden können. Die Antwort enthält außerdem eine Information, ob eine Klartextbeschreibung für den Kanal verfügbar ist, die dann über anschließende UPLOAD-Befehle gelesen werden kann.
D6 h *	FREE_DAQ	Gibt alle dynamisch eingerichteten DAQ-Listen frei. Muss gesendet werden, bevor dynamische DAQ-Listen mit ALLOC_DAQ eingerichtet werden.
D5 h *	ALLOC_DAQ	Reserviert Speicherplatz für eine bestimmte Anzahl von DAQ-Listen (so genannte dynamische DAQ-Listen).
D4 h *	ALLOC_ODT	Reserviert Speicherplatz für eine bestimmte Anzahl von ODTs und ordnet sie einer bestimmten DAQ-Liste zu.
D3 h *	ALLOC_ODT_ENTRY	Reserviert Speicherplatz für eine bestimmte Anzahl von ODT-Einträgen und ordnet Sie einer ODT zu.

Die automatische, periodische Datenübertragung wird in derselben Weise wie bei CCP durch in DAQ-Listen zusammengefasste Object Descriptor Tabellen ODT mit Hilfe von SET_DAQ_PTR und WRITE_DAQ konfiguriert (Abb. 6.6 und 6.7). Neu bei XCP ist, dass nicht nur Messdaten vom Steuergerät zum Applikationssystem übertragen werden können (DAQ), sondern dass auch Stimulidaten in umgekehrter Richtung übertragen werden dürfen (STIM). Außerdem können zusammen mit den Daten Zeitstempel übertragen werden, die sich auf den steuergeräteinternen Zeitgeber beziehen und anzeigen, wann die Messdaten im Steuergerät erfasst bzw. wann die Stimulidaten im Steuergerät wirksam werden sollen. Ob das Steuergerät den STIM-Betrieb unterstützt und welche Optionen bei DAQ und STIM im Einzelnen möglich sind, kann über GET_DAQ_…_INFO-Botschaften abgefragt werden.

Das Starten und Stoppen der eigentlichen Datenübertragung erfolgt wiederum mit Hilfe von START_STOP…-Befehlen.

Während bei CCP die Menge der gleichzeitig übertragbaren Messwerte durch die in der Steuergerätesoftware vorgegebene Anzahl von DAQ-Listen und ODT-Einträgen fest vorgegeben war, können XCP-taugliche Steuergeräte optional das dynamische Anfordern weiterer DAQ-Listen und ODT-Einträge mit Hilfe von ALLOC_…-Botschaften zulassen, wenn sie über ausreichend freien RAM-Speicher verfügen und dessen dynamische Verwaltung erlauben.

Neben den Antworten auf unmittelbare Anfragen des Applikationssystems und der periodischen Messdatenübertragung kann das Steuergerät spontan *EV-Botschaften* an das Applikationssystem senden, wenn spezielle Ereignisse eintreten. Dazu gehört z. B., wenn der Speichervorgang von Kalibrierdaten oder die Konfiguration von DAQ-Listen im Flash-ROM abgeschlossen ist, wenn die Ausführung eines Befehls länger dauert und die Timeout-Überwachung neu gestartet werden muss, wenn die Messdatenübertragung das Steuergerät überlastet ist, wenn das Steuergerät von sich aus die Verbindung beenden will oder wenn ein herstellerdefiniertes Ereignis auftritt. Mit Hilfe einer *SERV-Botschaft* kann das Steuergerät das Applikationssystem sogar auffordern, das Steuergerät zurückzusetzen. Dies erfordert in der Regel ein Abschalten und Wiedereinschalten der Spannungsversorgung.

XCP über FlexRay: XCP-Botschaften sind sowohl im statischen als auch im dynamischen Segment eines FlexRay-Kommunikationszyklus zulässig (vgl. Abschn. 3.3). Die zugehörigen Zeitschlitze werden bei der Konfiguration des FlexRay-Bussystems für XCP reserviert, wobei die beiden FlexRay-Kanäle unabhängig voneinander belegt werden dürfen. Zeitschlitze im statischen Segment müssen eindeutig einem Steuergerät zugeordnet werden, während Zeitschlitze im dynamischen Segment von mehreren Steuergeräten im Multiplex-Betrieb genutzt werden können. Dieses *Slot Multiplexing* erfolgt über den Zykluszähler der FlexRay-Botschaften. Auf diese Weise kann z. B. festgelegt werden, dass mehrere Steuergeräte denselben Zeitschlitz verwenden, wobei ein Steuergerät A nur in jedem zweiten Kommunikationszyklus sendet, während zwei andere Geräte B und C sich in den verbleibenden Zeitschlitzen abwechseln (Abb. 6.11).

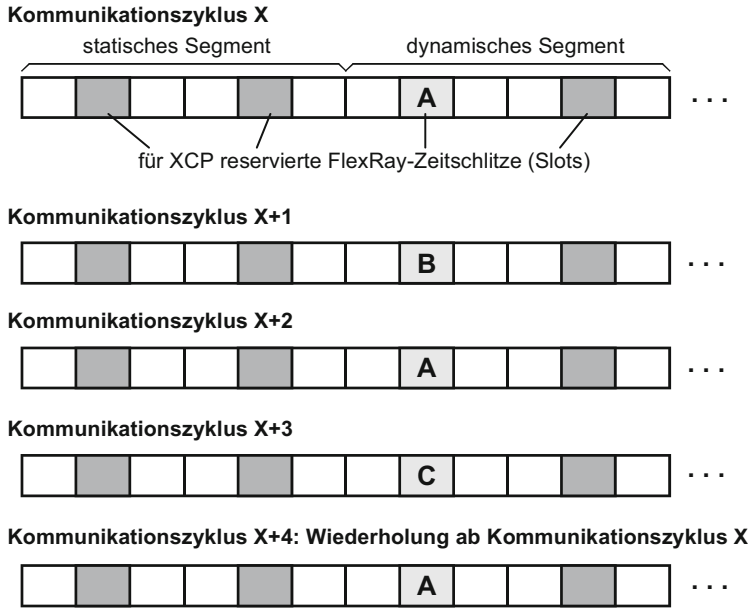

Abb. 6.11 Beispiel für die Reservierung von FlexRay-Zeitschlitzen (Slots) für XCP

XCP-Botschaften werden grundsätzlich nur ereignisgesteuert gesendet. Falls keine Daten zu senden sind, bleibt der entsprechende Zeitschlitz einfach leer. Die Lage, Länge und Zuordnung dieser Zeitschlitze wird mit Hilfe von AML/A2 L- oder FIBEX-Konfigurationsdateien (siehe Abschn. 6.2.3 bzw. 6.3) festgelegt und kann, soweit die Steuergeräte dies zulassen, vom Applikationssystem mit Hilfe der FlexRay-spezifischen XCP-Botschaften FLX_ASSIGN, FLX_ACTIVATE bzw. FLX_DEACTIVATE dynamisch konfiguriert und aktiviert werden. Wieweit dies tatsächlich genutzt werden kann, wird durch die verfügbaren FlexRay-Kommunikationscontroller jedoch erheblich eingeschränkt. Bei vielen derzeitigen Kommunikationscontrollern ist die Zuordnung von Zeitschlitzen und die Veränderung der Botschaftslänge nur nach einer Neuinitialisierung und damit einer Unterbrechung der Netzwerkkommunikation möglich, so dass die notwendige Bandbreite für XCP-Anwendungen oft dauerhaft reserviert werden muss, obwohl sie eigentlich nur kurzzeitig benötigt wird. Eine ähnliche Problematik ergibt sich derzeit übrigens auch bei der Flash-Programmierung über FlexRay.

Zur eindeutigen Unterscheidung der Steuergeräte beim *Slot Multiplexing* erhält jedes Gerät eine netzweit eindeutige, 1 Byte lange Knotenadresse NAX (*Network Address for XCP*), die z. B. mit der physikalischen Steuergeräteadresse für die KWP 2000- bzw. UDS-Diagnose identisch sein kann. Bei den vom Applikationssystem gesendeten XCP-Botschaften dient die Knotenadresse zur Festlegung des Empfänger-Steuergerätes, bei den vom Steuergerät zum Applikationssystem gesendeten XCP-Botschaften zur Identifikation des Senders. Die Knotenadresse NAX und einige weitere Steuerbytes werden in einem

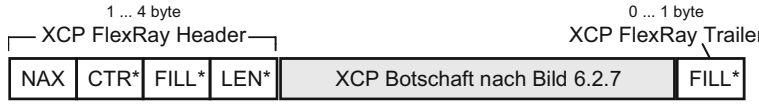

Abb. 6.12 XCP on FlexRay Botschaftsformat (* optionale Bytes)

XCP FlexRay Header gesendet (Abb. 6.12), dem dann die eigentliche XCP-Botschaft mit dem üblichen Format nach Abb. 6.9 folgt. Optional kann im Header ein Zählbyte CTR (*Counter*) mitgesendet werden, mit dem Applikationssystem und Steuergeräte ihre jeweils gesendeten Botschaften unabhängig voneinander durchnummerieren können, so dass der Empfänger fehlende Botschaften leichter erkennen kann. Ebenfalls optional ist ein Längenbyte LEN (*Length*), das die Länge der folgenden, eigentlichen XCP-Botschaft nach Abb. 6.9 angibt. Dies ist vor allem dann sinnvoll, wenn die Länge der XCP-Botschaften dynamisch variiert.

Der Header muss gegebenenfalls durch Füll-Bytes aufgefüllt werden, falls der Kommunikationscontroller beispielsweise aus Geschwindigkeitsgründen nur Datenblöcke verarbeiten kann, die 2 oder 4 Byte lang sind. Dasselbe gilt für die gesamte Botschaft. Da FlexRay-Botschaften immer nur eine geradzahlige Anzahl von Bytes enthalten dürfen, muss die Botschaft eventuell mit einem Füll-Byte am Ende (XCP FlexRay Trailer) aufgefüllt werden. Der Inhalt der Füll-Bytes ist beliebig. Welches Header-Format jedes Steuergerät verwendet, wird bei der XCP Konfiguration eindeutig festgelegt.

Bei Bedarf können mehrere XCP Botschaften (von einem Sender zu genau einem Empfänger) innerhalb derselben FlexRay-Botschaft zusammengefasst werden. Dabei enthält lediglich die erste XCP-Botschaft den vollständigen Header. Bei den weiteren XCP-Botschaften wird nur noch das LEN-Byte gesendet.

6.2.3 AML-Konfigurationsdateien für XCP und CCP

Die CCP- und XCP-Standards definieren Konfigurationsdateien, mit denen die Fähigkeiten und Parameter eines konkreten Steuergerätes beschrieben werden können. Bei den Beschreibungsdateien handelt es sich um Textdateien, die eine Syntax verwenden, die als *ASAP2/ASAM Meta Language AML* bezeichnet wird. Die Konfigurationsdateien werden in der Regel mit der Endung *.A2 L* gespeichert.

Die Beschreibung erfolgt hierarchisch in aufeinander folgenden oder verschachtelten/`begin` …/`end` Blöcken (Tab. 6.2). Ein hierarchisch übergeordneter Block definiert allgemeine Projektdaten, z. B. Name und Versionsnummer des Projekts. Auf einer tieferen Ebene folgt eine Definition der vom Steuergerät unterstützten optionalen XCP-Befehle und getrennt davon der für das konkrete Bussystem erforderlichen Parameter wie z. B. der CAN-Identifier.

Tab. 6.2 Prinzip der XCP-Konfigurationsdateien

```
/begin PROJECT XCP
  "Projekt 28147891"    /* Allgemeine Angaben zum Projekt */
  /begin HEADER
    "EDC 21 for Customer X Model Y"
    VERSION   "V3.1"
    . . .
  /end HEADER
  /begin MODULE
    /begin A2ML                /* AML Header-Dateien*/
      /include XCP_Common.aml
      /include XCP_on_CAN.aml
    /end A2ML
    /begin IF_DATA XCP
      /begin PROTOCOL_LAYER    /* Protokollkonfiguration */
        BYTE_ORDER_MSB_FIRST
        ADDRESS_GRANULARITY_BYTE
        OPTIONAL_CMD GET_ID
        OPTIONAL_CMD SET_REQUEST
        . . .
      /end PROTOCOL_LAYER
        . . .
      /begin XCP_ON_CAN        /* Bussystem-Konfiguration */
        0x0100                 /* XCP on CAN version 1.0 */
        CAN_ID_BROADCAST 0x0100 /* Broadcast CAN Id */
        CAN_ID_MASTER 0x0200    /* Application Sys. CAN Id*/
        CAN_ID_SLAVE 0x0300     /* ECU CAN Id */
        BAUDRATE 500000         /* CAN Baudrate */
        . . .
      /end XCP_ON_CAN
    /end IF_DATA
  /end MODULE
/end PROJECT
```

Im Sinne der Hierarchie ist es möglich, in einem hierarchisch höher stehenden Block Defaultwerte für bestimmte Parameter anzugeben, die dann in einem niedriger stehenden Block überschrieben werden.

Sinnvollerweise werden die Konfigurationsdaten für den Bussystem-unabhängigen und für den Bussystem-abhängigen Teil in separate Dateien ausgelagert, die mit Hilfe von/`include` Anweisungen in die zentrale Konfigurationsdatei eingebunden werden.

Welche Parameter konfiguriert werden können, die dabei zu verwendeten Schlüsselworte sowie die zulässigen Werte sind in Definitionsdateien festgelegt, die eine C-artige Syntax verwenden und häufig mit der Endung *.AML* gespeichert werden (Tab. 6.3). Auf diese Weise kann mit Hilfe eines Softwarewerkzeuges sowohl eine Bedienoberfläche geschaffen werden, die nur diejenigen Parameter und Optionen zur Konfiguration anbietet, die zu verändern sind, als auch eine automatisierte Überprüfung der Syntax der Konfigurationsdateien durchgeführt werden.

AML stammt noch aus einer Zeit, in der solche Pseudo-Textdateien mit anwendungsabhängig definierter Struktur und Syntax zur Konfiguration von Software allgemein üblich

Tab. 6.3 Ausschnitt aus der Definitionsdatei für XCP über CAN

```
/* A2ML data type description */
/* uchar    unsigned 8 Bit */
/* ulong    unsigned integer 32 Bit */
. . .
/*************** start of CAN *************/
struct CAN_Parameters
{ uint;          /* XCP on CAN version e.g. "1.0" = 0x0100 */
  taggedstruct
  { "CAN_ID_BROADCAST" ulong;    /* Broadcast CAN Id */
    "CAN_ID_MASTER"    ulong;    /* Application Sys. CAN Id*/
    "CAN_ID_SLAVE"     ulong;    /* ECU CAN Id */
    "BAUDRATE"         ulong;    /* CAN Baudrate */
    . . .
  };
  . . .
};
/*************** end of CAN *************/
```

waren. Leider führte dies auch im Fall von AML zu einer individuellen Lösung und erfordert daher speziell für AML entwickelte Werkzeuge. Die Erstellung mit einem einfachen Texteditor ist zwar theoretisch möglich, in der Praxis aber zu aufwendig und fehlerträchtig. Mittlerweile bietet die Informatik mit XML (Extended Markup Language, siehe Kasten unten) ein verallgemeinertes Grundkonzept für solche Beschreibungs- und Konfigurationsdateien an. Für deren Erstellung und Überprüfung existieren gute Werkzeuge, die eine höhere Verbreitung besitzen und daher ausgereifter und zukunftssicherer sein dürften. Aus diesem Grund ist zu erwarten, dass die XCP-Definitionsdateien zukünftig als XML Schema vorliegen und die Konfigurationsdateien in XML erstellt werden. Ein möglicher Kandidat dafür ist das *Calibration Data Format CDF* (siehe Abschn. 6.5.2), ergänzt durch FIBEX für die Buskommunikation. Es kann davon ausgegangen werden, dass die Werkzeughersteller Konvertierungswerkzeuge anbieten werden, die die Umsetzung von AML-Dateien nach XML automatisiert durchführen und für eine Übergangszeit eine Koexistenz beider Formate zulassen werden.

6.2.4 Interface zwischen Bus und Applikationssystem ASAM MCD 1b

Um die darüberliegenden Schichten des Applikationssystems vom darunterliegenden Treiber für das Busprotokoll (CCP, XCP, …) zu entkoppeln (vgl. Abb. 6.3), wird eine Zwischenschicht mit standardisierten Funktionen eingesetzt (Abb. 6.13).

Ein Messvorgang wird durch Aufruf der Funktion INIT_READ() initialisiert, die Messung durch SYNC() gestartet. Das Lesen der Messwerte erfolgt durch ein oder mehrfachen Aufruf der Funktion READ(). Mit STOP() wird die Messung wiederum beendet. Der Zugriff auf Applikationsparameter eines Steuergerätes bei einem Kalibriervorgang wird mit INIT_ACCESS() eingeleitet und mit ACCESS() ausgeführt. In beiden Fällen wird der Vor-

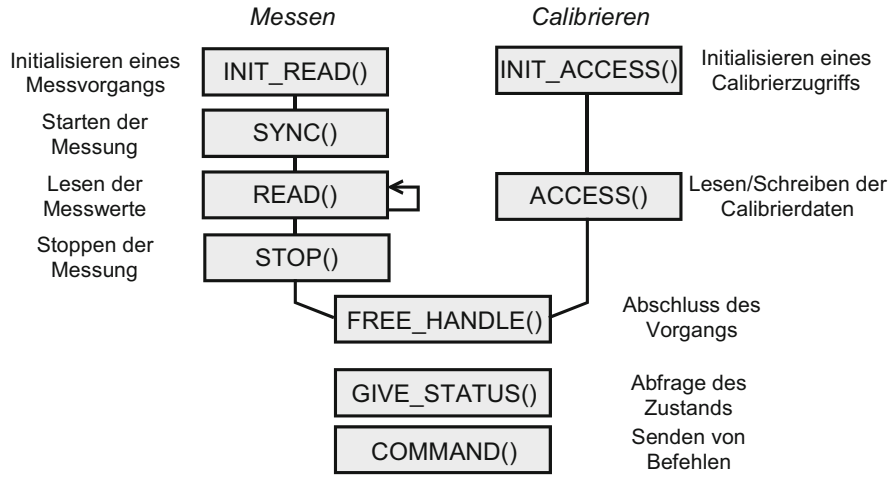

Abb. 6.13 Standardisierte Funktionen der ASAM MCD 1b Schnittstelle

gang mit der Funktion FREE_HANDLE() abgeschlossen. Mit Hilfe der Funktion COMMAND() können jederzeit herstellerspezifische Befehle an das Steuergerät gesendet werden, z. B. zur Auswahl einer Speicherseite oder zum Zurücksetzen des Steuergerätes. Alle Funktionen erwarten einen komplexen Parametersatz, der über verkettete Strukturen übergeben wird. Die Mehrzahl dieser Parameter, z. B. die Speicheradressen, auf die im Steuergerät zugegriffen werden soll, oder die Größen und das Datenformat der zugehörigen Variablen erhält der ASAM MCD 1-Treiber aus einer ASAP2-Konfigurationsdatei (siehe Abschn. 6.2.3).

Bei neueren Werkzeugen wird die MCD 1b-Schnittstelle voraussichtlich durch die MVCI/D-PDU-Schnittstelle abgelöst werden, die in Abschn. 6.8 beschrieben wird.

Extensible Markup Language XML & Unified Modeling Language UML

XML ist ein Mitte der 90er Jahre zunächst für Internet-Anwendungen vorgeschlagenes Textformat, mit dem Informationen strukturiert werden können (Abb. 6.14).

Die in XML-Dateien enthaltene Information wird durch so genannte Tags strukturiert. Tags sind in `<...>` geschriebene, vordefinierte Bezeichner, z. B. `<Bussysteme>`, die kennzeichnen, welche Art von Information folgt. Die Information selbst, als Wert des Elements bezeichnet, steht hinter dem in `<...>` geschriebenen Bezeichner und wird mit dem in `</...>` wiederholten Bezeichner abgeschlossen, wie z. B. der Name `PowertrainCAN` des CAN-Bussystems in Abb. 6.14. Bei hierarchisch untergeordneten Informationen werden die Tags verschachtelt, z. B. `<Bussysteme>` und `<Bussystem>`. Bei hierarchisch gleichwertigen Informationen werden die Tags einfach aneinander gereiht, z. B. `<Bussysteme>` und

<Steuergeräte>, wobei dasselbe Tag auch mehrfach wiederholt werden darf. Auf der obersten Hierarchieebene muss und darf es genau ein Element geben, das sogenannte Wurzelelement, im Beispiel <Fahrzeug>. Darunter können Elemente beliebig aneinander gereiht oder geschachtelt werden. Optisch wird die Verschachtelung durch Einrückungen der Elemente in der Datei visualisiert.

```
<Fahrzeug>
   <name> AudiVolksPorscheBenzMotorenWagen T520 </name>
   <Bussysteme . . . >
      <Gateway-Ref ID="ecu1" />
      <Bussystem ID="bus1" type="CAN" . . . >
         <name> PowertrainCAN </name>
         . . .
      </Bussystem>
      <Bussystem ID="bus2" type="LIN" . . . >
         <name> LeftDoorLIN </name>
         <Gateway-Ref ID="ecu5" />
         . . .
      </Bussystem>
   </Bussysteme>
   . . .
   <Steuergeräte . . . >
      <Steuergerät ID="ecu1" . . . >
         <name> EngineECU </name>
         <Bussystem-Ref ID-REF="bus1" />
         . . .
      </Steuergerät>
      . . .
      <Steuergerät ID="ecu5" . . . >
         <name> DashboardECU </name>
         <Bussystem-Ref ID-REF="bus1" />
         <Bussystem-Ref ID-REF="bus2" />
      </Steuergerät>
   </Steuergeräte>
</Fahrzeug>
```

Abb. 6.14 Beispiel einer XML-Datei

Zusätzlich zu den Werten der Elemente können den Tags auch so genannte Attribute direkt zugeordnet werden, die einen Namen und einen Wert aufweisen. In solchen Attributen können weitere Informationen untergebracht werden, vorzugsweise Sortier- und Klassifikationswerte, im Beispiel etwa ID="bus1" oder type="CAN".

Die Beschreibung des Datenmodells, das den XML-Konfigurationsdateien zugrunde liegt, erfolgt in neueren Spezifikationen wie FIBEX oder ODX, die im folgenden beschrieben werden, häufig in einer UML-artigen Darstellung (Abb. 6.15). Die **Unified Modeling Language UML** ist eine standardisierte grafische Notation, mit der objektorientierte Softwaresysteme beschrieben werden. Zu den wesentlichen Konzepten, die dabei immer wieder auftauchen, gehören Klassen und

Objekte sowie deren Beziehungen untereinander. Im dargestellten Beispiel enthält das Objekt `Fahrzeug` ein oder mehrere Objekte der Klassen `Bussystem` und `Steuergerät`. Die Zuordnung „*enthält*“ wird in der Informatik als *Aggregation* bezeichnet und in der UML-Notation mit einer Raute an der Verbindungslinie zwischen den Objekten dargestellt. Da ein Steuergerät an mehrere Bussysteme angeschlossen sein kann, wird die Beziehung zwischen `Steuergerät` und `Bussystem` nicht durch eine *Aggregation* sondern durch eine *Assoziation*, d. h. in der UML-Grafik eine einfache Linie zwischen zwei Objekten, dargestellt. Die zulässige Anzahl der Objekte wird durch *Kardinalitäts-* oder *Multiplizitätszahlen* beschrieben, wobei im Beispiel `1` Fahrzeug eine beliebige Anzahl `0..*` von Bussystemen bzw. Steuergeräten enthalten kann, während `1` Bussystem mindestens zwei oder mehr (`2..*`) Steuergeräte verbindet. Bussysteme existieren in verschiedenen Ausprägungen, z. B. CAN und FlexRay, die einige logische Eigenschaften teilen, z. B. die Bitrate, aber auch unterschiedliche Eigenschaften haben, z. B. *Sync Frames*, die es nur bei FlexRay gibt. In der Informatik spricht man dabei von *Eltern-Kind-Beziehung*, wobei in der UML-Notation die *Eltern*-Seite durch ein Dreieck gekennzeichnet wird.

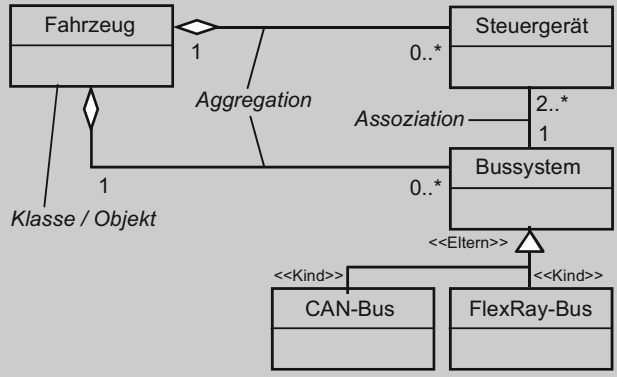

Abb. 6.15 UML-Darstellung von Objektbeziehungen

Bei der Umsetzung solcher UML-Datenmodelle in die XML-Beschreibung werden *Aggregationen* in der Regel durch ineinander verschachtelte XML-Elemente, *Assoziationen* durch Referenzen auf andere Objekte abgebildet. In Abb. 6.14 etwa verweist der Eintrag für das Steuergerät `DashboardECU` über das Tag `<Bussystem-Ref>` auf das Bussystem `bus2`. Auch das Konzept der *Vererbung* von Eigenschaften wird in XML abgebildet. So wird auf der übergeordneten Ebene der `Bussysteme` in Abb. 6.14 mit `Gateway-Ref ID="ecu1"` ein Steuergerät als Default-Gateway festgelegt. Dieses wird von allen hierarchisch untergeordneten Bussen verwendet (*geerbt*), die nicht selbst eine eigene Gateway-Definition enthalten. Das untergeordnete Bussystem `bus2` dagegen wiederholt dieses Tag

`Gateway-Ref`, verweist aber für diesen Bus auf das Steuergerät `ecu5` als Gateway.

XML ist primär ein Textformat, das selbstverständlich auch Zahlen in Textform enthalten kann. Aber auch binäre Informationen, z. B. compilierter Programmcode, lassen sich aus einer XML-Beschreibung heraus als externe Datei referenzieren. Die Auswertung der binären Information oder deren direkte Bearbeitung in XML dagegen ist nicht möglich und erfordert externe Werkzeuge.

Die Syntax von XML hat einige Ähnlichkeit mit der Auszeichnungssprache HTML, mit der die Informationen im Internet dargestellt werden. Dort werden Überschriften verschiedener Ebenen z. B. mit Tags wie `<H1>...</H1>` oder Textpassagen mit `<P>...</P>` gekennzeichnet. Attribute wie `font="Helvetica"` werden für die Festlegung von Schriftarten oder Schriftgrößen verwendet. Im Unterschied zu HTML, bei dem die Tags vor allem die optische Darstellung der Information festlegen, beschreiben die Tags bei XML vor allem die inhaltliche Bedeutung der Information. Weit entscheidender ist jedoch, dass die Tags und ihre Bedeutung bei HTML unveränderlich vorgegeben sind, während sie bei XML frei definierbar sind. Die Definition erfolgt in der Regel in einem so genannten **XML-Schema**. Dort wird festgelegt, welche Tags in einem Dokument an welcher Stelle und in welcher Häufigkeit verwendet werden können und müssen, welche Attribute ein Tag haben darf und welche Art von Werten (Text, Zahlen, Wertebereich usw.) für die Elemente und Attribute zulässig ist. Erst in Verbindung mit einer derartigen, für eine bestimmte Aufgabenstellung standardisierten XML-Schema-Definition sind Daten auf XML-Basis wirklich austauschbar.

Obwohl XML ein textbasiertes Format und damit theoretisch mit einfachsten Mitteln lesbar ist, werden XML-Dokumente schnell unübersichtlich und bei Änderungen fehleranfällig. In der Praxis sind daher stets geeignete Werkzeuge notwendig, die den Anwender bei der Suche nach Informationen unterstützen und bei der Bearbeitung eine sofortige Überprüfung durchführen. Insoweit unterscheiden sich in XML erstellte Beschreibungen für den Anwender nicht von Beschreibungen in AML, CanDB, OIL oder anderen Industriestandards. Der wesentliche Unterschied zu diesen proprietären Formaten ist, dass es für die Toolhersteller bei XML leichter ist, solche Werkzeuge zu erstellen, weil es für XML bereits viele Softwarekomponenten aus anderen Bereichen der Informatik gibt, die verhältnismäßig einfach in die Werkzeuge für den Automobilbereich integriert werden können.

6.3 Field Bus Exchange Format FIBEX

Mit der FIBEX-Spezifikation versucht ASAM, ein einheitliches XML-Beschreibungsformat für die Kommunikation über die verschiedenen Bussysteme von CAN über LIN, FlexRay bis zu MOST und Erweiterungsmöglichkeiten für weitere Busse zu definieren. FIBEX soll die bisher üblichen proprietären Formate wie CANdb (DBC Dateien) oder LIN Configuration Language und Node Capability File (LDF und NCF Dateien) mittelfristig ablösen. Die schnellste Einführung dieses Formats erfolgte bei FlexRay, da dafür noch kein anderes Beschreibungsformat etabliert war.

Der Schwerpunkt von FIBEX liegt auf der Beschreibung der On-Board-Kommunikation im normalen Fahrzeugbetrieb und soll in Spezifikations-, Test- und Simulationswerkzeugen verwendet werden. Wie bei vielen anderen derartigen Standardisierungsbestrebungen bleibt aber auch bei FIBEX seltsam unklar, ob es sich wirklich um einen universellen Standard handeln wird, der von allen Werkzeugherstellern unterstützt wird. Prinzipiell wäre es wohl möglich, etwa die Beschreibung von CCP und des ebenfalls vom ASAM-Konsortium spezifizierten XCP in FIBEX zu integrieren, wenn dort entsprechende Erweiterungen oder eine klar definierte Schnittstelle vorgesehen würden. Dies ist genauso wenig erkennbar wie die Antwort auf die Frage, welche Bedeutung FIBEX behalten wird, wo AUTOSAR (Kap. 8) doch erheblich mächtigere Dateiformate definiert hat, aber eben doch nicht alle Informationen abdeckt, die in FIBEX festgehalten werden können.

Die Syntax der FIBEX-Beschreibungsdateien ist in einem frei verfügbaren XML-Schema allgemein spezifiziert, wobei die Einbeziehung herstellerspezifischer Erweiterungen ausdrücklich vorgesehen ist. Die wesentlichen Elemente der hierarchischen XML-Baumstruktur sind in Abb. 6.16 dargestellt. Im Hauptelement ELEMENTS werden einerseits die Struktur des Netzes, d. h. die verwendeten Bussysteme (CLUSTERS, CHANNELS) sowie Steuergeräte (ECUS, GATEWAYS), und andererseits die in diesem Netz ablaufende Kommunikation, d. h. die Botschaften (FRAMES) und deren Dateninhalt (SIGNALS), beschrieben. In der FIBEX-Nomenklatur bezeichnet der Begriff CLUSTER ein einzelnes Bussystem, das etwa bei FlexRay aus mehreren parallelen Kanälen (CHANNELS) bestehen kann, sowie den daran angeschlossenen Steuergeräten (ECUS), von denen einige auch die Funktion eines GATEWAYS zu einem der anderen Bussysteme übernehmen können. Mit FIBEX 3.0 wurden als neues Element sogenannte *Protocol Data Units* PDUS als Zwischenschicht zwischen FRAMES und SIGNALS eingeführt, um die zentralen Datenelemente des AUTOSAR-Protokollstapels (vgl. Kap. 8) abzubilden. Da FIBEX 2.0 aber bereits eingesetzt wurde und die Änderungen nicht vollständig kompatibel sind, wird zunächst das ältere Format beschrieben und anschließend auf die Änderungen eingegangen.

Als Beispiel für eine FIBEX-Beschreibung wird das in Abb. 6.17 dargestellte System betrachtet, bei dem zwei Busse über ein Gateway-Steuergerät miteinander verbunden sind. Dabei soll das linke Steuergerät über das Gateway eine Botschaft an das rechte Steuergerät senden. Bei beiden Bussen wird CAN eingesetzt. Jedes Bussystem hat jeweils einen Kanal. Die Bussysteme sowie die daran angeschlossenen Steuergeräte werden in der FIBEX-Beschreibung (Tab. 6.4) als CLUSTER bezeichnet (hier clMotorCAN {1} bzw. clKarosse-

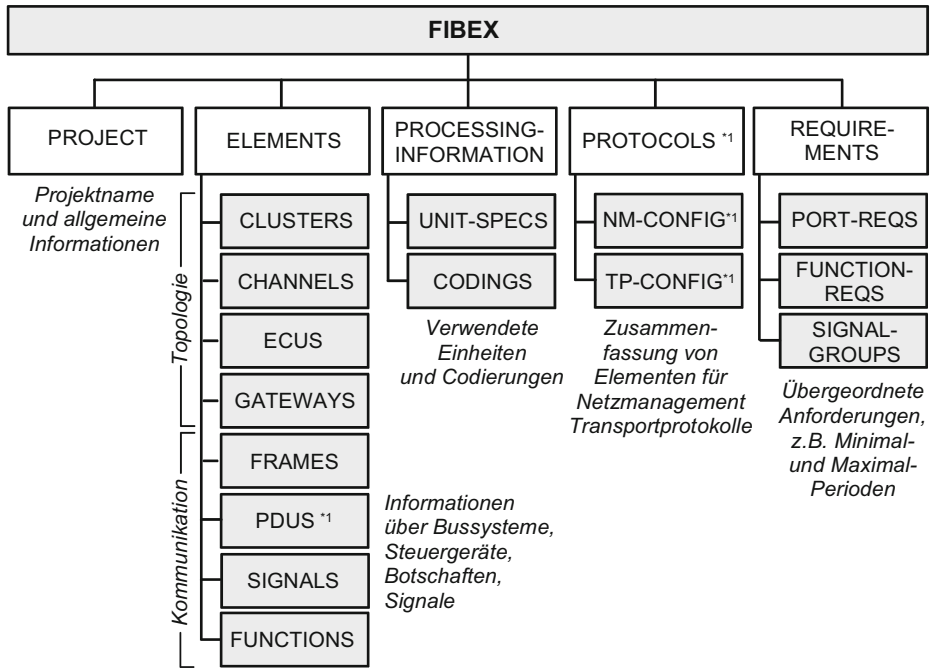

Abb. 6.16 Hauptelemente einer FIBEX-Beschreibungsdatei; *1 neu seit FIBEX 3.0

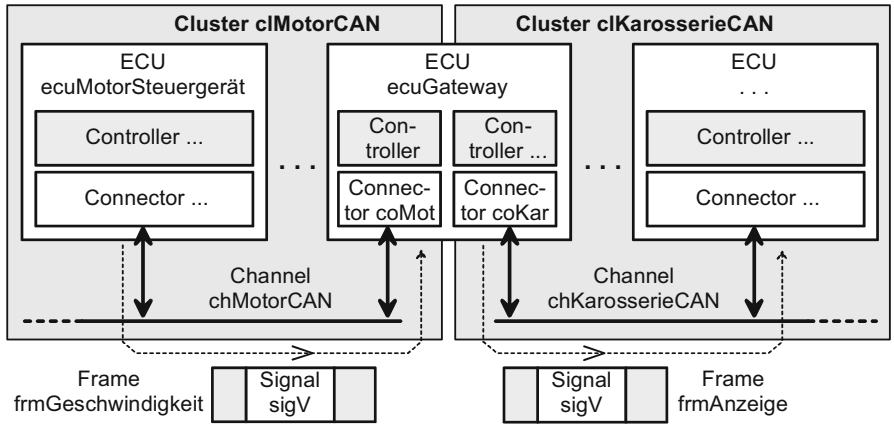

Abb. 6.17 Beispielsystem mit zwei über ein Gateway gekoppelten CAN-Bussystemen

rieCAN {2}). Die hier angegebenen Nummern {1}, {2} usw. beziehen sich auf die entsprechenden Stellen in den XML-Dateien.

Im CLUSTER-Abschnitt wird unter PROTOCOL {3} der Typ des Bussystems und unter SPEED dessen Bitrate aufgelistet. Unter CHANNEL-REFS {4} wird auf den zugehö-

Tab. 6.4 FIBEX-Beschreibungsdatei des Systems nach Abb. 6.17 – Teil 1

Beschreibung der Netzstruktur (CLUSTERS, CHANNELS)

```
<?xml version="1.0"?>

<FIBEX xmlns="http://www.asam.net/xml/fbx" . . . VERSION="2.0.0b">

  <fx:PROJECT>
    <ho:SHORT-NAME>CAN_Beispiel</ho:SHORT-NAME>
    <ho:DESC>Beispielprojekt . . . </ho:DESC>
  </fx:PROJECT>

  <fx:ELEMENTS>
    <fx:CLUSTERS>
      <fx:CLUSTER ID="clMotorCAN">                                    {1}
        <ho:SHORT-NAME>Motorbus</ho:SHORT-NAME>
        <ho:DESC>Motorseitiger High Speed CAN Bus</ho:DESC>
        <fx:PROTOCOL xsi:type="can:PROTOCOL-TYPE">CAN</PROTOCOL>      {3}
        <fx:SPEED>500000</fx:SPEED>
        . . .
        <fx:CHANNEL-REFS>
          <fx:CHANNEL-REF ID-REF="chMotorCAN"/>                       {4}
        </fx:CHANNEL-REFS>
      </fx:CLUSTER>

      <fx:CLUSTER ID="clKarosserieCAN">                               {2}
        <ho:SHORT-NAME>Karosseriebus</ho:SHORT-NAME>
        . . .
      </fx:CLUSTER>
    </fx:CLUSTERS>

    <fx:CHANNELS>
      <fx:CHANNEL ID="chMotorCAN">                                    {5}
        . . .
        <fx:FRAME-TRIGGERINGS>
          <fx:FRAME-TRIGGERING ID="ftgGeschwindigkeit">
            <fx:IDENTIFIER>
              <fx:IDENTIFIER-VALUE>125</fx:IDENTIFIER-VALUE>
            </fx:IDENTIFIER>
            <fx:FRAME-REF ID-REF="frmGeschwindigkeit"/>                {7}
          </fx:FRAME-TRIGGERING>
        </ fx:FRAME-TRIGGERINGS>
      </ fx:CHANNEL>

      <fx:CHANNEL ID="chKarosserieCAN">                               {6}
        . . .

        <fx:FRAME-TRIGGERINGS>
          <fx:FRAME-TRIGGERING ID="ftgAnzeige">
            <fx:IDENTIFIER>
              <fx:IDENTIFIER-VALUE>32</fx:IDENTIFIER-VALUE>
            </fx:IDENTIFIER>
            <fx:FRAME-REF ID-REF="frmAnzeige"/>                        {8}
          </fx:FRAME-TRIGGERING>
        </fx:FRAME-TRIGGERINGS>
      </fx:CHANNEL>
    </fx:CHANNELS>
```

rigen CHANNEL (hier chMotorCAN {5} bzw. chKarosserieCAN {6}) verwiesen. Unter CHANNEL wird aufgeführt, welche Botschaften über den jeweiligen Bus übertragen werden (FRAME-REF, hier frmGeschwindigkeit {7} bzw. frmAnzeige {8}) und welche CAN-Identifier verwendet werden.

Die Steuergeräte werden im Abschnitt ECUS aufgelistet (Tab. 6.5). Jedes Steuergerät {9} hat einen oder mehrere *Stecker* (CONNECTOR) {10, 11}, die unter CHANNEL-REF {12, 13} auf das Bussystem verweisen, an das das Gerät über diesen *Stecker* angeschlossen ist. Hier wird unter OUTPUTS – FRAME-TRIGGERING-REF bzw. INPUTS – SIGNAL-INSTANCE-REF auch angegeben, welche Botschaften von diesem Steuergerät gesendet bzw. empfangen werden. Im Beispielsystem empfängt das Gateway-Steuergerät das Signal sigV {14} vom Motor-CAN (das Teil der Botschaft frmGeschwindigkeit ist, siehe unten) und sendet eine Botschaft ftgAnzeige {15}.

Anstatt auf die Botschaften im CONNECTOR-Abschnitt direkt zu verweisen, ist alternativ ein hier nicht verwendeter FUNCTION-Abschnitt möglich. Im Unterabschnitt CONTROLLER {16} können der Typ des eingesetzten Kommunikationscontrollers und dessen Konfigurationsparameter aufgeführt werden. Steuergeräte, die eine Gateway-Funktion zwischen verschiedenen Bussystemen haben, werden im Abschnitt GATEWAY unter ECU-REF referenziert {17}.

Die Details der Botschaften werden im Abschnitt FRAME beschrieben (Tab. 6.6). Hier wird angegeben, wie lange eine Botschaft ist (BYTE-LENGHT), aus welchen Signalen sie sich zusammensetzt (SIGNAL-INSTANCE, SIGNAL-REF) und an welcher Bitposition (BIT-POSITION) innerhalb der Botschaft die Signale jeweils beginnen. Die Botschaft frmGeschwindigkeit {18} im Beispiel ist zwei Byte lang {19} und enthält ab Bitposition 0 {20} in Little-Endian-Reihenfolge das Signal sigV {21, 22} mit der Codierung CODING-speed {23}.

Diese Codierung wird im Abschnitt CODING {24} beschrieben (Tab. 6.7), wobei im Beispiel als Einheit (UNIT-REF {25} und UNIT-SPEC {26}) für die Fahrgeschwindigkeit km/h verwendet wird und ein linearer Zusammenhang {27} zwischen dem auf dem Bus übertragenen 16 bit-Wert und dem physikalischen Wert mit 0 als kleinstem und 400 km/h als größtem Wert festgelegt wird {28}.

Wie man bereits an diesem einfachen Beispiel sieht, ist die fehlerfreie Erstellung und Pflege einer derartigen Konfigurationsdatei ohne die Hilfe eines geeigneten Werkzeuges, das entsprechende Eingabemasken bereitstellt, die Struktur des Bussystems sowie die Verweise zwischen den einzelnen Elementen visualisiert und die Konsistenz der Daten sicherstellt, nahezu unmöglich.

Seit Version 2.0 des FIBEX-Standards wird erstmals ein XML-Schema für Time-Triggered CAN bereitgestellt. Als wesentliche Erweiterung wird nun auch MOST offiziell unterstützt. Die Dienst-orientierte Struktur der MOST *Network Services* (siehe Abschn. 3.4) mit ihrer Objekthierarchie zwang zu einigen Erweiterungen des bisherigen Datenschemas im Vergleich zu der Signal-orientierten Beschreibung von CAN, LIN und FlexRay. Ein MOST-Funktionsblock wird in FIBEX als `FBlock` abgebildet, der die zugeordneten MOST-Funktionen als FIBEX-`Function` enthält. Die von MOST bekannten Kennziffern

Tab. 6.5 FIBEX-Beschreibungsdatei des Systems nach Abb. 6.17 – Teil 2

Beschreibung der Steuergeräte (ECUS, GATEWAYS)

```
<fx:ECUS>
  <fx:ECU ID="ecuMotorSteuergerät">
    . . .
    <fx:CONNECTORS>
      <fx:CONNECTOR ID=". . .">
        <fx:CHANNEL-REF ID-REF="chMotorCAN"/>
        <fx:OUTPUTS ID=". . .">
          <fx:FRAME-TRIGGERING-REF ID-REF="ftgGeschwindigkeit"/>
        </fx:OUTPUTS>
      </fx:CONNECTOR>
    </fx:CONNECTORS>

    <fx:CONTROLLERS>
      <fx:CONTROLLER xsi:type="can:CONTROLLER-TYPE" ID="...">
        . . .                                                    {16}
        <fx:CHIP-NAME>SJA1000</fx:CHIP-NAME>
        . . .
      </fx:CONTROLLER>
    </fx:CONTROLLERS>
  </fx:ECU>

  <fx:ECU ID="ecuGateway">                                       {9}
    . . .
    <fx:CONNECTORS>
      <fx:CONNECTOR ID="coMot">                                  {10}
        <fx:CHANNEL-REF ID-REF="chMotorCAN"/>                    {12}

        <fx:INPUTS>
          <fx:INPUT-PORT ID=". . .">                             {14}
            <fx:SIGNAL-INSTANCE-REF ID-REF="Instance-sigV"/>
          </fx:INPUT-PORT>
        </fx:INPUTS>
      </fx:CONNECTOR>
      <fx:CONNECTOR ID="coKar">                                  {11}
        <fx:CHANNEL-REF ID-REF="chKarosserieCAN"/>               {13}
        <fx:OUTPUTS ID=". . .">
          <fx:FRAME-TRIGGERING-REF ID-REF="ftgAnzeige"/>         {15}
        </fx:OUTPUTS>
      </fx:CONNECTOR>
    </fx:CONNECTORS>
  </fx:ECU>
</fx:ECUS>
<fx:GATEWAYS>
  <fx:GATEWAY ID="gGateway">
    . . .
    <fx:ECU-REF ID-REF="ecuGateway"/>                            {17}
  </fx:GATEWAY>
</fx:GATEWAYS>
```

Tab. 6.6 FIBEX-Beschreibungsdatei des Systems nach Abb. 6.17 – Teil 3

Beschreibung der Botschaften (FRAMES, SIGNALS)

```
<fx:FRAMES>
  <fx:FRAME ID="frmGeschwindigkeit">                                {18}
    <ho:SHORT-NAME>Fahrgeschwindigkeit</ho:SHORT-NAME>
    <fx:BYTE-LENGTH>2</fx:BYTE-LENGTH>                              {19}
    <fx:FRAME-TYPE>APPLICATION</fx:FRAME-TYPE>
    <fx:SIGNAL-INSTANCES>
      <fx:SIGNAL-INSTANCE ID="Instance-sigV">
        <fx:BIT-POSITION>0</fx:BIT-POSITION>                        {20}
        <fx:IS-HIGH-LOW-BYTE-ORDER>true</IS-HIGH-LOW-BYTE-ORDER>
        <fx:SIGNAL-REF ID-REF="sigV"/>                              {21}
      </fx:SIGNAL-INSTANCE>
    </fx:SIGNAL-INSTANCES>
  </fx:FRAME>

  <fx:FRAME ID="frmAnzeige">
    . . .
  </fx:FRAME>
</fx:FRAMES>
<fx:SIGNALS>
  <fx:SIGNAL ID="sigV">                                             {22}
    . . .
    <fx:CODING-REF ID-REF="CODING-speed"/>                          {23}
  </fx:SIGNAL>
</fx:SIGNALS>
</fx:ELEMENTS>
```

`FBlckID`, `FktID` usw. tauchen in der FIBEX-Beschreibung als XML-Elemente wieder auf. Der MOST `OPType`, der präzise angibt, welche MOST-Funktion im Detail ausgeführt wird und letztlich auf eine MOST-Steuerdaten-Botschaft abgebildet wird, referenziert schließlich einen FIBEX `Frame`, d. h. die bekannte Datenstruktur für Busbotschaften. Zur Darstellung der Parameter der MOST-Funktionen, d. h. der „Signale“ in den MOST-Botschaften, wurden einige neue Typen eingeführt, unter anderem `Array` und `Stream`.

Zu den weiteren Änderungen ab Version 2.0 gehört die durchgängige Verwendung von XML-Namensräumen, wobei einige bisherige Namensräume umbenannt wurden. Für die XML-Beschreibung von Gateways wurden einige Datenelemente umorganisiert und das Mapping von Signalen zwischen den verschiedenen Bussystemen in der Dokumentation ausführlicher dargestellt. Darüber hinaus wurde die Darstellung aller Datenstrukturen in der Dokumentation vollständig auf UML-Diagramme umgestellt und damit an die Form der ODX-Dokumentation angepasst.

Die Austauschbarkeit der Dateninhalte mit AUTOSAR war Hauptziel der FIBEX Version 3.0. Dazu wurden die *Protocol Data Units* Elemente als Zwischenschicht zwischen FRAMES und SIGNALS eingeführt. Diese Änderung ist nicht vollständig rückwärts kompatibel zur Version 2.0. Eine FIBEX PDU entspricht der *Interaction Layer I-PDU* der AUTOSAR COM Schicht, während ein FIBEX FRAME praktisch der AUTOSAR *Data Link*

Tab. 6.7 FIBEX-Beschreibungsdatei des Systems nach Abb. 6.17 – Teil 4

Beschreibung der Signal-Einheiten und -Codierung (UNITS, CODINGS)

```
<fx:PROCESSING-INFORMATION xmlns="http://www.asam.net/xml">
  <ho:UNIT-SPEC>
    <ho:PHYSICAL-DIMENSIONS>
      <ho:PHYSICAL-DIMENSION ID="UNIT-speed">                    {26}
        <ho:SHORT-NAME>speed</ho:SHORT-NAME>
        <ho:DESC>Fahrgeschwindigkeit in km/h</ho:DESC>
      </ho:PHYSICAL-DIMENSION>
    </ho:PHYSICAL-DIMENSIONS>
  </ho:UNIT-SPEC>

  <fx:CODINGS>
    <fx:CODING ID="CODING-speed">                                {24}
      . . .
      <ho:CODED-TYPE ho:BASE-DATA-TYPE="A_UINT16" ...
                                              ENCODING="UNSIGNED">
         <ho:BIT-LENGTH>16</ho:BIT-LENGTH>
      </ho:CODED-TYPE>

      <ho:COMPU-METHOD>
        . . .
        <ho:UNIT-REF ID-REF="UNIT-speed"/>                       {25}
        <ho:CATEGORY>LINEAR</ho:CATEGORY>                        {27}
        <ho:COMPU-INTERNAL-TO-PHYS>
           <ho:COMPU-SCALES>
              <ho:COMPU-SCALE>
                 <ho:LOWER-LIMIT INTERVAL-TYPE="CLOSED">0        {28}
                 </ho:LOWER-LIMIT>
                 <ho:UPPER-LIMIT INTERVAL-TYPE="CLOSED">400
                 </ho:UPPER-LIMIT>
                 <ho:COMPU-CONST>
                    <ho:V>1.0</ho:V>
                 </ho:COMPU-CONST>
              </ho:COMPU-SCALE>
           </ho:COMPU-SCALES>
           <ho:COMPU-DEFAULT-VALUE>
              <ho:V>0.0</ho:V>
           </ho:COMPU-DEFAULT-VALUE>
        </ho:COMPU-INTERNAL-TO-PHYS>
      </ho:COMPU-METHOD>
    </fx:CODING>
  </fx:CODINGS>
</fx:PROCESSING-INFORMATION>
</fx:FIBEX>
```

Layer L-PDU entspricht, also der tatsächlich versendeten Busbotschaft (siehe Abschn. 8.4, insbesondere Abb. 8.12). Die I-PDU besteht aus einem oder mehreren Signalen, d. h. Steuergerätewerten wie z. B. der Motordrehzahl, der Kühlwassertemperatur oder der Fahrgeschwindigkeit aus dem obigen FIBEX-Beispiel. Bei CAN oder LIN, deren Busbotschaften maximal 8 Byte Nutzdaten enthalten können, kann eine I-PDU nur relativ wenige Signale enthalten. Dort wird die I-PDU praktisch direkt auf den Nutzdatenbereich der eigentlichen

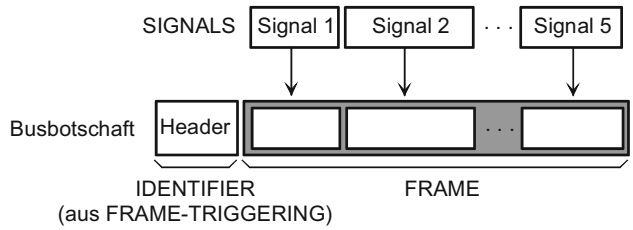

Abb. 6.18 Beschreibung einer Busbotschaft ab FIBEX 2.0

Busbotschaft, des FRAMES, abgebildet. Eine Zwischenebene war nicht wirklich notwendig. In dieser Weise war die Beschreibung bei FIBEX 2.0 aufgebaut (Abb. 6.18).

Bei FlexRay dagegen, dessen Botschaften bis zu 254 Byte lang sein können, werden aus Effizienzgründen möglicherweise mehrere I-PDUs im Nutzdatenbereich einer einzelnen Busbotschaft, einem FRAME, zusammengefasst. Um auf den oberen Ebenen weitgehend unabhängig vom eigentlichen Bussystem zu sein, wird die Abbildung von Signalen auf I-PDUs und von dort aus dann auf den Nutzdatenbereich von Busbotschaften bei AUTOSAR durchgängig bei allen Bussystemen verwendet. Dieses Konzept wurde ab FIBEX 3.0 übernommen (Abb. 6.19).

Die im obigen Beispiel notwendigen Änderungen zur Anpassung an FIBEX 3.0 sind überschaubar (Tab. 6.8). Im Wesentlichen wird im FRAMES-Abschnitt aus Tab. 6.6 der Begriff FRAME durch PDU ersetzt {29} und im neuen FRAMES-Abschnitt mit derselben Syntax auf die PDUs statt auf die Signale verwiesen {30}.

Das wahlweise Multiplexen von PDUs im selben Frame wird über die neuen Elemente PDU-MULTIPLEXING und das Zeitverhalten von PDUs über PDU-TRIGGERING und I-TIMING beschrieben, die im vorliegenden Beispiel nicht verwendet wurden.

Bei FIBEX-Version 3.1 wurde das Beschreibungsschema für LIN überarbeitet. Hauptneuerung aber war die Aufnahme weiterer AUTOSAR-Elemente. Damit können nun auch das Transportprotokoll AUTOSAR TP und das Netzmanagement AUTOSAR NM in FIBEX beschrieben werden.

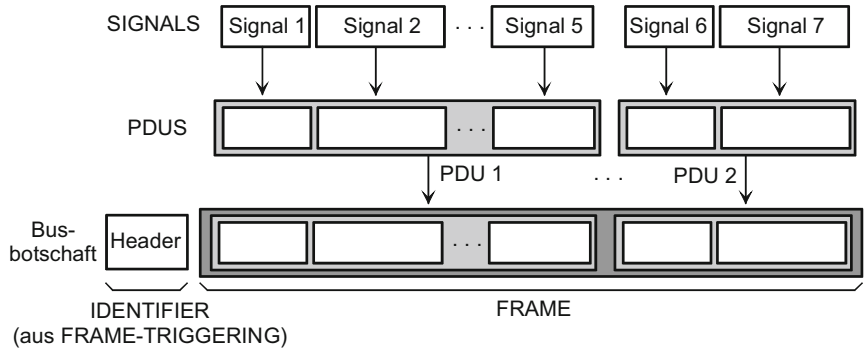

Abb. 6.19 Erweiterte Beschreibung einer Busbotschaft ab FIBEX 3.x

Tab. 6.8 Prinzipiell notwendige Anpassung des Beispiels an FIBEX 3.x

```
<FIBEX xmlns="http://www.asam.net/xml/fbx" . . . VERSION="3.0.0">
  . . .
  <fx:ELEMENTS>
    . . .
    <fx:PDUS>   entspricht dem alten FRAME-Abschnitt aus Tabelle 6.3.3   {29}
      <fx:PDU ID="pduGeschwindigkeit">
        <ho:SHORT-NAME>Fahrgeschwindigkeit</ho:SHORT-NAME>
        <fx:BYTE-LENGTH>2</fx:BYTE-LENGTH>
        <fx:PDU-TYPE>APPLICATION</fx:PDU-TYPE>
        <fx:SIGNAL-INSTANCES>
          <fx:SIGNAL-INSTANCE ID="Instance-sigV">
            <fx:SIGNAL-REF ID-REF="sigV"/>
             . . .
        </fx:SIGNAL-INSTANCES>
      </fx:PDU>
      . . .
    </fx:PDUS>

     <fx:FRAMES>          neuer FRAME-Abschnitt, verweist auf PDUs      {30}
       <fx:FRAME ID="frmGeschwindigkeit">
          <ho:SHORT-NAME>Fahrgeschwindigkeit</ho:SHORT-NAME>
          <fx:BYTE-LENGTH>2</fx:BYTE-LENGTH>
          <fx:FRAME-TYPE>APPLICATION</fx:FRAME-TYPE>
          <fx:PDU-INSTANCES>
             <fx:PDU-INSTANCE ID=. . .>
                <fx:PDU-REF ID-REF="pduGeschwindigkeit"/>
                <fx:BIT-POSITION>0</fx:BIT-POSITION>
                <fx:IS-HIGH-LOW-BYTE-ORDER> . . .
             </fx:PDU-INSTANCE>
          </fx:PDU-INSTANCES>
        </fx:FRAME>
        . . .
     </fx:FRAMES>
  <fx:/ELEMENTS>
```

Die Grundstruktur der FIBEX Beschreibung des in Abschn. 4.2 vorgestellten Transportprotokolls für FlexRay ist in Abb. 6.20 dargestellt. Im Abschnitt TP-CONFIG werden die FlexRay-Kanäle sowie die angeschlossenen Steuergeräte als *Knoten* definiert. Die für das Transportprotokoll vorgesehenen Botschaften im statischen bzw. dynamischen Segment werden über TP-PDU-USAGE bzw. bei Flow Control Botschaften über TP-FC-PDU-USAGE referenziert und als normale PDUs beschrieben.

Das AUTOSAR Netzmanagement, das in Abschn. 8.5 dargestellt wird, kann mittels der FIBEX Struktur nach Abb. 6.21 beschrieben werden. Die am Netzmanagement eines einzelnen Bussystems beteiligten Netzknoten werden im Abschnitt NM-CLUSTERS definiert, der über NM-PDU-USAGE auf die Beschreibung der zugehörigen PDUs verweist. Für Gateways sind die Abschnitte NM-CLUSTER-COUPLING bzw. NM-COORDINATOR vorgesehen, falls das Netzmanagement übergreifend über mehrere Bussysteme erfolgt und koordiniert wird.

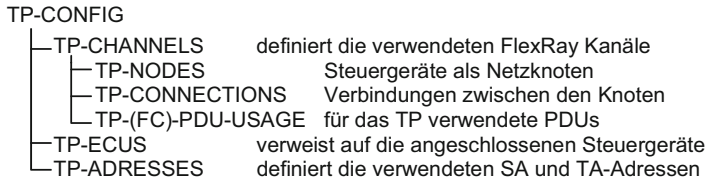

Abb. 6.20 Struktur der AUTOSAR TP Beschreibung in FIBEX

In zukünftigen FIBEX-Versionen ist mit der Integration weiterer AUTOSAR-kompatibler Beschreibungselemente zu rechnen. Um die Aufwärtskompatibilität neuer Systeme, die mit AUTOSAR-Methoden und Werkzeugen (siehe Kap. 8) entworfen werden, zu vorhandenen Werkzeugen für die Entwicklung und Analyse der Buskommunikation zu gewährleisten, stellen die Werkzeughersteller Konvertierungsprogramme bereit, mit denen die AUTOSAR XML-Beschreibungsformate in etablierte Datenformate wie FIBEX, CANdb oder LDF umgewandelt werden können. Da in diesen Formate aber einige semantische Elemente fehlen, die in AUTOSAR enthalten sind, müssen die älteren Formate erweitert, damit keine Definitionen verloren gehen und die Umwandlung gegebenenfalls auch wieder in umgekehrter Richtung möglich ist.

6.4 Überblick über ASAM AE MCD 2 und MCD 3

ASAM AE MCD 3 definiert die Schnittstelle zu übergeordneten Mess-, Kalibrier- und Diagnosewerkzeugen, die diese sowohl von den Details der darunterliegenden Kommunikationsprotokolle und Bussysteme als auch von der kompletten Datenhaltung für die zugehörigen Beschreibungsdaten entkoppelt (vgl. Abb. 6.2). Das Format der Beschreibungsdateien ist in den ASAM MCD 2 Spezifikationen vorgegeben.

Die Grundidee besteht darin, Diagnosetester und Mess- und Kalibrierwerkzeuge nicht für jedes Fahrzeug und Steuergerät neu zu programmieren, sondern ein universelles Programm, den sogenannten MCD 3 Server (Abb. 6.22) einzusetzen, der sich selbstständig über MCD 2-Beschreibungsdateien auf das jeweilige Fahrzeug und seine Steuergeräte parametriert und konfiguriert (*datengetriebener Tester*).

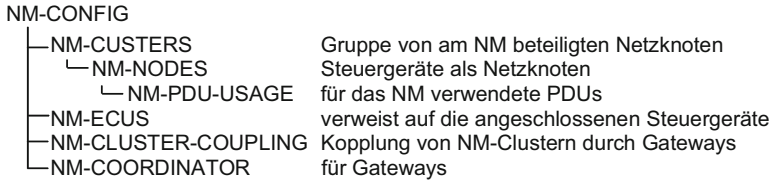

Abb. 6.21 Struktur der AUTOSAR NM Beschreibung in FIBEX

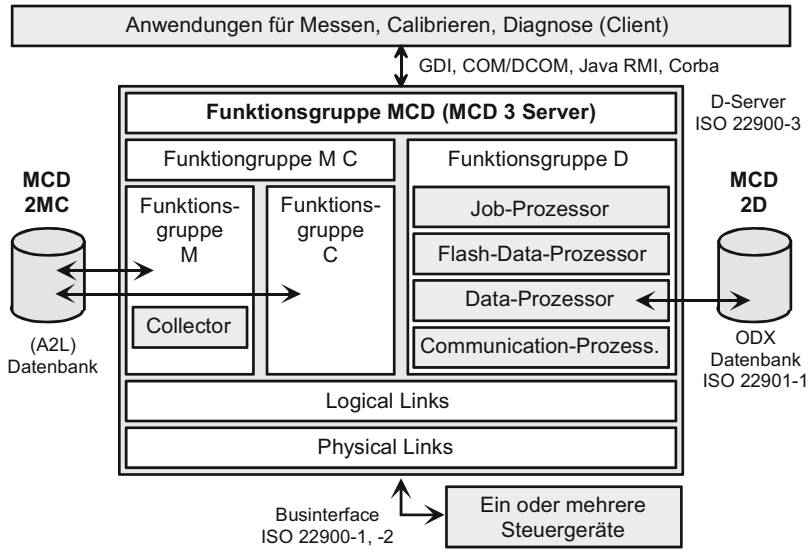

Abb. 6.22 Aufbau eines ASAM-Systems

Der für den Benutzer sichtbare Teil des Systems, bei ASAM als Client bezeichnet, enthält die Mess-, Kalibrier- und/oder Diagnoseanwendung mit der Bedienoberfläche, wobei mehrere solcher Clients gleichzeitig aktiv sein können. Der Client selbst ist in ASAM MCD 3 ausdrücklich nicht spezifiziert, um auf den individuellen Einsatzfall zugeschnittene Lösungen zu erlauben und den Wettbewerb zwischen den Werkzeugherstellern nicht unnötig einzuschränken. Die Schnittstelle zum Server dagegen und die Struktur des Servers, des ASAM MCD 3 – Laufzeitsystems, wird detailliert beschrieben. Aber auch wenn die geforderte Funktionalität an der Schnittstelle durch MCD 3 exakt vorgegeben ist, darf die Kommunikation zwischen Client und Server von den Herstellern mit verschiedenen Programmiersprachen und Standard-Technologien implementiert werden. Die MCD 3 Spezifikation erwähnt Java und Java Remote Method Invocation (RMI), das meist in Verbindung mit C/C++ und neuerdings C# eingesetzte Microsoft COM/DCOM (Active X), das aus der UNIX-Welt stammende Corba und das ASAM-eigene GDI als Implementierungsvarianten. Trotz formaler MCD 3 Kompatibilität werden Client und Server verschiedener Hersteller daher nur dann zusammenarbeiten, wenn sie auch für die Implementierung der Schnittstelle dieselbe Technologie verwenden.

Wie in der Einführung zu diesem Kapitel bereits dargestellt, existieren im Bereich Messen-Kalibrieren einerseits und Diagnose andererseits eine große Anzahl von unterschiedlichen, zueinander inkompatiblen Standards und Datenformaten, die seit Langem etabliert sind und vermutlich noch lange koexistieren werden. Diese Unterschiede versucht man durch die in Version 2 neu konzipierte, einheitlich objektorientierte MCD 3 Schnittstelle zu verdecken. Bei den bis Anfang der 90er Jahre zurückreichenden, aber in realen

Produkten nur selten eingesetzten Vorgängerversionen waren die MC- und die D-Welt noch streng getrennt und die ältere MC-Schnittstelle war prozedural definiert. Völlig durchgängig ist die Integration auch weiterhin nicht, die Funktionsgruppen M, C und D werden immer noch unterschieden (Abb. 6.22) und die Datenformate für die Datenablage von MC-Informationen einerseits und D-Informationen andererseits sind historisch bedingt unterschiedlich. Im MC-Bereich wird das im folgenden Abschn. 6.5 beschriebene ASAP2-Format (A2 L, AML) verwendet, im D-Bereich ODX (Abschn. 6.6).

Die objektorientierte Schnittstelle verwendet für beide Teilbereiche dieselben Konzepte und versucht, die Details der Datenformate zu kaschieren, so dass für den Anwender nur dort Unterschiede sichtbar werden, wo dies durch die unterschiedliche Aufgabenstellung bedingt ist. Insgesamt ist ASAM MCD 3 allerdings sehr komplex. Die Spezifikation allein umfasst über 1600 Seiten, die Beschreibung der ASAP2- und ODX-Datenformate benötigt weitere 700 Seiten. ASAP MCD 2 und MCD 3-kompatible Systeme kommen daher nur schrittweise auf den Markt und implementieren in der Regel zunächst nur Teilmengen der gesamten Spezifikation. Die Spezifikation lässt ausdrücklich zu, dass nur einer der Funktionsblöcke M, C oder D oder eine Kombination davon implementiert wird, wobei die Implementierung eines Funktionsblocks aber in jedem Fall vollständig sein muss. Ein Beispiel dafür wird in Abschn. 9.7 vorgestellt. Angesichts der Komplexität ist auf den folgenden Seiten nur ein grober, stark vereinfachter Überblick möglich. Dabei sollen zunächst die beiden Datenformate für den Bereich MC und für den Bereich D beschrieben werden, bevor anschließend auf Interna der MCD 3-Schicht eingegangen wird.

6.5 Applikationsdatensätze nach ASAM MCD 2 MC

6.5.1 ASAP2/A2 L-Applikationsdatensätze

Mess- und Kalibrieraufgaben fallen hauptsächlich während der Entwicklung von Geräten, Aggregaten und des Gesamtfahrzeuges an. Hierbei ist der möglichst flexible Zugriff auf alle steuergeräteinternen Daten und leichte Verstellbarkeit wichtig.

Die Beschreibung, welche internen Größen des Steuergerätes messbar und welche Parameter veränderbar sind (vgl. Abb. 6.2), erfolgt mit derselben Syntax, die bereits für die Beschreibung der XCP- bzw. CCP-Schnittstelle zwischen Applikationssystem und Steuergerät verwendet wird und im Abschn. 6.2.3 erörtert wurde. Die Grundstruktur der Beschreibung vollständiger Applikationsdatensätze, häufig nach dem früheren Namen von ASAM als ASAP2-Format bezeichnet, ist in Abb. 6.23 dargestellt.

Ein Projekt kann ein einzelnes Steuergerät oder ein vollständiges Fahrzeug umfassen (Tab. 6.9). Da ein Fahrzeugprojekt in der Regel mehrere Geräte enthalten wird, kann die Beschreibung der einzelnen Steuergeräte (MODULE) in separaten Dateien erfolgen, die mit Hilfe von/include-Anweisungen in die Projektdatei eingebunden werden. In derselben Weise wird meist auch die schon aus Abschn. 6.2.3 bekannte Beschreibung der Kommu-

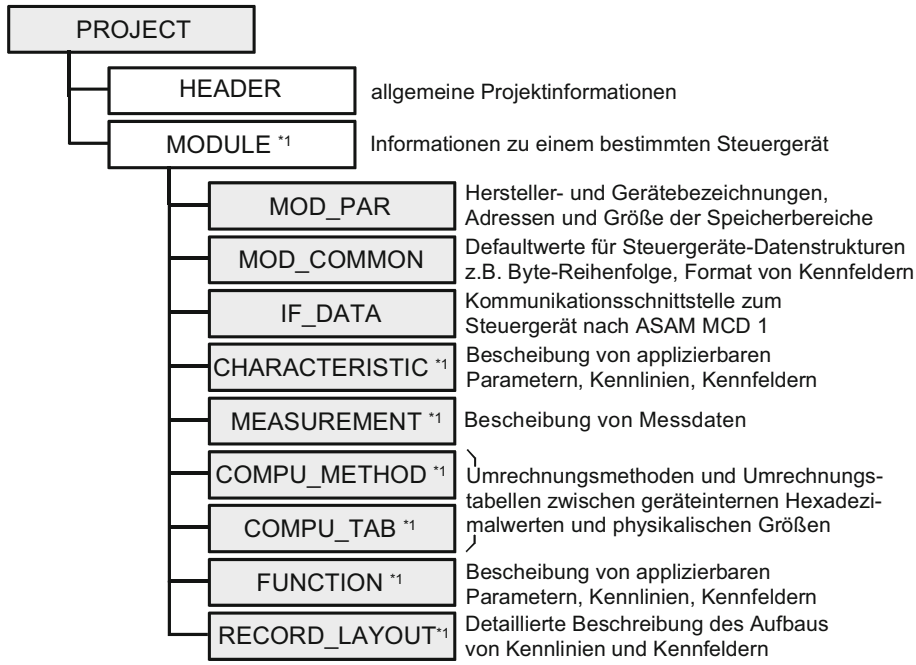

Abb. 6.23 Struktur einer ASAM MCD 2MC-Beschreibung; *1 Abschnitte dürfen mehrfach vorkommen

nikationsschnittstelle IF_DATA zum Steuergerät nach ASAM MCD 1 eingebunden (vgl. Tab. 6.2).

Der Abschnitt MOD_PAR enthält allgemeine Informationen über das Steuergerät wie den Hersteller (SUPPLIER, CUSTOMER, USER), Gerätetyp (ECU) und Versionsnummer (VERSION), Typ der Steuergeräte-CPU (CPU_TYPE), Anzahl der Kommunikationsschnittstellen, Anfangsadressen und Größen der Speicherbereiche des Steuergeräteprogramms und der Applikationsdaten (MEMORY_LAYOUT), globale Konstanten (SYSTEM_CONSTANT) sowie herstellerabhängige Parameter für die Applikationsschnittstelle (CALIBRATION_METHOD).

Im folgenden Abschnitt MOD_COMMON finden sich Hinweise zu steuergeräteinternen Datenstrukturen wie der Byte-Reihenfolge (BYTE_ORDER), der Standard-Datenwortbreite und der Datenausrichtung (DATA_SIZE, ALIGNMENT_…) und dem Standardaufbau von Tabellen für Kennlinien und Kennfelder (DEPOSIT, S_REC_LAYOUT).

Für jede applizierbare Datenstruktur ist ein CHARACTERISTIC-Abschnitt vorhanden, der einen Klartext-Bezeichner für die Datenstruktur, die Adresse sowie den Aufbau der Struktur angibt. Dabei kann es sich um Einzelwerte, Textstrings, Arrays, Kennlinien oder Kennfelder handeln. Für Details zum Aufbau von Kennlinien und Kennfeldern wird auf den entsprechenden RECORD_LAYOUT-Eintrag verwiesen. Zusätzlich können in einge-

Tab. 6.9 Auszug aus einer ASAP2-Konfigurationsdatei

```
/begin PROJECT XCP
   . . .
   /include "ABS_ECU.a21"  /* Einbinden eines weiteren Steuergerätes*/
   . . .
   /begin MODULE           /* Beschreibung eines Steuergerätes */
      /begin MOD_PAR
         SUPPLIER "Muster AG"
         CUSTOMER "ToyCarProductions"
         ECU „Electronic Diesel Engine Management EDC 24"
         CPU_TYPE "FreeIntelfineon CoreMpcTri"
         . . .
      /end MOD_PAR
      . . .
      /begin MOD_COMMON  Allgemeines_Datenformat
         . . .
         BYTE_ORDER BIG_ENDIAN
         DATA_SIZE 8          /* Standard-Datengröße 8 bit */
         ALIGNMENT_BYTE       /* Datenausrichtung an Byte-Grenzen */
         . . .
      /end MOD_COMMON
      . . .
      /begin CHARACTERISTIC Maximale_Einspritzmenge
         . . .
         VALUE                /* Typ Konstante */
         0x7140               /* Adresse im Steuergerätespeicher */
         . . .
         FUEL_QUANTITY        /* Verweis auf Umwandlungsvorschrift */
         0.0    80.0          /* Minimalwert, Maximalwert  */
         . . .
      /end CHARACTERISTIC
      . . .
      /begin MEASUREMENT Motordrehzahl
         . . .
         UWORD                /* Datentyp 16bit unsigned */
         N_RULE               /* Verweis auf Umwandlungsvorschrift */
         4                    /* Auflösung in bit */
         1.0                  /* Genauigkeit in % */
         0.0  6000.0          /* Minimalwert, Maximalwert */
         . . .
         /begin FUNCTION_LIST N_CTRL INJECTION
         /end FUNCTION_LIST /* Referenz auf Funktionsgruppen */
         . . .
      /end MEASUREMENT
      . . .
      /begin COMPU_METHOD N_RULE /* Umwandlung Hex-Wert -> phys.Wert*/
         . . .
         RAT_FUNC             /* Gebrochen rationale Funktion
                                 y = (a x² + b x + c) / (d x² + e x + f)   y…hex., x… phys*/
         "%4.0"               /* Formatierung für Bildschirmdarstellung*/
         "1/min               /* Dimension */
         COEFFS 0.0  255.0 0.0   /*Koeffizienten a,b,…,f der Funktion*/
                0.0 5800.0 0.0
      /end COMPU_METHOD
      . . .
      /begin FUNCTION N_CTRL "speed control"
      /end FUNCTION           /* Funktionsgruppe Drehzahlregelung
      /begin FUNCTION INJECTION "injection control"
      /end FUNCTION           /* Funktionsgruppe Einspritzsteuerung */
      . . .
   /end MODULE
/end PROJECT
```

schränkter Form auch Abhängigkeiten von anderen Datenstrukturen und Grenzwerte für den Verstellbereich angegeben werden.

Für jeden Messwert, d. h. alle internen Größen des Steuergerätes, die im Applikationswerkzeug angezeigt und aufgezeichnet werden können, gibt es einen MEASUREMENT-Abschnitt. Neben dem Datentyp und der Speicheradresse werden Unter- und Obergrenzen, Auflösung und Genauigkeit des Messwertes angegeben und auf eine Umrechnungsformel in einem COMPU_METHOD-Abschnitt verwiesen, mit der der steuergeräteinterne Hexadezimalwert in einen physikalischen Messwert umgerechnet werden kann. Umrechnungsmethoden können Formeln, Umrechnungstabellen oder, z. B. bei Bitwerten, eine Klartextbedeutung sein. Bei Umrechnungstabellen wird dabei auf einen entsprechenden COMPU_TAB-Eintrag verwiesen.

Über FUNCTION-Abschnitte können die applizierbaren Parameter bzw. Messdaten zu Funktionsgruppen zusammengefasst werden, indem z. B. eine Funktionsgruppe Leerlaufregler definiert und auf alle Parameter und Messdaten verwiesen wird, die für diese Funktionsgruppe relevant sind. Da Motorsteuerungen heute oft mehrere Tausend applizierbare Datenstrukturen und Messwerte besitzen, ist eine derartige Gruppierung für eine sinnvolle Benutzerführung im Applikationswerkzeug unabdingbar.

Bei praktisch jedem Abschnitt der Beschreibung können in einem Unterabschnitt IF_DATA herstellerspezifische Parameter für die ASAM 1-Schicht, d. h. für den eigentlichen Zugriff auf das Steuergerät definiert werden. Das Format der Parameter, z. B. Strings, Hexadezimalzahlen usw., hängt von der Aufgabenstellung ab. Aus Sicht von ASAM 2 handelt es sich dabei um Wertefolgen, die Binary Large Objects BLOB, die ohne weitere Interpretation an die darunterliegende Schicht 1 weitergegeben werden.

6.5.2 Calibration Data Format CDF und Meta Data Exchange MDX

Das XML-basierte, ebenfalls vom ASAM-Konsortium spezifizierte CDF ergänzt die ASAP2/A2 L-Datensätze um übergeordnete Informationen und könnte es in Verbindung mit MDX langfristig vielleicht ablösen.

Eine CDF-Beschreibung (Abb. 6.24) enthält für jedes Steuergerät ein SW-SYSTEM-Element, unterhalb dessen die einzelnen Applikationsdaten als SW-INSTANCE-Knoten abgebildet werden. Dabei können alle im etablierten A2 L-Format vorhandenen Datentypen wie Skalare (VALUE), Strings (ASCII), Kennlinien (CURVE) oder Kennfelder (MAP) abgebildet werden. Neben dem eigentlichen Wert (SW-VALUES-PHYS) kann die physikalische Einheit (UNIT-DISPLAY-NAME) angegeben werden.

Im Vergleich zum A2 L-Format erlaubt es CDF nun, den Qualitätsstand (*Quality Meta Data*) der Applikationsdaten mit Hilfe von SW-CS-HISTORY Elementen zu beschreiben. Darin wird einem Datenelement das Kalenderdatum (DATE) der letzten Änderung, der Name des verantwortlichen Applikationsingenieurs (CSUS) sowie der Qualitätsstand (STATE) und weitere Informationen zugeordnet werden. Unter Qualitätsstand wird angegeben, ob das Datenelement noch gar nicht appliziert ist, gegenüber einem früheren Stand

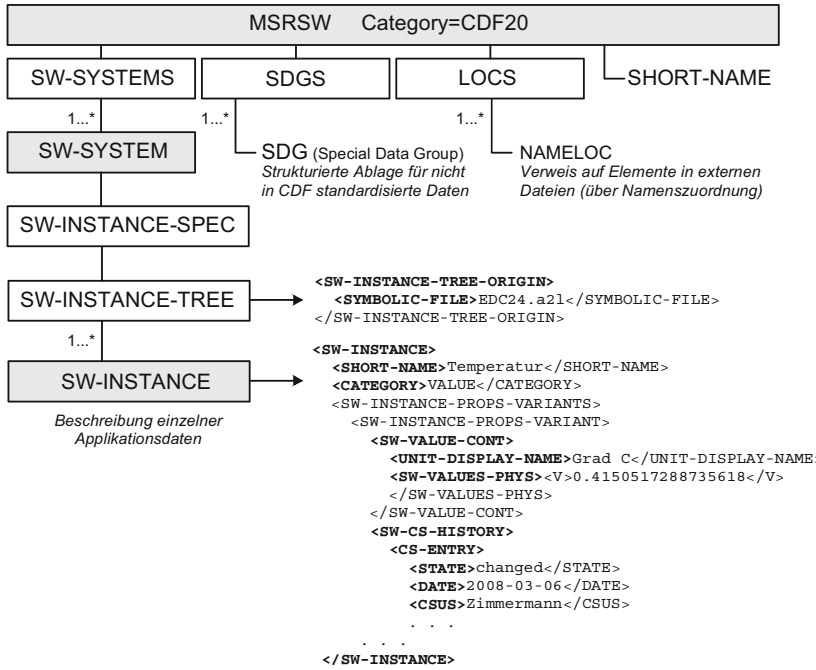

Abb. 6.24 Vereinfachte Grundstruktur einer CDF-Beschreibung

verändert (*changed*) wurde, ob der neue Wert nur vorläufig (*prelimCalibrated*) oder bereits endgültig festgelegt (*calibrated*) wurde und ob er bereits gegengeprüft (*checked*) bzw. für die Serie freigegeben (*completed*) wurde. Dieses Konzept wurde aus dem *Parameter Content (PaCo)* Standardentwurf des MSR-Konsortiums übernommen, das sich mittlerweile an ASAM angeschlossen hat.

CDF speichert grundsätzlich die physikalischen Werte der Applikationsdaten. Umrechnungsformeln zwischen den physikalischen und steuergeräteinternen Werten, wie die COMPU-METHOD in A2 L, oder die Speicheradresse, unter der die Daten im Steuergerät gespeichert sind, werden nicht abgebildet. In der derzeitigen Form kann CDF die bekannten A2 L-Datensätze für die Steuergeräte-Applikation deshalb nicht ersetzen, sondern nur ergänzen. Daher enthält die CDF-Datei unter SYMBOLIC-FILE in der Regel einen Verweis auf die zugehörige A2 L-Datei. Die Zuordnung setzt voraus, dass die Datenwerte in der A2 L- und in der CDF-Datei denselben Kurznamen (SHORT-NAME) verwenden oder dass mit Hilfe von NAMELOC Definitionen ein Namensmapping vorgenommen wird.

Die bei CDF fehlenden Informationen über die steuergeräteinternen Werte könnten statt aus der A2 L-Datei auch aus einer MDX-Datei bezogen werden. Das *Meta Data Exchange Format for Software Module Sharing* soll die Steuergerätesoftware so beschreiben, dass Softwaremodule zwischen verschiedenen Projekten oder zwischen Softwarezulieferer und Geräte- bzw. Fahrzeughersteller leichter ausgetauscht werden können. Das Format

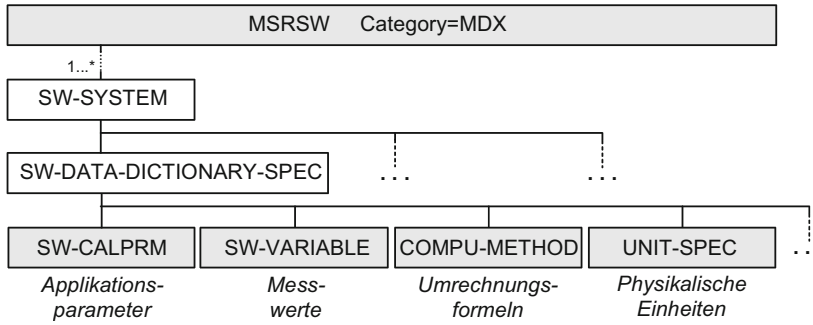

Abb. 6.25 Wichtige Elemente einer MDX-Beschreibung für Applikationsdatensätze

wurde im selben MSR-Konsortium definiert wie CDF und ähnelt diesem in der Struktur sehr stark (Abb. 6.25). Ob sich MDX für die Beschreibung von Softwaremodulen auf dem Markt durchsetzt, wird stark davon abhängen, ob und wie es in die Beschreibungswelt von AUTOSAR integriert wird (siehe Kap. 8).

6.6 ODX-Diagnosedatensätze nach ASAM AE MCD 2D

Open Diagnostic Data Exchange (ODX) spezifiziert ein Datenmodell für die Haltung bzw. den Austausch alle diagnoserelevanten Daten eines Steuergerätes oder eines Fahrzeugs [1]. Unter Diagnose werden Aufgaben verstanden, die bei der Wartung und Fehlersuche in Werkstätten aber auch in der Fertigung von Fahrzeugen anfallen:

- Auslesen und Löschen des Fehlerspeichers,
- Durchführung von Funktionstests, Abfrage von Mess- und Stellgrößen des Fahrzeugs,
- Auslesen von Identifikationsdaten,
- Variantendefinition, Variantenauswahl und Variantenkonfiguration,
- Programmierung neuer Steuergerätesoftware und Datensätze (*Flashen*).

Die Umsetzung erfolgt mit den in Kap. 5 beschriebenen Diagnoseprotokollen UDS oder KWP 2000. Diese Protokolle definieren jedoch nur das Botschaftsformat der Diagnosedienste. Die innerhalb der Botschaften übertragenen Parameter unterscheiden sich je nach Fahrzeughersteller (OEM), Steuergerätehersteller und Gerätetyp. Insbesondere in der Art und Weise, wie, wann und in welchem Umfang Fehler in den Fehlerspeicher eingetragen werden, gibt es sehr unterschiedliche Herstellerphilosophien, die durch gesetzliche Anforderungen wie OBD nur teilweise harmonisiert werden.

Ein nicht minder wichtiges Themengebiet stellt die sogenannte Variantenverwaltung und -identifikation dar. Dahinter verbirgt sich die Möglichkeit, dass Steuergeräte des gleichen Typs unterschiedliche Ausprägungen ihrer Funktionalität besitzen können. So wird

zum Beispiel in einer Variante eines Tachometer-Steuergerätes für Europa die Geschwindigkeit im km/h angezeigt, in der Variante für die USA hingegen in mph, d. h. die Rohdaten der Geschwindigkeitsmessung werden je nach Variante anders interpretiert. Ein zweiter Ansatz zur Verwendung von Varianten stellt die Dokumentation von Entwicklungs- und Serienständen eines Steuergerätes dar. Die jeweilige Variante des Steuergerätes lässt sich anhand der Identifikationsdaten erkennen. Zur Vermeidung von redundanter Datenhaltung (*Single-Source*-Prinzip) und zur Vereinfachung der Datenpflege besteht an das Austauschformat die Forderung, alle Varianten ausgehend von einer Basis-Variante innerhalb eines einzigen Datensatzes zu halten.

Der oben beschriebene Diagnoseumfang eines datengetriebenen Diagnosetesters muss durch ein Datenmodell vollständig abgebildet werden. Die Datensätze müssen innerhalb von Entwicklung, Produktion und Service zwischen OEM, Zulieferer, Tool-Hersteller und Testeinrichtung, z. B. in einer Werkstatt, ausgetauscht werden können, ohne dass weitere Informationen notwendig sind.

ASAM definierte zunächst ein SGML-basiertes Datenformat, das sich für die Praxis als zu komplex erwies. Infolgedessen wurde eine vereinfachte Version 1.1.4 unter dem Namen *ASAM MCD 2D Basic* herausgeben, für die sich die Bezeichnung ODX (*Open Diagnostic Data Exchange*) einbürgerte. Das Datenformat basiert auf XML (siehe Kasten am Ende von Abschn. 6.2) und wurde zu dem nachfolgend beschriebenen *ASAM-MCD 2D (ODX)* weiterentwickelt und in der Version ODX V2.1 als ISO 22901-1 standardisiert. Es basiert auf einem objektorientierten Ansatz und verwendet zur visuellen Darstellung in der Dokumentation die in der Informatik mittlerweile üblichen UML-Diagramme (*Unified Modeling Language*).

6.6.1 Aufbau des ODX-Datenmodells

Das ODX-Datenmodell ist unterteilt in acht große Kategorien (ODX-CATEGORY) (Abb. 6.26) für unterschiedliche Aufgabenbereiche:

Beschreibung der Kommunikationsprotokolle und des Fahrzeugnetzes

- **VEHICLE-INFO-SPEC**
  Definiert alle Informationen für die Fahrzeugidentifikation und den Fahrzeugzugang. Dabei wird auch die Netzwerk-Topologie beschrieben, welche für den Zugriff auf die Steuergeräte über die Bussysteme und Gateways relevant ist.
- **COMPARAM-SPEC**
  Fasst die Kommunikationsparameter für einen Protokoll-Stapel (PROT-STACK) bestehend aus mehreren COMPARAM-SUBSETs zusammen.
- **COMPARAM-SUBSET** (ab ODX 2.1)
  Beschreibt die Kommunikationsparameter für eine bestimmte Schicht im ISO/OSI- Referenzmodel, z. B. das Zeitverhalten für die physikalische Schicht oder das Transport-Protokoll.

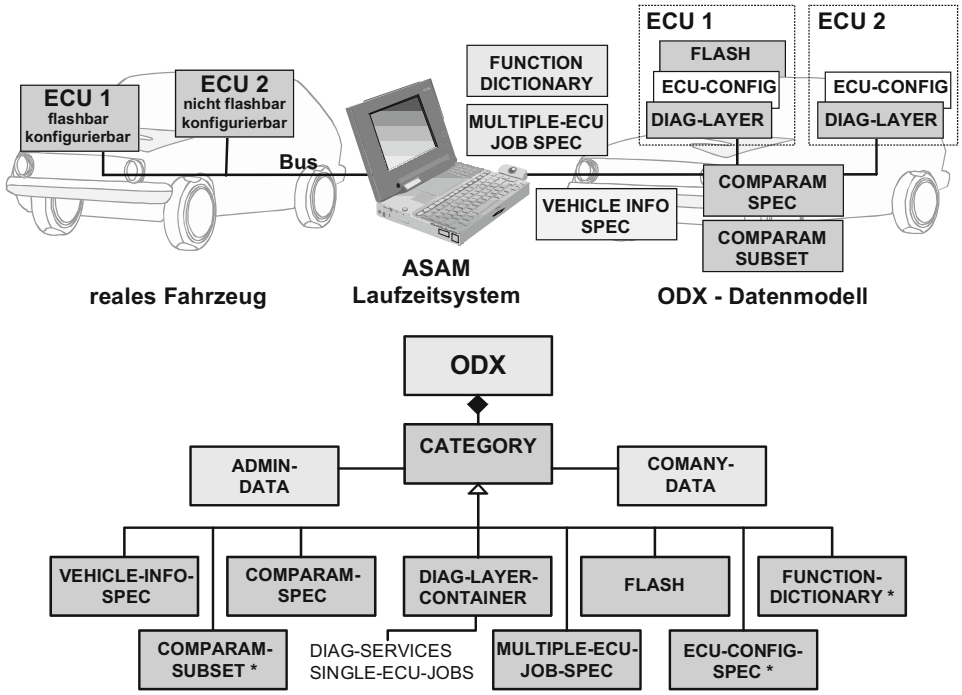

Abb. 6.26 Komponenten des ODX-Datenmodells (* ab ODX V2.1)

Hierarchische Beschreibung der Diagnosestruktur, Diagnosedienste und Abläufe

- **DIAG-LAYER-CONTAINER**
  Beschreibt mit den Objekten BASE-VARIANT, ECU-VARIANT, PROTOCOL, FUNC TIONAL-GROUP und ECU-SHARED-DATA den hierarchisch aufbaugebauten Datensatz eines Steuergeräte-Typs und dessen Varianten aus Diagnosesicht. Die PROTOCOL-Schicht verweist auf Diagnosedienste (DIAG-SERVICES) und Diagnoseabläufe (SINGLE-ECU-JOB), zusammen als DIAG-COMM bezeichnet, mit den notwendigen Daten und Parametern, sogenannten Datenobjekten DOP.
- **MULTIPLE-ECU-JOB-SPEC**
  Beschreibt sogenannte *Diagnostic Jobs (Abläufe oder Makros)*, welche sich auf mehrere Steuergeräte beziehen und daher nicht innerhalb der eigentlichen Steuergeräte-Beschreibungen (DIAG-LAYER) dargestellt werden können.

Spezielle Aufgabenbereiche

- **FLASH**
  Hier werden in ECU-MEM-Objekten alle Informationen abgelegt, die für das Programmieren von Code oder Daten in ein Steuergerät notwendig sind.

- **ECU-CONFIG-SPEC** (ab ODX 2.1)
  Beinhaltet die Informationen zur Steuergerätekonfiguration (mitunter als *Variantencodierung* bezeichnet).
- **FUNCTION-DICTIONARY** (ab ODX 2.1)
  Beinhaltet Informationen zur funktionsorientierten Diagnose.

Wird für eine dieser ODX-Kategorien ein Datensatz (Instanz) angelegt, so spricht man von einem *ODX-Dokument*. Ein *ODX-Dokument* beinhaltet immer nur eine der oben aufgeführten ODX-Kategorien. Die ODX-Kategorisierung erfolgte unter dem Aspekt, dass die Dateninhalte für die vollständige Beschreibung der Diagnoseumfänge eines Fahrzeuges aus unterschiedlichen Quellen stammen und unterschiedliche Lebenszeiten besitzen. So kann der Umfang der notwendigen Diagnosedienste (DIAG-LAYER-CONTAINER) für ein Steuergerät am besten von dessen Hersteller definiert werden. Dieser hat aber keine Informationen darüber, wie das Steuergerät in das Netzwerk des Fahrzeuges integriert wird und der externe Zugriff darauf erfolgt (VEHICLE-INFO-SPEC). Bestimmte Kommunikationsparameter einer entsprechenden Schicht im ISO/OSI-Referenzmodel (COMPARAM-SUBSET), z. B. die Baudrate der physikalischen Schicht „CAN“, sind für alle Steuergeräte an einem bestimmten Bus gleich. Dasselbe gilt für ganze Protokollstapel bestehend aus physikalischer, Transport- und Diagnoseschicht (COMPARAM-SPEC).

Um eine redundante Datenhaltung zu vermeiden, besteht innerhalb von ODX-Dokumenten die Möglichkeit, auf Datensätze anderer ODX-Dokumente zu referenzieren und so indirekt Zugriff auf diese Datensätze zu bekommen.

Viele ODX-Dateninhalte werden während der Projektlebenszeit durch unterschiedliche Personen und Firmen Änderungen unterzogen, die gegebenenfalls nachvollzogen werden müssen. Zu vielen ODX-Elementen lassen sich daher zusätzlich ADMIN-DATA (Versionierung, Datum und Änderungstext) und COMPANY-DATA Informationen ablegen, mit denen die Änderungshistorie beschrieben werden kann.

Im Folgenden wird zunächst die Hierarchiestruktur der DIAG-LAYER beschrieben. Danach werden die ODX-Beschreibungselemente für die Kommunikationsverbindung und die Diagnosedienste dargestellt. Abschließend folgen die Beschreibungsmechanismen für Sonderaufgaben wie die Flash-Programmierung.

6.6.2 DIAG-LAYER: Hierarchische Diagnosebeschreibung

Im einfachsten Fall besteht die ODX-Beschreibung aus drei Teilen (Abb. 6.27), der Beschreibung der Fahrzeugtopologie, dem Parametersatz des Bussystemes und als wichtigster Teil für jedes im Fahrzeug eingebaute Steuergerät genau einem DIAG-LAYER-CONTAINER. Die Beschreibung der Fahrzeugtopologie (VEHICLE-INFO-SPEC) enthält Informationen über den Hersteller und Typ des Fahrzeugs und beschreibt, welche Steuergeräte im Fahrzeug verbaut sind, über welche Gateways und Bussysteme sie für den Diagnosetester erreichbar sind und verweist auf den Datensatz des jeweiligen Steuergeräts

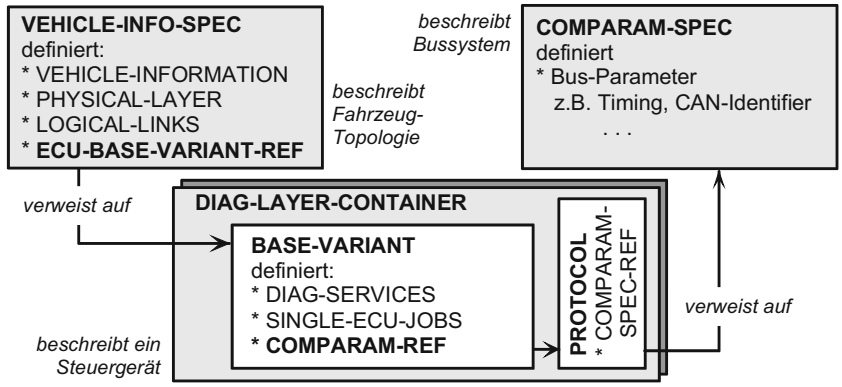

Abb. 6.27 Minimal-Struktur einer ODX-Beschreibung

im DIAG-LAYER-CONTAINER. Dieser enthält im einfachsten Fall im Unterabschnitt BASE-VARIANT sämtliche Diagnosedienste und Diagnoseabläufe eines Steuergerätes und verweist über das PROTOCOL auf die COMPARAM-SPEC, in der die Timing-Parameter, CAN-Identifier usw. des Bussystems definiert sind, über das kommuniziert wird.

In der Praxis existieren für jedes Steuergerät verschiedene Varianten. In der einfachen Struktur nach Abb. 6.27 müsste man jede Gerätevariante in einem eigenen BASE-VARIANT-Datensatz abbilden, der alle Diagnosedienste vollständig beschreibt, die das Gerät unterstützt. Um die Komplexität der Datensätze zu reduzieren und die Konsistenz der Datensätze leichter sicherzustellen, ist es besser, die gemeinsamen Eigenschaften aller Gerätevarianten nur einmal zu beschreiben, und für jede Gerätevariante nur die Unterschiede darzustellen. Dies wird durch das Konzept der Diagnoseschichten (*Diagnostic Layer*) in ODX ermöglicht (Abb. 6.28).

Die Steuergeräte-Beschreibung kann eine gemeinsame Bibliothek (ECU-SHARED-DATA) sowie bis zu vier Schichten, die DIAG-LAYER, enthalten, wobei lediglich die BASE-VARIANT-Schicht verbindlich vorgeschrieben ist. Die Verwendung des Schichtenmodells soll an einem Beispiel erläutert werden. Betrachtet werden die Steuergeräte für die Türen eines Fahrzeugs, die an das Karosserie-Bussystem angeschlossen sind. Das Türsteuergerät wird im Fahrzeug zweimal eingebaut, wobei die Variante für die Front-Türen zusätzlich die Steuerung der Außenspiegel übernimmt.

Diese Funktionen sind im Steuergerät für die Heck-Türen aus Kostengründen nicht enthalten. Die zugehörige ODX-Beschreibung könnte folgenden Aufbau haben (Abb. 6.28 und Tab. 6.10):

- In der obersten Schicht, dem PROTOCOL-LAYER (1) werden die für das verwendete Diagnoseprotokoll relevanten Daten und Dienste spezifiziert werden. Dort könnten z. B. die UDS Diagnosedienste wie *Read Data by Identifier* (vgl. Abschn. 5.2) definiert und auf die COMPARAM-SPEC des Bussystems verwiesen werden.

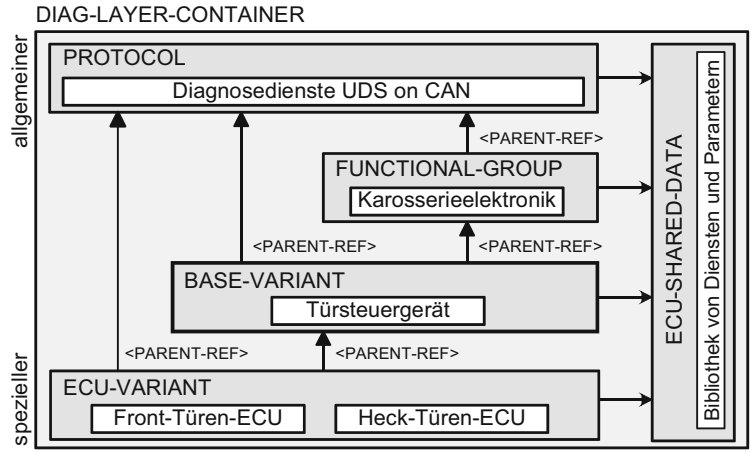

Abb. 6.28 Hierarchische Beschreibung (*Diagnostic Layer*) für Steuergeräte

- Die zugehörigen Parameter der Dienste, z. B. die *Identifier*, die bei jeder Steuergerätefamilie andere Werte haben, würde man dagegen sinnvollerweise in der ECU-BASE-VARIANT-Schicht definieren (2).
- Manche dieser *Identifier* allerdings werden nicht nur von den Türsteuergeräten verwendet, sondern sind vom Fahrzeughersteller vielleicht für alle Steuergeräte im Bereich der Karosserieelektronik einheitlich vorgegeben. ODX bietet die Möglichkeit, solche für alle Geräte einer bestimmten Anwendungsklasse gültigen Daten in einer Funktionsgruppe, der FUNCTIONAL-GROUP-Schicht, zusammenzufassen.
- Die Unterschiede zwischen der Front- und der Heck-Variante des Türsteuergerätes werden in der ECU-VARIANT-Schicht beschrieben (3). Dort wird dargestellt, welche zusätzlichen oder geänderten Diagnosedienste bzw. Daten das jeweilige Steuergerät im Vergleich zur Basisvariante besitzt und/oder welche Dienste das Steuergerät nicht unterstützt.

Die Beschreibungshierarchie erlaubt im Informatik-Jargon ein *Vererben* von Daten (*Value inheritance*) von oben nach unten, d. h. ein spezielles Steuergerät (untere Schicht oder *Kind-Schicht*) wird durch seine eigenen spezifischen Daten und durch alle in den höheren Schichten (*Eltern-Schicht*) definierten Daten beschrieben. Falls ein bestimmter Datenwert mehrfach definiert wird, so hat der auf der weiter unten liegende, d. h. speziellere Wert, Vorrang vor dem weiter oben stehenden, d. h. allgemeineren Wert. Auf diese Weise kann eine bestimmte Ausführung eines Steuergerätes einen Defaultwert des standardisierten Diagnoseprotokolls durch einen gerätespezifischen Wert überschreiben. Ausgedrückt wird die *Vererbung* in der zugehörigen XML-Beschreibungsdatei (Tab. 6.10) durch ein PARENT-REF-Element, das auf die zugehörige *Eltern-Schicht* verweist (4). Umgekehrt kann eine untere Schicht auch Daten einer höheren Schicht, die überhaupt nicht benötigt oder un-

Tab. 6.10 Hierarchische ODX-Beschreibung einer Steuergerätefamilie

```
<DIAG-LAYER-CONTAINER>
  <SHORT-NAME>ECU1</SHORT-NAME>
  <PROTOCOLS>                          ← Protokoll-Schicht                    (1)
    <PROTOCOL ID="P.UDS.ID" TYPE="ISO_15765_3_on_ISO_15765_2">
      <DIAG-COMMS>
        <DIAG-SERVICE . . .>           ← Diagnosedienste des UDS-Protokolls
        . . .
      <COMPARAM-SPEC-REF ID-REF="COMPARAM.ISO15765.ID" . . . />   (6)
    </PROTOCOL>
  </PROTOCOLS>
  <BASE-VARIANTS>                      ← Grundvariante der Türsteuergeräte    (2)
    <BASE-VARIANT ID="Door_ECU">
      <DIAG-COMMS>                     ← Spezielle Diagnosedienste und Daten
      . . .                              der Türsteuergeräte
      <PARENT-REFS>                    ← Erben der Dienste der Protokoll-Schicht
        <PARENT-REF xsi:type="PROTOCOL-REF" ID-REF="P.UDS.ID" />   (4)
      </PARENT-REFS>
    </BASE-VARIANT>
  </BASE-VARIANTS>
  <ECU-VARIANTS>                       ← Varianten der Türsteuergeräte
    <ECU-VARIANT ID="Rear_Door_ECU">   ← Variante Heck-Tür-Steuergerät    (3)
      <DIAG-COMMS>                     ← Spezielle Diagnosedienste und Daten
      . . .                              des Heck-Tür-Steuergerätes
      <PARENT-REFS>                    ← Erben der Dienste der Grundvariante
        <PARENT-REF xsi:type="BASE-VARIANT-REF" ID-REF="Door_ECU">
          <NOT-INHERITED-DIAG-COMMS>   ← Vom Heck-Tür-Steuergerät nicht  (5)
            <NOT-INHERITED-DIAG-COMM>    unterstützter Dienst der Grundvariante
              <DIAG-COMM-SNREF SHORT-NAME="SeedAndKey" />
    . . .
  </ECU-VARIANTS>
</DIAG-LAYER-CONTAINER>
```

terstützt werden, mit Hilfe von NOT-INHERITED-Elementen gezielt von der Vererbung ausblenden (5).

6.6.3 VEHICLE-INFO-SPEC: Fahrzeugzugang und Bustopologie

Das VEHICLE-INFO-SPEC-Dokument gibt an, für welches Fahrzeugmodell der ODX-Datensatz gültig ist, welche Steuergeräte verbaut sind, über welche Schnittstelle der Diagnosetester Zugang zum Fahrzeug erhält und wie die interne Topologie der Bussysteme im Fahrzeug beschaffen ist (Abb. 6.29).

Im Beispiel nach Abb. 6.29 bzw. Tab. 6.11, auf die sich die nachfolgend in ( ) angegebenen Ziffern beziehen, erfolgt die *Fahrzeugidentifikation* (1,2) schrittweise ausgehend

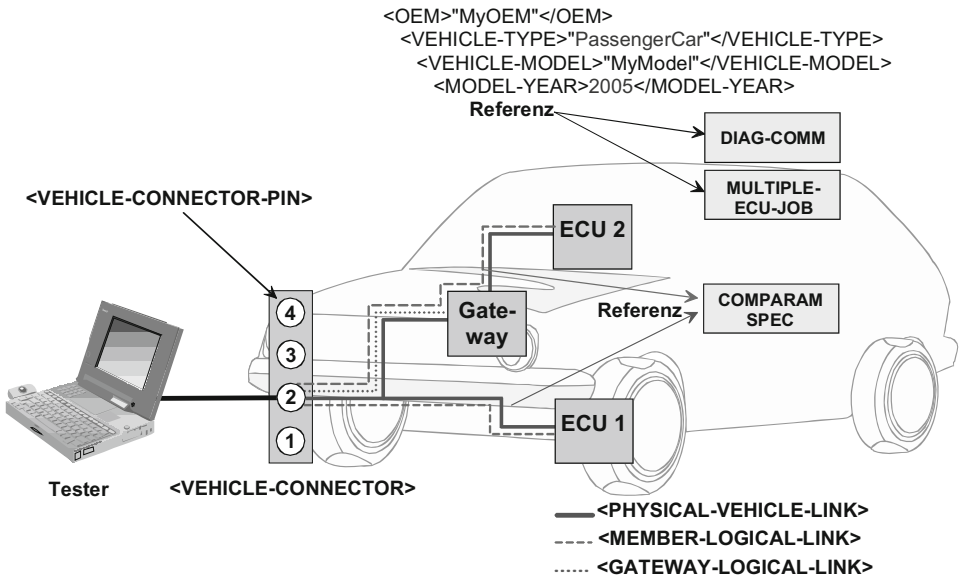

Abb. 6.29 Informationen aus dem ODX-Dokument VEHICLE-INFO-SPEC

vom Hersteller (OEM), über die Fahrzeugklasse (VEHICLE-TYPE), das Fahrzeugmodell (VEHICLE-MODEL) bis zum Modelljahr (MODEL-YEAR). Falls die Selektion im Laufzeitsystem nicht manuell durch den Bediener erfolgt, kann zusätzlich ein Verweis auf das Ergebnis eines Diagnosedienstes (DIAG-COMM oder MULTIPLE-ECU-JOB) angegeben werden, mit dem die Fahrzeugidentifikation gelesen wird.

Für den Fahrzeugzugang, z. B. über den OBD-Stecker, wird zunächst der Stecker selbst als VEHICLE-CONNECTOR (3) beschrieben. Ein VEHICLE-CONNECTOR-PIN definiert durch den PHYSICAL-VEHICLE-LINK (5) und dessen Typ, z. B. CAN, die physikalische Verbindung vom Diagnosetester zum Fahrzeugstecker und von dort zum Steuergerät.

Ein LOGICAL-LINK (4) hingegen repräsentiert einen logischen Pfad vom Diagnosetester zu einem Steuergerät. Falls ein Steuergerät mit mehreren Bussystemen verbunden ist, existieren möglicherweise über die verschiedenen physikalischen Busverbindungen mehrere logische Pfade zu einem Steuergerät. ODX unterscheidet bei einem LOGICAL-LINK zwischen einer Verbindung zu einem Gateway (GATEWAY-LOGICAL-LINK) oder zu einem regulären Steuergerät (MEMBER-LOGICAL-LINK). Jede logische Verbindung kann wiederum auf andere logische Verbindungen referenzieren. Um nun, wie im Abb. 6.29 dargestellt, eine reale Verbindung zum Steuergerät ECU 2 zu bekommen, muss der LOGICAL-LINK „rückwärts“ ausgewertet werden: Zunächst gibt es eine logische Verbindung von ECU 2 zum Gateway, von dort zum Pin 2 des Diagnosesteckers. Das ASAM-Laufzeitsystem kann nun eine Verbindung über die physikalische Schicht, z. B. CAN an Pin 2 des Diagnosesteckers, zum Gateway aufbauen. Das Gateway leitet die Information dann zum Steuergerät weiter.

Tab. 6.11 ODX-Beschreibung des Systems nach Abb. 6.29

```
<VEHICLE-INFO-SPEC ID="COMPACT-CLASS">
  <SHORT-NAME>CompactCar</SHORT-NAME>
  ...
  <INFO-COMPONENTS>                                                    (1)
    <INFO-COMPONENT ID="ID_OEM">                                       (7)
      <SHORT-NAME>OEM</SHORT-NAME>                                     (8)
      <EXPECTED-VALUE>MyOEM</EXPECTED-VALUE>
    </INFO-COMPONENT>

    <INFO-COMPONENT ID="ID_VEHICLE-TYPE">
      <SHORT-NAME>VEHICLE-TYPE</SHORT-NAME>
      <EXPECTED-VALUE>PassengerCar</EXPECTED-VALUE>
    </INFO-COMPONENT>
        ...
  </INFO-COMPONENTS>

  <VEHICLE-INFORMATION>
    <SHORT-NAME>VEHICLE-INFO</SHORT-NAME>
    ...
    <INFO-COMPONENT-REFS>                                              (2)
      <INFO-COMPONENT-REF ID-REF="ID_OEM"/>                            (9)
      <INFO-COMPONENT-REF ID-REF="ID_VEHICLE-TYPE"/>
      ...
    </INFO-COMPONENT-REFS>
    ...
    <VEHICLE-CONNECTOR>                                                (3)
      <SHORT-NAME>OBD-Connector</SHORT-NAME>
      <VEHICLE-CONNECTOR-PINS>
        <VEHICLE-CONNECTOR-PIN TYPE="HI" ID="ID_CAN-HI">
          <SHORT-NAME>CAN-HI</SHORT-NAME>
          <PIN-NUMBER>2</PIN-NUMBER>
        </VEHICLE-CONNECTOR-PIN>
        ...
      </VEHICLE-CONNECTOR-PINS>
    </VEHICLE-CONNECTOR>
    <LOGICAL-LINKS>                                                    (4)
      <LOGICAL-LINK ID="LL_ECU-1" xsi-type="MEMBER-LOGICAL-LINK">
        <SHORT-NAME>ECU-1</SHORT-NAME>
        <PHYSICAL-VEHICLE-LINK-REF ID-REF="PL-CAN-1"/>
        <BASE-VARIANT-REF ID-REF="ECU-1"/>                             (6)
        ...
      </LOGICAL-LINK>
      <LOGICAL-LINK ID="LL_GATEWAY" xsi-type="MEMBER-LOGICAL-LINK">
        <SHORT-NAME>GATEWAY</SHORT-NAME>
        <PHYSICAL-VEHICLE-LINK-REF ID-REF="PL-CAN-1"/>
        <BASE-VARIANT-REF ID-REF="GATEWAY-ECU"/>
        ...
      </LOGICAL-LINK>
      <LOGICAL-LINK ID="LL_ECU-2" xsi-type="GATEWAY-LOGICAL-LINK">
        <SHORT-NAME>ECU-2</SHORT-NAME>
        <GATEWAY-LOGICAL-LINK-REF ID-REF="GATEWAY"/>
        <PHYSICAL-VEHICLE-LINK-REF ID-REF="PL-CAN-2"/>
        <BASE-VARIANT-REF ID-REF="ECU-2"/>
        ...
      </LOGICAL-LINK>
    </LOGICAL-LINKS>
```

```
    <PHYSICAL-VEHICLE-LINKS>                                    (5)
      <PHYSICAL-VEHICLE-LINK ID="PL-CAN-1">
        <SHORT-NAME>PHYSICAL-LINK-CAN-1</SHORT-NAME>
        <VEHICLE-CONNECTOR-PIN-REFS>
          <VEHICLE-CONNECTOR-PIN-REF ID="CAN-HI">
          ...
        </VEHICLE-CONNECTOR-PIN-REFS>
      </PHYSICAL-VEHICLE-LINK>
      <PHYSICAL-VEHICLE-LINK ID="PL-CAN-2">
        ...
      </PHYSICAL-VEHICLE-LINK>
    </PHYSICAL-VEHICLE-LINKS>
  </VEHICLE-INFORMATION>
</VEHICLE-INFO-SPEC>
```

Damit das Laufzeitsystem auch die korrekten Parameter für die Verbindungen, z. B. Baudrate und Zeitverhalten, einstellen kann, enthält der LOGICAL-LINK einen Verweis auf das DIAG-LAYER-Dokument (hier als BASE-VARIANT) für das jeweilige Steuergerät (6). Dort wird das für die Verbindung eingesetzte Busprotokoll definiert und auf die COMPARAM-SPEC verwiesen (vgl. (6) in Tab. 6.10).

An diesem Beispiel wird auch das Konzept von ODX deutlich, Daten möglichst nur an einer Stelle zu definieren (*Single Source* Prinzip) und von anderen Stellen aus durch Verweise zugänglich zu machen. Dazu erhalten ODX-Elemente als Attribut einen *Identifier ID*, eine Kennung, die für den jeweiligen Elementtyp im gesamten Datensatz eindeutig sein muss (7). Die ID wird von entsprechenden ODX-Autorenwerkzeugen häufig automatisch vergeben. Daher erhält das Element zusätzlich auch noch eine Klartext-Kurzbezeichnung SHORT-NAME (8). Auf ein derartig gekennzeichnetes Element kann von einer anderen Stelle aus nun entweder mit Hilfe des Identifiers über einen ODX-Link ID-REF (9) oder mit Hilfe von SN-REF über die Kurzbezeichnung verwiesen werden. Zusätzlich zum Identifier ID und zum Kurznamen SHORT-NAME, die bei praktisch allen ODX-Elementen vorgeschrieben sind, kann einem Element auf Wunsch auch ein längerer Name LONG-NAME und eine ausführliche Beschreibung DESCRIPTION zugeordnet werden, um dem Anwender eines ODX-Werkzeugs oder dem Benutzer des Diagnosetesters weitere Informationen anzuzeigen.

6.6.4 COMPARAM-SPEC und COMPARAM-SUBSET: Busprotokoll

Aus der COMPARAM-SPEC entnimmt das Laufzeitsystem Angaben über das Kommunikationsprotokoll. Die entsprechenden Parameter sind dabei nach ISO/OSI-Schichten getrennt in einzelne COMPARAM-SUBSETs gegliedert, welche wiederum zu einem Protokollstapel (PROT-STACK) zusammengefasst werden können (Abb. 6.30). Für jedes eingesetzte Protokoll existiert seit ODX V2.1 ein eigenes COMPARAM-SUBSET mit dessen spezifischen Kommunikationsparametern. So sind Bus- und Adressinfor-

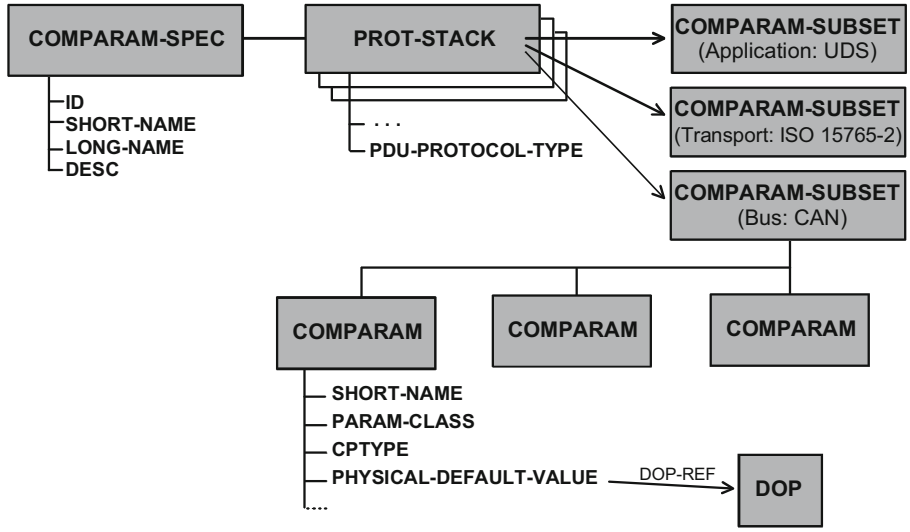

Abb. 6.30 Struktur der Parameterbeschreibung für das Busprotokoll

mationen des Steuergerätes (z. B. Baudrate oder CAN Identifier) und Angaben zum Aufbau einer Kommunikationsverbindung (z. B. Wakeup-Pattern) in einem Subset für die physikalische Schicht abgelegt. Transportprotokoll-Parameter (z. B. ST_{min} bei ISO-15765-2) oder Diagnoseprotokollparameter (z. B. P2 bei UDS) besitzen eigene Subsets. Auch Parameter wie etwa die Anzahl der Sendewiederholungen im Fehlerfall sind hier definiert. Ein Protokollstapel (PROT-STACK) fasst nun die einzelnen Protokollschichten zusammen, indem auf die jeweiligen COMPARAM-SUBSETs referenziert wird. Die COMPARAM-SPEC selbst beinhaltet einen oder mehrere solcher PROT-STACKs. ISO 22901-1 (entspricht ODX V2.1) in Verbindung mit ISO 22900-2 (D-PDU-API) definiert die ODX-Datensätze für die Diagnoseprotokolle KWP 2000 und UDS, das Transportprotokoll ISO 15765-2 und das Bussystem CAN (ISO 11898) sowie die entsprechenden Protokollstack-Kombinationen (PDU-PROTOCOL-TYPE), z. B. *UDS on CAN* als ISO_15765_3_on_ISO_15765_2. Welches Steuergerät welchen Protokollstapel verwendet, wird durch Verweis auf einen PROT-STACK im PROTOCOL-Feld der Diagnostic-Layer-Beschreibung des jeweiligen Steuergerätes festgelegt (siehe (6) in Tab. 6.10).

Jeder einzelne Kommunikationsparameter wird über einen eigenen COMPARAM-Abschnitt in der ODX-Datei beschrieben. Zur Identifikation des Parameters dient sein SHORT-NAME, der Parameter hat einen PHYSICAL-DEFAULT-Wert und verweist auf ein Datenobjekt Data-Object-Property DOP. Über die COMPU-METHOD des DOP kann der physikalische Wert in das interne Format des Bussystems umgerechnet werden. Der allgemeine Aufbau von DOP und COMPU-METHOD wird in Abschn. 6.6.6 beschrieben. Die Art der Kommunikationsparameter wird durch die PARAM-CLASS unterschieden (Tab. 6.12).

Tab. 6.12 Klassen von Kommunikationsparametern

PARAM-CLASS	Beschreibung
TIMING	Zeiten für den Botschaftsverkehr auf dem Bussystem
INIT	Initialisierungsparameter
COM	Allgemeine Kommunikationsparameter (z. B. CAN-Identifier)
ERRHDL	Parameter für die Fehlerbehandlung (z. B. Sende-Wiederholung)
BUSTYPE	Bussystem-spezifische Parameter (z. B. Baudrate)
UNIQUE_ID	Eindeutige Identifier

Tab. 6.13 Weitere Unterteilung von Kommunikationsparametern

CPTYPE	Beschreibung
STANDARD	Der Kommunikationsparameter ist fester Bestandteil eines standardisierten Protokolls und muss von jedem Laufzeitsystem unterstützt werden, welches dieses Protokoll implementiert.
OEM-SPECIFIC	Der Kommunikationsparameter ist fester Bestandteil eines nicht standardisierten, OEM-spezifischen Protokolls. Der Parameter muss vom Laufzeitsystem unterstützt werden, wenn das OEM-spezifische Protokoll implementiert wird.
OPTIONAL	Diese Parameter müssen vom Laufzeitsystem nicht unterstützt werden. Wird ein solcher Parameter trotzdem verwendet, kann das Laufzeitsystem diesen ignorieren und die Kommunikation ohne diesen Parameter weiterführen.

Für das Laufzeitsystem werden alle Kommunikationsparameter nochmals in drei Kategorien unterteilt (Tab. 6.13), mit denen zwischen Standardwerten, herstellerspezifischen und optionalen Werten unterschieden wird.

Im Beispiel nach Tab. 6.14 wird der Parameter P2 min nach ISO 15765 definiert, der dort den Mindestbotschaftsabstand zwischen Testeranfrage und Steuergeräteantwort angibt (vgl. Abschn. 2.2.3). Dieser Parameter (1) besitzt den Initialwert 25 und referenziert ein Datenobjekt DOP namens DOP_P2MIN (2). Durch die Umrechnungsfunktion IDENTICAL des DOPs und die Einheit (UNIT) Millisekunden kann das Laufzeitsystem den Wert für den Kommunikationsparameter P2 min auf 25 ms einstellen.

6.6.5 DIAG-COMM und DIAG-SERVICE: Diagnosedienste

Ein KWP 2000- oder UDS-Diagnosedienst setzt sich aus einer Diagnose-Anforderung vom Tester an das Steuergerät (Request) sowie ein oder mehreren positiven oder negativen Antwortbotschaften vom Steuergerät zum Tester (Response) zusammen (Abb. 6.31, vgl. Kap. 5). Um beispielsweise einen Temperaturwert aus dem Steuergerät auszulesen, könnte der UDS-Diagnosedienst *Read (Data) Memory By Address* SID = 23 h verwendet werden:

Tab. 6.14 Ausschnitt aus einem COMPARAM-SUBSET

```
<COMPARAM-SUBSET ID="COMPARAM-SUBSET.ISO15765">
   <SHORT-NAME>ISO_15765_COMPARAM</SHORT-NAME>
   <COMPARAMS>
      <COMPARAM CPTYPE="STANDARD"                                    (1)
                ID="COMPARAM-SUBSET.ISO15765.p2min"
                PARAM-CLASS="TIMING">
         <SHORT-NAME>P2min</SHORT-NAME>

         <PHYSICAL-DEFAULT-VALUE>25</PHYSICAL-DEFAULT-VALUE>
         <DATA-OBJECT-PROP-REF>
                ID-REF="COMPARAM-SUBSET.ISO15765.P2MIN"
         </DATA-OBJECT-PROP-REF>
      </COMPARAM>
      ...
   </COMPARAMS>
   <DATA-OBJECT-PROP ID="P2MIN">                                     (2)
      <SHORT-NAME>DOP_P2min</SHORT-NAME>
      ...
      <COMPU-METHOD>
         <CATEGORY>IDENTICAL</CATEGORY>
      </COMPU-METHOD>
      <UNIT ID="Unit_MILISECOND">
         <SHORT-NAME>Unit_MilliSecond</SHORT-NAME>
         <DISPLAY-NAME>ms</DISPLAY-NAME>
      </UNIT>
      ...
   </DATA-OBJECT-PROP>
</COMPARAM-SUBSET>
```

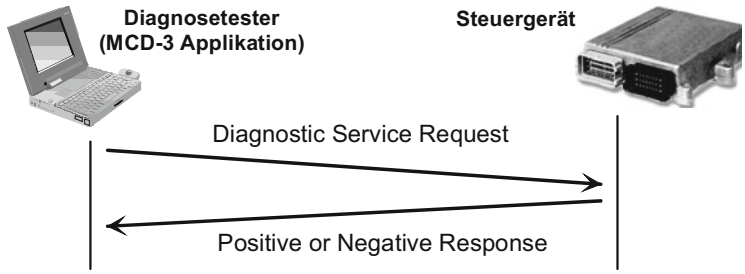

Abb. 6.31 Diagnosedienst

- Falls der Temperaturwert im Steuergerätespeicher ab der Adresse 0524 h steht und 2 Byte lang ist, würde die Anforderungsbotschaft folgendermaßen aussehen:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
SID = 23 h	12 h	05 h	24 h	02 h

- Im Erfolgsfall (Positive Response) erhält der Tester die Antwort

Byte 0	Byte 1	Byte 2
63 h	04 h	E5 h

  dabei ist 63 h die vom Steuergerät wiederholte SID, die in der Antwort protokollgemäß mit gesetztem Bit 6 zurückgesendet wird. Der aktuelle Temperaturwert sei 04E5 h, der wie die Speicheradresse in High-Low-Byte-Reihenfolge (*Big Endian*) übertragen werden soll.
- Im Fehlerfall (Negative Response) könnte die Antwort beispielsweise lauten

Byte 0	Byte 1	Byte 2
7Fh	23 h	31 h

  wobei 7Fh anzeigt, dass es sich um eine Fehlermeldung handelt, 23 h ist die fehlerhafte SID und 31 h steht für den UDS-Fehlercode *Request out of Range*.

Die Struktur der ODX-Beschreibung für eine derartige Botschaft ist in Abb. 6.32 dargestellt. Hauptelement der Diagnosebeschreibung sind DIAG-COMM Objekte, mit denen Diagnosedienste DIAG-SERVICE und Diagnoseabläufe SINGLE-ECU-JOB beschrieben werden. Über die Attribute AUDIENCE und SECURITY-ACCESS-LEVEL kann die Verwendung des Diagnosedienstes für bestimmte Anwendergruppen (z. B. Hersteller, Entwicklung, Fertigung, Werkstatt) zugelassen bzw. verboten werden und von einer bestimmten Sicherheitsstufe abhängig gemacht werden. Wie das Laufzeitsystem die Berechtigung des Anwenders überprüft, ist allerdings im MCD 2/3-Standard nicht definiert. Außerdem lassen sich die Diagnosedienste über DIAGNOSTIC-CLASS so klassifizieren, dass dem Anwender in einer bestimmten Situation nur bestimmte in diesem Kontext sinnvolle Dienste angeboten werden.

Im DIAG-SERVICE-Abschnitt des zugehörigen ODX-Dokumentes wird auf die Anforderungs- und die Antwortbotschaften für den Diagnosedienst verwiesen (Tab. 6.15). Die Botschaften selbst werden in den entsprechenden REQUEST, POS-RESPONSE und NEG-RESPONSE-Abschnitten definiert (1 bis 3). Die in einer Botschaft vorhandenen Daten werden durch PARAM-(Parameter)-Abschnitte beschrieben (Tab. 6.16), wobei ein Datenelement selbst durch seine Byte- und Bit-Position innerhalb der Botschaft sowie einen Querverweis auf ein entsprechendes Datenobjekt DOP (siehe unten) definiert wird, das die zugehörigen Einzelheiten enthält.

Bei der Definition der positiven Antwort (Tab. 6.16) auf die oben beschriebene Temperaturabfrage wird beispielsweise angegeben, dass das Byte 0 der Antwort den Wert 99 = 63 h = 23 h + 40 h haben muss. Da dieser Wert nicht weiter von Interesse ist, wird er vollständig im PARAM-Feld (4) beschrieben. Für den eigentlichen Temperaturwert (5) dagegen wird auf ein separates Datenobjekt DOP verwiesen (6). Im PARAM-Feld selbst

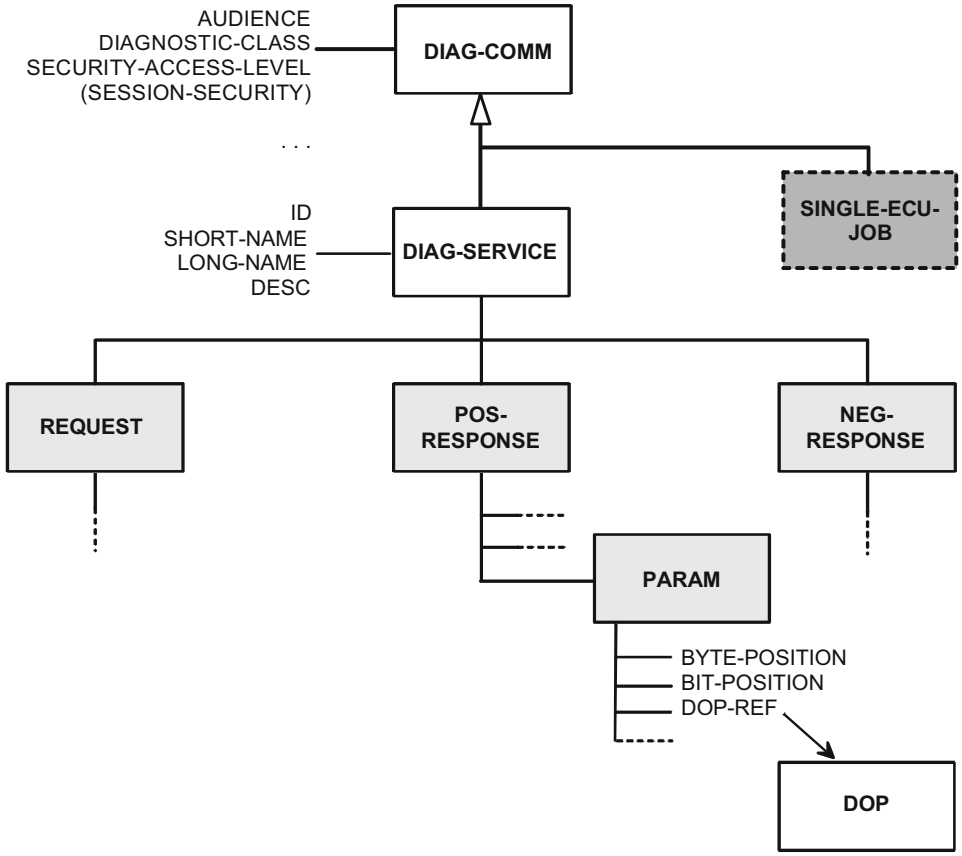

Abb. 6.32 Struktur der ODX-Beschreibung für Diagnosedienste

wird lediglich angegeben, dass der Temperaturwert ab Bit 0 (BIT_POSITION) des Bytes 1 (BYTE-POSITION) beginnt.

Mit Hilfe von Parametern kann das ASAM-Laufzeitsystem somit die Anforderungsbotschaft eines Diagnosedienstes aus DOP-Elementen zusammensetzen bzw. die Antwortbotschaft wieder auflösen. In Tab. 6.17 sind die vordefinierten PARAM-Typen aufgeführt.

DIAG-VARIABLE: Diagnosevariable Beim Messen und Kalibrieren wird direkt auf Größen im Speicher des Steuergerätes zugegriffen. Für den Zugriff wird dabei der steuergeräteinterne Name der Größe verwendet, die mit Hilfe einer vom Compiler/Linker der Steuergerätesoftware erstellten Mapping-Datei einer Speicheradresse im Steuergerät zugeordnet wird. ODX bietet mit den Diagnosevariablen (DIAG-VARIABLE) einen ähnlichen Mechanismus (Tab. 6.18). Im Beispiel wird eine Diagnosevariable DV_PHY_TMOT definiert (1), die der Variablen Tmot in der Steuergerätesoftware entspricht. Das eigentliche Lesen und Schreiben der Variable erfolgt durch die Diagnosedienste DS_ReadMotorTemperature

Tab. 6.15 ODX-Beschreibung eines Diagnosedienstes

```
<DIAG-SERVICE ID="DS_ReadMemoryByAddress">
   <SHORT-NAME>DS_ReadMemoryByAddress</SHORT-NAME>
   . . .
   <REQUEST-REF ID-REF="REQUEST_ReadMemoryByAddress"/>            (1)
   <POS-RESPONSE-REFS>
      <POS-RESPONSE-REF ID-REF="RESP_ReadMemoryByAddress"/>       (2)
   </POS-RESPONSE-REFS>
   <NEG-RESPONSE-REFS>
      <NEG-RESPONSE-REF ID-REF="NEGRESP_IllegalFormat"/>          (3)
   </NEG-RESPONSE-REFS>
   . . .
</DIAG-SERVICE>
```

Tab. 6.16 Beschreibung einer Antwortbotschaft in ODX

```
<POS-RESPONSE ID="RESP_ReadMemoryByAddress">
   <SHORT-NAME>RESP_ReadMemoryByAddress</SHORT-NAME>
   . . .
   <PARAMS>
      <PARAM SEMANTIC="SERVICE-ID" xsi:type="CODED-CONST"/>       (4)
         <SHORT-NAME>POS-RESP-ReadMemoryByAddress</SHORT-NAME>
         <BYTE-POSITION>0</BYTE-POSITION>
         <CODED-VALUE>99</CODED-VALUE>
         <DIAG-CODED-TYPE BASE-DATA-TYPE="A_UINT32"/>
         <BIT-LENGTH>8</BIT-LENGTH>
      </PARAM>
      <PARAM xsi:type="VALUE"/>                                   (5)
         <SHORT-NAME>Temperature</SHORT-NAME>
         <BYTE-POSITION>1</BYTE-POSITION>
         <BIT-POSITION>0</BIT-POSITION>
         <DOP-REF ID-REF="DOP_Temperature"/>                      (6)
      </PARAM>
   </PARAMS>
   . . .
</POS-RESPONSE>
```

bzw. DS_WriteMotorTemperature, auf die in der Beschreibung verwiesen wird (2, 3). Dahinter verbergen sich z. B. die UDS-Diagnosedienste *Read Memory By Address* oder *Write Memory By Address*. Eine MCD-Anwendung kann transparent auf die Diagnosevariable wie auf eine lokale Variable zugreifen, während das MCD-Laufzeitsystem im Hintergrund die Datenübertragung von und zum Steuergerät abwickelt.

6.6.6 Einfache und komplexe Datenobjekte

Simple DOP: Einfache Datenobjekte Kernelement der ODX-Datensätze sind die *einfachen Datenobjekte* (Data Object Property DOP). Sie dienen dazu, die in den Parametern eines Diagnosedienstes verpackten und codierten Informationen eines Steuergerätes korrekt zu interpretieren und die Inhalte für den Menschen verständlich zu machen (Abb. 6.33).

Tab. 6.17 Parametertypen von Diagnosebotschaften

Typ	Bemerkung
VALUE	Normaler Datenwert, verweist auf ein DOP.
RESERVED	Wird verwendet, wenn das Laufzeitsystem den Parameter ignorieren soll.
CODED-CONST	Konstante, z. B. für Service Identifier (SID) verwendet.
PHYS-CONST	Wie CODED-CONST, enthält aber zusätzlich eine Umrechnung in einen physikalischen Wert.
LENGTH-KEY	Wird verwendet, wenn die Parameter-Länge von einem anderen Parameter abhängt.
MATCHING-REQUEST-PARAM	Wird verwendet, wenn ein Parameter der Antwort mit einem Wert der Anforderung verglichen werden muss (z. B. die Local-ID bei einem Diagnose-Dienst).
TABLE-KEY	Wird bei komplexen Datenobjekten (siehe Abschn. 6.6.6) verwendet, wenn Datenstrukturen über Kennwerte (Data Identifier) identifiziert werden wie bei den KWP 2000/UDS-Diagnosediensten *Read/Write Data By Identifier*.
TABLE-STRUCT	
TABLE-ENTRY	
DYNAMIC	Dieser Typ zeigt dem Laufzeitsystem an, dass der Parameter dynamisch definiert wird.
SYSTEM	Dieser Parameter wird verwendet, wenn ein Wert abhängig von System-Informationen berechnet werden soll (z. B. aus der aktuellen Systemzeit)
NRC-CONST	Wie CODED-CONST…

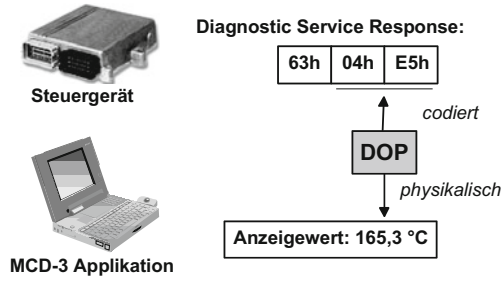

Abb. 6.33 Verwendung eines DOP

Bei den einfachen DOPs unterscheidet ODX zwei Typen:

- Der gewöhnliche DOP bildet die Basis aller anderen Datenobjekte.
- Der DTC-DOP (Diagnostic Trouble Code) beschreibt Fehlerspeichereinträge.

Jedem DOP wird zur Unterscheidung ein eindeutiger Identifier (ID) und eine Kurzbezeichnung (SHORT-NAME) zugeordnet. Mit einem optionalen LONG-NAME und einer DESCription wird ein DOP im Klartext beschrieben. Ein DOP besteht aus den in Tab. 6.19 aufgeführten Elementen.

Abbildung 6.34 zeigt als Beispiel ein DOP für einen Temperaturwert. Für das codierte, d. h. Steuergeräte-interne Format (DIAG-CODED-TYPE) wurde im Beispiel eine vor-

Tab. 6.18 Beschreibung einer Diagnosevariablen

```
<DIAG-VARIABLES>
  <DIAG-VARIABLE ID="DV_PHY_TMOT" IS-READ-BEFORE-WRITE="false">   (1)
    <SHORT-NAME>DV_PHY_TMOT</SHORT-NAME>
    <SW-VARIABLES>
      <SW-VARIABLE>
        <SHORT-NAME>Tmot</SHORT-NAME>
        <ORIGIN>A</ORIGIN>
      </SW-VARIABLE>
    </SW-VARIABLES>
    <COMM-RELATIONS>
      <COMM-RELATION VALUE-TYPE="CURRENT">
        <RELATION-TYPE>READ</RELATION-TYPE>
        <DIAG-COMM-SNREF SHORT-NAME="DS_ReadMotorTemperature"/>   (2)
          <OUT-PARAM-IF-SNREF SHORT-NAME="STAT_PHY_TMOT"/>
      </COMM-RELATION>
      <COMM-RELATION VALUE-TYPE="CURRENT">
        <RELATION-TYPE>WRITE</RELATION-TYPE>
        <DIAG-COMM-SNREF SHORT-NAME="DS_WriteMotorTemperature"/>  (3)
        <IN-PARAM-IF-SNREF SHORT-NAME="ARG_PHY_TMOT"/>
      </COMM-RELATION>
    </COMM-RELATIONS>
  </DIAG-VARIABLE>
</DIAG-VARIABLES>
```

Tab. 6.19 Elemente eines einfachen ODX-Datenobjekts DOP

DIAG-CODED-TYPE	Datentyp des codierten Wertes eines Datums.
INTERNAL-CONSTR	Grenzwerte für das Datum im codierten (internen) Format, z. B. Monatsangabe in hexadezimaler Schreibweise. Bsp.: 01 h ≤ Monat ≤ 0Ch, alle anderen Werte sind ungültig.
COMPU-METHOD	Umrechnungsmethode zwischen dem codierten Wert und der physikalischen Größe.
PHYSICAL-TYPE	Datentyp für den physikalischen Wert eines Datums.
UNIT	Einheit zum physikalischen Wert (°C, m/s, km/h, …).

zeichenlose 32 bit Ganzzahl des vordefinierten Datentyps A_UINT32 (unsigned integer 32 bit) gewählt. Mittels der Umrechnungsformel (COMPU-METHOD) kann das codierte (interne) Format in das physikalische Format (PHYSICAL-TYPE) umgerechnet werden, für das eine 32 bit Gleitkommazahl des Typs A_FLOAT32 verwendet wird. Die Einheit (UNIT) des physikalischen Wertes ist °C. Bei der Wandlung zwischen codiertem Format und physikalischem Wert können zusätzlich noch Grenzen (INTERNAL-CONSTR) für den Wertebereich der codierten Größe festgelegt werden. In Abb. 6.34 beträgt der steuergeräteinterne Wert 4E5 h. Dieser Wert liegt innerhalb des zulässigen Wertebereichs von 0 bis 2400 = 960 h. Er kann über die Umrechnungsformel $y = 40 + 0.1\ x = 40 + 0.1 \cdot 4E5\,\text{h}$ korrekt in den physikalischen Wert 165,3 umgerechnet und in der Einheit °C angezeigt werden.

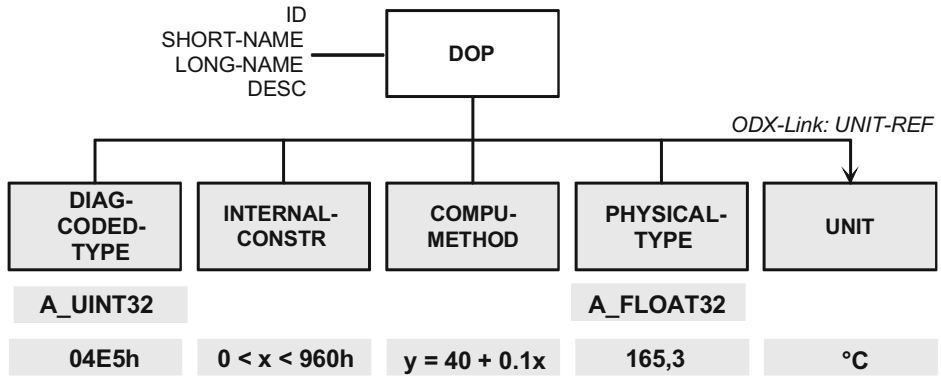

Abb. 6.34 Beispiel eines einfachen Datenobjekts DOP

Tabelle 6.20 zeigt den zugehörigen ODX-Datensatz, dessen Details im Folgenden erläutert werden sollen. Die in ( ) stehenden Ziffern beziehen sich auf die entsprechenden Nummern im Datensatz.

(1.) Aus Sicht des ASAM-Laufzeitsystems bilden alle zwischen Steuergerät und Tester in einer Diagnosebotschaft enthaltenen Daten zunächst einen Datenstrom, der zur Interpretation in seine Bestandteile zerlegt werden muss. Dazu benötigt das Laufzeitsystem die Information, wo ein Datum innerhalb des Datenstroms beginnt und wie lang es ist. Die Information über die Position im Datenstrom wurde bereits bei Tab. 6.16 erläutert. Die Länge bzw. die Art der Längenberechnung des Datums selbst wird über entsprechende LENGTH-INFO-TYPES festgelegt (Abb. 6.35). Durch diese Informationen werden die Daten aus dem Datenstrom separiert und im entsprechenden internen Datenformat (DIAG-CODED-TYPE) zwischengespeichert. Folgende Datentypen stehen dabei zur Verfügung:

Tab. 6.20 ODX-Datensatz für das Beispiel nach Abb. 6.34

```
<DATA-OBJECT-PROP ID="DOP_Temperature">
   <SHORT-NAME>MotTemp</SHORT-NAME>
   <LONG-NAME>Motortemperatur</LONG-NAME>
   <DESC>Kühlwassertemperatur in Grad Celsius</DESC>

   <DIAG-CODED-TYPE                                                    (1)
      BASE-DATA-TYPE="A_INT32"
      xsi:type="STANDARD-LENGTH-TYPE"
      IS-HIGH-LOW-BYTE-ORDER="true">

      <BIT-LENGTH>16</BIT-LENGTH>
   </DIAG-CODED-TYPE>

   <INTERNAL-CONSTR>                                                   (2)
      <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
      <UPPER-LIMIT INTERVAL-TYPE="CLOSED">2400</UPPER-LIMIT>
   </INTERNAL-CONSTR>

   <PHYSICAL-TYPE BASE-DATA-TYPE="A_FLOAT32" DISPLAY-RADIX="DEC">
      <PRECISION>1</PRECISION>                                         (3)
   </PHYSICAL-TYPE>

   <COMPU-METHOD>                                                      (4)
      <CATEGORY>LINEAR</CATEGORY>
      <COMPU-INTERNAL-TO-PHYS>
         <COMPU-SCALES>
            <COMPU-SCALE>
               <COMPU-RATIONAL-COEFFS>

                  <COMPU-NUMERATOR>
                     <V>400</V>
                     <V>1</V>
                  </COMPU-NUMERATOR>
                  <COMPU-DENUMERATOR>
                     <V>10</V>
                  </COMPU-DENUMERATOR>
               <COMPU-RATIONAL-COEFFS>
            </COMPU-SCALE>
         </COMPU-SCALES>
      </COMPU-INTERNAL-TO-PHYS>
   </COMPU-METHOD>

   <UNIT-SPEC>                                                         (5)
      <UNITS>
         <UNIT ID="Unit_Celsius">
            <SHORT-NAME>Unit_Celsius</SHORT-NAME>
            <DISPLAY-NAME>°C</DISPLAY-NAME>
         </UNIT>
         ...
      </UNITS>
   <UNIT-SPEC>

</DATA-OBJECT-PROP>
```

LENGTH-INFO-TYPE	Beschreibung
STANDARD-LENGTH-TYPE	Die Länge des Datenwerts wird explizit durch BIT-LENGTH spezifiziert.
LEADING-LENGTH-INFO-TYPE	Die Längeninformation steht im Datenbyte vor dem eigentlichen Datenwert innerhalb der PDU.
PARAM-LENGTH-INFO-TYPE	Die Längeninformation steht in einem anderen Parameter. Auf diesen zweiten Parameter wird hier nur referenziert.
MIN-MAX-LENGTH-TYPE	Für Datenwerte variabler Länge, wobei ein als TERMINATION definiertes Byte den Datenwert abschließt. Außerdem wird die minimale (MIN-LENGTH) und maximale (MAX-LENGTH) Datenlänge angeben. Mögliche Werte für TERMINATION sind: ZERO: Der zu extrahierende Bytestrom endet mit 00 h HEX-FF: Der zu extrahierende Bytestrom endet mit FFh END-OF-PDU: Es gibt keinen festgelegten Wert für das Ende des Bytestromes. Das Ende ist durch das Ende der Botschaft gegeben.

Wenn der Datenwert mehrere Bytes umfasst, muss die Reihenfolge der Datenbytes angegeben werden. Die beiden möglichen Formate High-Byte-First (Motorola Big-Endian-Format) und Low-Byte-First (Intel Little-Endian-Format) werden mit dem Parameter IS-HIGH-LOW-BYTE-ORDER (TRUE/FALSE) unterschieden.

(2.) Der Gültigkeitsbereich von internen Werten (DIAG-CODED-TYPE) kann durch Grenzwerte (INTERNAL-CONSTR) eingeschränkt werden. Diese Einschränkung umfasst sowohl einen unteren Grenzwert (LOWER-LIMIT) als auch einen oberen Grenzwert (UPPER-LIMIT). Darüber hinaus können Sub-Intervalle (SCALE-CONSTR) definiert werden, mit deren Hilfe gültige, ungültige, nicht definierte oder nicht verfügbare Bereiche festgelegt werden. Jeder interne Wert wird zunächst gegen die Grenzwerte getestet. Die Umrechnung in den physikalischen Wert erfolgt nur, wenn der Wert als gültig erkannt wird. Im Beispiel ist der Wertebereich der gültigen Temperaturwerte 0 … 2400 = 960 h und somit ein geschlossenes Intervall (INTERVAL-TYPE = “CLOSED”).

(3.) PHYSICAL-TYPE beschreibt den Datentyp für die physikalische Darstellung des internen Wertes. Die optionale Information DISPLAY-RADIX gibt an, in welchem Zahlenformat der Wert angezeigt werden soll. Möglich ist dabei eine hexadezimale (HEX), eine dezimale (DEC), eine oktale (OCT) oder binäre (BIN) Darstellung. PRECISION definiert die Anzahl der Nachkommastellen.

(4.) Die Umrechnung des internen Wertes DIAG-CODED-TYPE in den physikalischen Wert PHYSICAL-TYPE erfolgt über die im Feld COMPU-METHOD angegebene Umrechnungsmethode. Insgesamt gibt es acht Methoden CATEGORY:

CATEGORY	*Beschreibung*
IDENTICAL	Interner und physikalischer Wert sind identisch, keine Umrechnung erforderlich.
LINEAR	Linearer Zusammenhang nach der Formel: $y = \frac{v_{N0} + v_{N1} \cdot x}{v_{D0}}$, Einzelheiten siehe RAT-FUNC
SCALE-LINEAR	Abschnittweise lineare Funktionen. Jeder Abschnitt wird als COMPU-SCALE mit Unter- und Obergrenze spezifiziert.
TAB-INTP	Wertetabelle mit linearer Interpolation
TEXTTABLE	Umsetzung eines Zahlenwertes in einen Textwert.
COMPUCODE	Wird verwendet, wenn eine in der Programmiersprache JAVA geschriebene Funktion die Umrechnung durchführen soll.
RAT-FUNC	Allgemeine rationale Funktion nach der Formel $y = \frac{v_{N0} + v_{N1} \cdot x + v_{N2} \cdot x^2 + \ldots + v_{Nm} \cdot x^m}{v_{D0} + v_{D1} \cdot x + v_{D2} \cdot x^2 + \ldots + v_{Dn} \cdot x^n}$ dabei ist x der interne und y der physikalische Wert. Die Parameter v_{N0}, v_{N1}, …, v_{Nm} des Zählerpolynoms (COMPU-NUMERATOR) und v_{D0}, v_{D1}, …, v_{Dn} des Nennerpolynoms (COMPU-DENUMERATOR) sind Konstanten.
SCALE-RAT-FUNC	Abschnittweise rationale Funktionen.

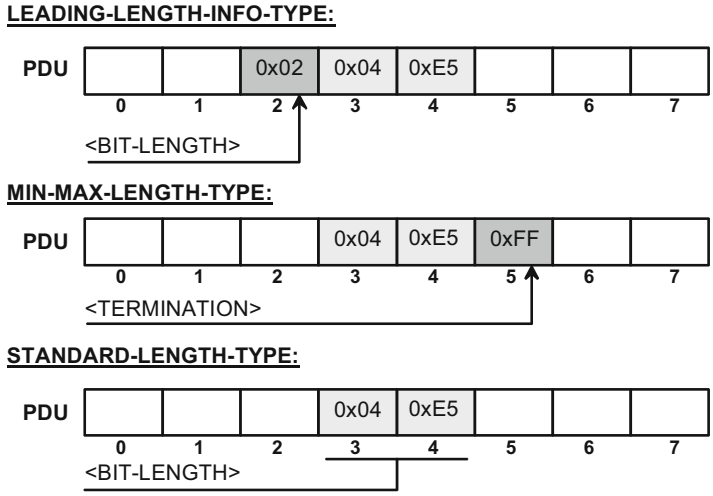

Abb. 6.35 Festlegung der Länge eines Datenfeldes in einer Diagnosebotschaft

Im obigen Beispiel für den Temperaturwert wird eine lineare Umrechnung mit

$$y = 40 + 0.1\,x = \frac{400 + 1\,x}{10}$$

verwendet. Die Zählerkoeffizienten sind 400 und 1, der Nennerkoeffizient ist 10.

(5.) Im Abschnitt UNIT-SPEC wird dem physikalischen Wert eine physikalische Einheit zugeordnet. ODX bietet die Möglichkeit, denselben physikalischen Wert in unterschiedlichen Einheiten darzustellen, z. B. km/h und mph. Hierfür wird eine Größe zunächst in den SI-Basiseinheiten, z. B. m/s, dargestellt. Andere Einheiten werden daraus über einen Faktor und einen Offsetwert abgeleitet:

$$\textit{Einheit} = \textit{SI} - \textit{Basiseinheit} \times \text{FACTOR-SI-TO-UNIT} + \text{OFFSET-SI-TO-UNIT}$$

Beispiel: Umrechnung von km/h in mph über die SI-Einheit m/s

$$[\text{m/s}] = [\text{km/h}] \times 1/3.6$$
$$\text{mph} = [\text{m/s}] \times 2{,}23741 + 0$$

DISPLAY-NAME beschreibt, wie die Einheit in der Diagnoseanwendung über die MCD 3-Schnittstelle visuell dargestellt werden soll. Physikalische Werte, die nicht als SI-Einheit sondern direkt gespeichert sind, wie im Beispiel die Temperatur in °C, verwenden nur den DISPLAY-NAME.

DTC-DOP: Datenobjekte für Fehlerspeichereinträge In der Regel legen Steuergeräte beim Auftreten eines internen Fehlers neben dem Fehlercode (*Diagnostic Trouble Code DTC*) zusätzliche Informationen über die Umgebungsbedingungen wie Motordrehzahl oder Motortemperatur ab. Damit soll der aufgetretene Fehler möglichst exakt dokumentiert werden. In ODX werden die Fehlercodes in *Diagnostic Trouble Code Datenobjekten* (DTC-DOP) beschrieben (Tab. 6.21), die neben den Informationen eines gewöhnlichen DOP noch zusätzliche DTC-spezifische Informationen wie den internen Fehlercode (TROUBLE-CODE), den für die Anzeige zu verwendenden Fehlercode (DISPLAY-TROUBLE-CODE), eine Fehlermeldung im Klartext (TEXT) und einen Wert (LEVEL) für die Klassifizierung des Fehlers nach Schwere oder Art enthalten. Die zugehörigen Umgebungsbedingungen werden als komplexe Datenobjekte (siehe unten) gespeichert.

Complex DOP: Komplexe Datenobjekte Komplexe Datenobjekte sind aus einfachen Datenobjekten zusammengesetzte Strukturen (Tab. 6.22). Sie werden für die Definition und Interpretation von komplexen Parametern von Diagnosediensten verwendet, wenn die Art und die Zusammensetzung der Parameter erst zur Laufzeit des Systems bestimmt werden können. Complex DOPs werden in derselben Weise durch PARAM-Abschnitte referenziert wie gewöhnliche DOPs.

Tab. 6.21 Beispiel eines DTC-DOP

```
<DTC-DOP ID="DTCDOP_ALL">
   <SHORT-NAME>DTCDOP_ALL</SHORT-NAME>
   . . .
   <DTCS>
      <DTC ID="DTC0140">
         <SHORT-NAME>DTC0140</SHORT-NAME>
         <TROUBLE-CODE>320</TROUBLE-CODE>
         <DISPLAY-TROUBLE-CODE>Error_140</DISPLAY-TROUBLE-CODE>
         <TEXT>Ashtray Illumination Defect</TEXT>
         <LEVEL>2</LEVEL>
      </DTC>
      . . .
   </DTCS>
</DTC-DOP>
```

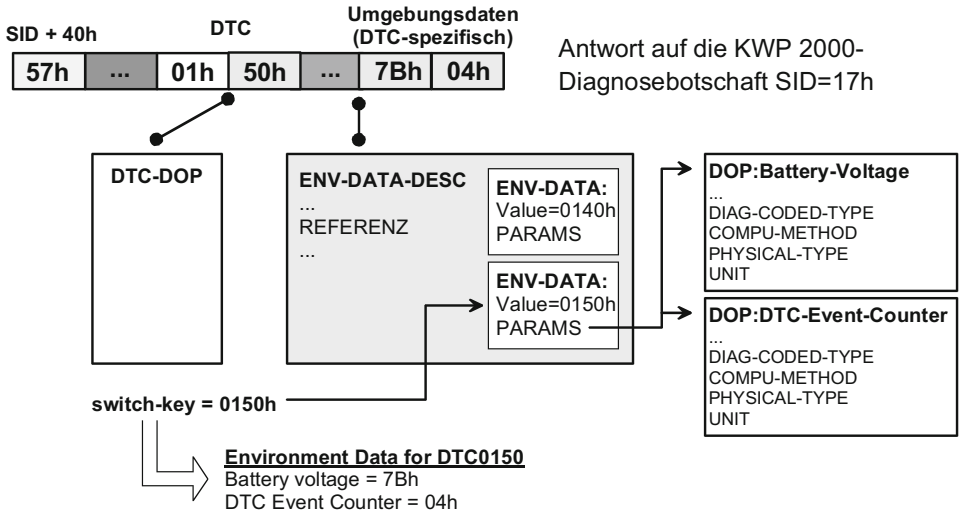

Abb. 6.36 Auslesen von Umgebungsbedingungen aus dem Fehlerspeicher

Das nachfolgende Beispiel zeigt den Aufbau eines komplexen Datenobjekts für die Dokumentation eines Fehlerspeichereintrages (DTC – Diagnostic Trouble Code). Ein Steuergerät legt für die spätere Analyse eines Fehlers die aktuell herrschenden Umgebungsbedingungen mit ab. Diese Daten sind eine Ansammlung unterschiedlicher physikalischer Größen, die strukturiert abgelegt werden müssen. Als Beispiel zeigt Abb. 6.36, wie der Diagnosetester die Umgebungsbedingungen zum Fehler mit der Fehlernummer DTC = 336 = 150 h ausliest. Die einzelnen Umgebungsbedingungen, in diesem Fall die Batteriespannung und ein Fehlerzähler (Event Counter), werden vom Steuergerät als Parameter hintereinander in der Antwortbotschaft zurückgesendet.

In Tab. 6.23 ist die zugehörige ODX-Beschreibung dargestellt. ODX sieht für Umgebungsbedingungen einen ENV-DATA-DESC-Abschnitt mit ENV-DATA-Feldern vor. Die-

Tab. 6.22 Typen von komplexen Datenobjekten in ODX

Type	Beschreibung
STRUCTURE	Vergleichbar mit einer Struktur innerhalb der Programmiersprache C, Zusammenfassung mehrerer DOPs (simple oder complex). Bsp.: Lese den letzten Eintrag des Steuergeräte-Historienspeichers für die Reprogrammierung.
STATIC-FIELD	Wird verwendet, wenn sich eine STRUCTURE innerhalb eines Bytestroms mehrfach mit fester Anzahl wiederholt. Bsp.: Lese die letzten drei Einträge des Steuergeräte-Historienspeichers für die Reprogrammierung.
DYNAMIC-LENGTH-FIELD	Wird verwendet, wenn sich eine STRUCTURE innerhalb eines Bytestroms wiederholt und die Anzahl der Wiederholungen nur dynamisch, d. h. erst zur Laufzeit, bekannt ist. Bsp.: Lese alle Einträge des Steuergeräte-Historienspeichers für die Reprogrammierung.
DYNAMIC-ENDMARKER-FIELD	Ein DYNAMIC-ENDMARKER-FIELD wird verwendet, wenn sich eine STRUCTURE innerhalb eines Bytestroms solange wiederholt, bis ein definierter TERMINATION-VALUE erkannt wird. Bsp.: Lese alle Einträge des Steuergeräte-Historienspeichers für die Reprogrammierung, wobei das Steuergerät als Abschlusskennzeichnung den Wert FFh sendet.
MUX	Ein Multiplexer (MUX) wird verwendet, wenn die Interpretation eines Datenstroms von einem bestimmten Parameter (SWITCH-KEY) abhängt. Über den SWITCH-KEY wird ein entsprechender CASE selektiert, der wiederum auf eine Parameterstruktur verweist. So können für alle möglichen Sub-Identifier eines Diagnosedienstes die jeweils definierten Parameterstrukturen modelliert werden.
END-OF-PDU-FIELD	Ein END-OF-PDU-FIELD wird verwendet, wenn sich eine STRUCTURE innerhalb eines Bytestroms bis zu dessen Ende wiederholt.
TABLE	Wird verwendet, wenn Datenstrukturen über Kennwerte (Data Identifier) identifiziert werden wie bei den KWP 2000/UDS-Diagnosediensten *Read/Write Data By Identifier*. Eine TABLE besteht aus einer KEY- und einer STRUCTURE-Spalte. Die KEY-Spalte beinhaltet die entsprechenden Data Identifier des Diagnosedienstes. Die STRUCTURE-Spalte verweist auf die zugehörigen Datenstrukturen, deren Inhalte in entsprechenden STRUCTURE-Abschnitten definiert sind.

ses komplexe Datenobjekt enthält die Umgebungsbedingungen für mehrere Fehler und ist daher als Multiplexer (MUX, vgl. Tab. 6.22) aufgebaut. Zur Auswahl (SWITCH-KEY) dient die Fehlernummer, die über DTC-VALUE mit 336 = 150 h festgelegt ist. Die ODX-Datei gibt an, dass zu diesem Fehlercode die Umgebungsparameter Batteriespannung und Fehlerzähler gehören und verweist (DOP-REF) auf die zugehörigen einfachen Datenobjekte, über die die Umrechnung zwischen hexadezimalem und physikalischem Wert, der Wertebereich, die Einheit usw. festgelegt werden. Am Beispiel der DTC-DOPs lässt sich wieder

Tab. 6.23 ODX-Beschreibung zu Abb. 6.36

```
<ENV-DATA-DESC ID="ENVDESC_EnvDataDesc1">
   <SHORT-NAME>ENVDESC_EnvDataDesc1</SHORT-NAME>
   <PARAM-SNREF SHORT-NAME="SwitchKeyDTC">
   <ENV-DATAS>
      <ENV-DATA ID="ED_1">
         <SHORT-NAME>ED_DTC0140</SHORT-NAME>
         . . .
      </ENV-DATA>
     <ENV-DATA ID="ED_2">
         <SHORT-NAME>ED_DTC0150</SHORT-NAME>
         <DTC-VALUES>
            <DTC-VALUE>336<DTC-VALUE>
         </DTC-VALUES>
         <PARAMS>
            <PARAM xsi:type="VALUE">
               <SHORT-NAME>BatteryVoltage</SHORT-NAME>
               <BYTE-POSITION>0</BYTE-POSITION>
               <DOP-REF ID-REF="DOP_BatteryVoltage">
            </PARAM>
            <PARAM xsi:type="VALUE">
               <SHORT-NAME>EventCounter</SHORT-NAME>
               <BYTE-POSITION>1</BYTE-POSITION>
               <DOP-REF ID-REF="DOP_EventCounter">
            </PARAM>
         </PARAMS>
      </ENV-DATA>
   </ENV-DATAS>
</ENV-DATA-DESC>
```

das Single-Source-Prinzip von ODX erkennen. Sind Fehlercodes bereits an einer Stelle definiert, so wird an einer anderen Stelle nur auf die dortige Definition verwiesen (Referenz).

Special Data Groups: Herstellerspezifische Daten Über sogenannte SDG-Datenobjekte kann ein Hersteller Datensätze in ODX-Beschreibungen aufnehmen, für die der Standard keine Datenobjekte vorgesehen hat. Standard-ODX-Werkzeuge dürfen solche herstellerspezifischen Erweiterungen ignorieren.

6.6.7 SINGLE-ECU-JOB und MULTIPLE-ECU-JOB: Diagnoseabläufe

Häufig müssen bei der Diagnose bestimmte Abläufe ausgeführt werden, die aus der sequenziellen Abarbeitung verschiedener Diagnosebotschaften bestehen, wobei die Parameter oder die Art der Folgebotschaften von den Ergebnissen der vorigen Botschaften abhängt. Ein Beispiel dafür ist der *Seed-and-Key*-Mechanismus, mit dem der Zugriff auf bestimmte Steuergerätefunktionen freigeschaltet wird. Dabei fordert der Diagnosetester vom Steuergerät mit einer ersten Botschaft einen Initialisierungswert an, berechnet daraus dann

Tab. 6.24 Single-ECU-Job: ODX-Beschreibung und Java-Programm

ODX-Beschreibung:

```
<SINGLE-ECU-JOB ID="ID_734711" DIAGNOSTIC-CLASS="COMMUNICATION">
   . . .
   <PROG-CODES>
      <PROG-CODE>                    //Verweis auf JAVA-Programm
         <CODE-FILE>Beispiel.java</CODE-FILE>
         . . .
      </PROG-CODE>
   </PROG-CODES>
   <INPUT-PARAMS>
      <INPUT-PARAM>
         <SHORT-NAME>Address</SHORT-NAME>
         <DOP-BASE-REF ID-REF="DOP-3BYTE-IDENTICAL"/>
      </INPUT-PARAM>
      . . .
   </INPUT-PARAMS>
   <OUTPUT-PARAMS>
      <OUTPUT-PARAM ID="ID_734712">
         <SHORT-NAME>Returned_Value</SHORT-NAME>
         <DOP-BASE-REF ID-REF="DOP-ByteArray"/>
      </OUTPUT-PARAM>
   </OUTPUT-PARAMS>
</SINGLE-ECU-JOB>
```

einen Schlüssel und sendet diesen an das Steuergerät zurück. Für derartige Szenarien sieht ODX sogenannte *Jobs* vor, mit denen ein solcher Ablauf definiert werden kann.

Falls ein solcher *Job* Botschaften an mehrere Steuergeräte umfasst, muss er im separaten ODX-Dokument MULTIPLE-ECU-JOB definiert werden, ansonsten wird er als SINGLE-ECU-JOB einem DIAG-COMM-Element zugeordnet (Abb. 6.32). Bei einem MULTIPLE-ECU-JOB muss das Laufzeitsystem dann mehrere gleichzeitige logische Verbindungen (Logical Links) zu den einzelnen Steuergeräten aufbauen. Ein typischer Anwendungsfall für einen MULTIPLE-ECU-JOB ist die Identifikation aller Geräte eines Fahrzeugs. Dabei wird gleichzeitig mit allen Steuergeräten kommuniziert, um deren Identifikationsdaten auszulesen. Anstatt vom Diagnosesystem nacheinander alle Geräte abzufragen und die Informationen in der Bedienebene oberhalb des Laufzeitsystems zu sammeln, kann durch einen MULTIPLE-ECU-JOB bereits im Laufzeitsystem eine Bündelung der Daten erfolgen.

Der eigentliche Ablauf eines *Jobs* wird in einer separaten Datei in der Programmiersprache Java beschrieben, auf die in der ODX-Datei verwiesen wird (Tab. 6.24). Darüber hinaus definiert die ODX-Datei, welche Parameter an die Java-Funktion zu übergeben und wie deren Rückgabewerte zu interpretieren sind. Für derartige Java-Abläufe stellt ASAM ein Java-Paket asam.d.* zur Verfügung, das die im Beispielprogramm eingesetzten, mit MCD_… beginnenden Hilfsfunktionen und Klassendefinitionen enthält. Optional kann ein ASAM-Laufzeitsystem herstellerabhängig auch andere Programmiersprachen unterstützen, z. B. in C/C++ erstellte Windows-DLL-Funktionen.

JAVA-Programm in Datei Beispiel.java:

```
import asam.d.*

public class JobExample implements JobTemplate
{ . . .
  public void execute( MCDRequestParameters inputParameters,
                       MCDJobApi jobHandler, MCDLogicalLink link,
                       MCDSingleEcuJob apiJobObject
                     ) throws MCDException
  {  MCDService service;
     MCDResult serviceResult, jobResult;
     MCDResultState resultState;
     MCDResponse serviceResponse, jobResponse;
     MCDResponseParameter value;

// Create the ReadMemoryByAddress diagnostic service to be executed
     service = link.createService(link.getDbObject()
                                       .getDBLocation.getServices()
                                  .getItemByName("ReadDataByAddress"));

// Set variable request parameters of the service
     service.setParameterValue("Address", inputParameters
                                 .getItemByName("Address").getValue());
     . . .
     resultState = service.executeSync(); // Execute the service
     serviceResult = resultState.getResults().getItemByIndex(1);
     serviceResponse = serviceResult.getResponses()
                                                  .getItemByIndex(1));
// Create the job result and response and copy the second service
// response parameter into the job response
     jobResult   = apiJobObject.createResult(. . .);
     jobResponse = jobResult.getResponses().add(. . .);
     jobResponse.getResponseParameters.addElementWithContent(
          serviceResponse.getResponseParameters().getItemByIndex(2));
     link.deleteComPrimitive(service);       // Delete the service
     jobHandler.sendFinalResult(jobResult);  // Return result
     return;
  }
}
```

6.6.8 STATE-CHART: Diagnosesitzungen

ODX bildet das Konzept der Diagnosesitzungen (vgl. Abschn. 5.1.2) auf einen Zustandsautomaten (STATE-CHART) ab, der Teil eines DIAG-LAYER-Elements (Abschn. 6.6.2) ist. Für diesen Zustandsautomaten werden Zustände (STATE) und Übergänge zwischen den Zuständen (STATE-TRANSITION) definiert (Abb. 6.37)

Bei den Diagnosediensten und -abläufen (DIAG-COMM-Elemente) kann durch einen Verweis (STATE-TRANSITION-REF) auf einen entsprechenden Zustandsübergang angegeben werden, dass der Zustandsautomat in einen anderen Zustand wechseln soll, wenn der Diagnosedienst erfolgreich ausgeführt wurde. Umgekehrt kann mit Hilfe von PRE-CONDITION-STATE-REF-Verweisen auf einen oder mehrere Zustände festgelegt werden,

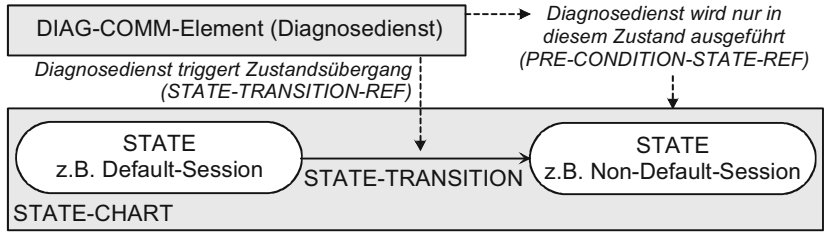

Abb. 6.37 Zustandsautomat für Diagnosesitzungen

dass der Diagnosedienst nur ausgeführt werden darf, wenn sich das Steuergerät in den dabei definierten Zuständen befindet.

6.6.9 ECU-CONFIG: Beschreibung der Steuergeräte-Konfiguration

Unter der Konfiguration oder Parametrisierung eines Steuergerätes (umgangssprachlich als *Codieren* bezeichnet) versteht man die Aktivierung bestimmter interner Einstellungen oder Funktionalitäten eines Steuergerätes. So kann zum Beispiel bei einem Steuergerät für das Außenlicht konfiguriert werden, ob ein Glühfaden-Scheinwerfer oder ein Xenon-Scheinwerfer angesteuert werden soll. Bei der Konfiguration eines Steuergerätes wird mittels eines Diagnosedienstes (z. B. UDS *WriteDataByAddress*, siehe Kap. 5) ein bestimmter Datensatz an eine definierte Adresse im Steuergerät geschrieben. Das Steuergerät wertet dann zur Laufzeit das Bitmuster des Datensatzes aus und setzt intern die entsprechende Konfiguration. Ein solcher Datensatz besteht aus der seriellen Anordnung der einzelnen Informationen und kann, je nach Anzahl der zu konfigurierenden Parameter, zwischen einem Bit und mehreren Byte lang sein. Das ODX-Datenmodell bietet abhängig vom Anwendungsfall zwei Möglichkeiten für den Umgang mit solchen Datensätzen.

Während der Entwicklung werden die Konfigurationsdatensätze je nach gewünschter Funktionalität einzeln zusammengesetzt (Abb. 6.38). Daher muss in ODX die Bedeutung jedes einzelnen Elementes (CONFIG-ITEM) des Datensatzes (CONFIG-RECORD) hinterlegt sein.

In der Produktion sind die Datensätze für ein Steuergerät in einem bestimmten Fahrzeug in der Regel bereits erstellt. Die Konfiguration (CONFIG-RECORD) wird hier als kompletter Datensatz (DATA-RECORD) mit einem Diagnosedienst ins Steuergerät geschrieben.

Mit einem weiteren Diagnosedienst kann die Konfiguration auch aus dem Steuergerät ausgelesen werden. Durch die Wandlung der einzelnen CONFIG-ITEMs in ihren physikalischen Wert (DOP) kann der Konfigurationsstand des Steuergerätes ermittelt werden.

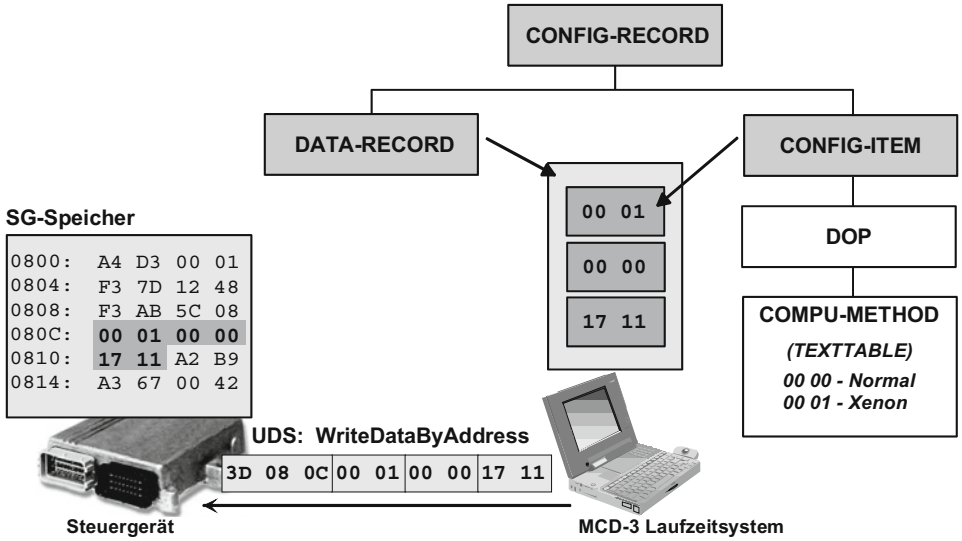

Abb. 6.38 Vereinfachte Darstellung für die Steuergeräte-Konfiguration

6.6.10 ECU-MEM: Beschreibung der Flash-Programmierung

Wie oben bereits erwähnt, ist das Reprogrammieren von Steuergeräten, umgangssprachlich als *Flashen* bezeichnet, ebenfalls Bestandteil der Steuergeräte-Diagnose. Dabei wird durch eine definierte Abfolge von Diagnosediensten der Flash-Speicher des Steuergerätes gelöscht und neue Software programmiert. Eine ausführliche Beschreibung dieses Vorgangs erfolgt in Abschn. 9.4. ODX beschreibt innerhalb des ECU-MEM-Dokumentes einen Datencontainer, in dem alle relevanten Informationen für die Durchführung eines solchen Programmiervorgangs auf der Seite des Diagnosetesters abgelegt sind. Innerhalb dieses Datencontainers definiert ODX sogenannte SESSIONS, mit denen die zu programmierenden Daten verwaltet werden. Abbildung 6.39 zeigt den prinzipiellen Aufbau dieser Verwaltung am Beispiel des Tacho-Steuergerätes mit der Variante für Europa (km/h) und USA (mph).

Die Steuergerätesoftware besteht aus dem Codeteil (Applikation) und dem Datenteil (Parametersatz). Die Applikation ist für beide Varianten gleich, der Parametersatz dagegen ist in der *mph*- und in der *km/h*-Variante unterschiedlich. Zusätzlich benötigt das Steuergerät zur Durchführung der Programmierung noch einen Flash-Treiber, der dynamisch in das Steuergerät geladen wird und dort den eigentlichen Programmiervorgang durchführt. Diese vier Dateien werden in der ODX-Beschreibung durch das Anlegen von zwei SESSIONs (Programmiersitzungen) logisch gegliedert. In der *Session 1* wird für die USA-Variante auf die Dateien *Parametersatz mph*, *Applikation* und *Flash-Treiber* referenziert, *Session 2* (Europa-Variante) dagegen verweist auf *Parametersatz km/h*, *Applikation* und *Flash-Treiber*. Das Laufzeitsystem hat nach Auswahl der entsprechenden SESSION Zugriff auf alle relevanten Informationen.

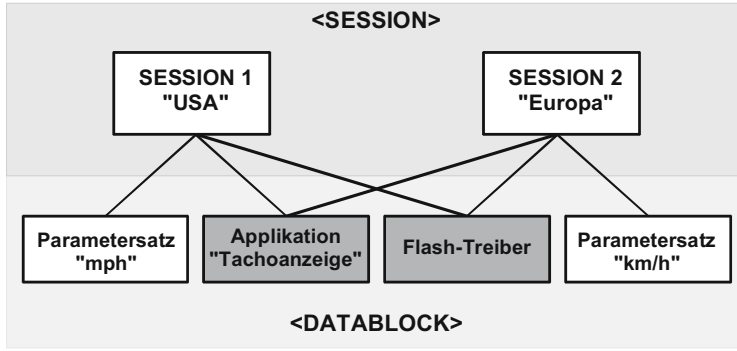

Abb. 6.39 Programmierdatensätze für ein Tachometer-Steuergerät

Abbildung 6.40 zeigt stark vereinfacht den Aufbau des Datencontainers. Ein DATABLOCK beschreibt die Struktur des Adressbereiches, der zu programmieren ist. Dazu werden zusammenhängende Datenblöcke der Programmierdaten (FLASHDATA) innerhalb von ODX auf SEGMENTe abgebildet und durch SOURCE-START-ADDRESS und SOURCE-END-ADDRESS abgegrenzt. Jeder DATABLOCK besitzt mindestens ein SEGMENT. Tabelle 6.25 zeigt die zugehörige ODX-Beschreibung.

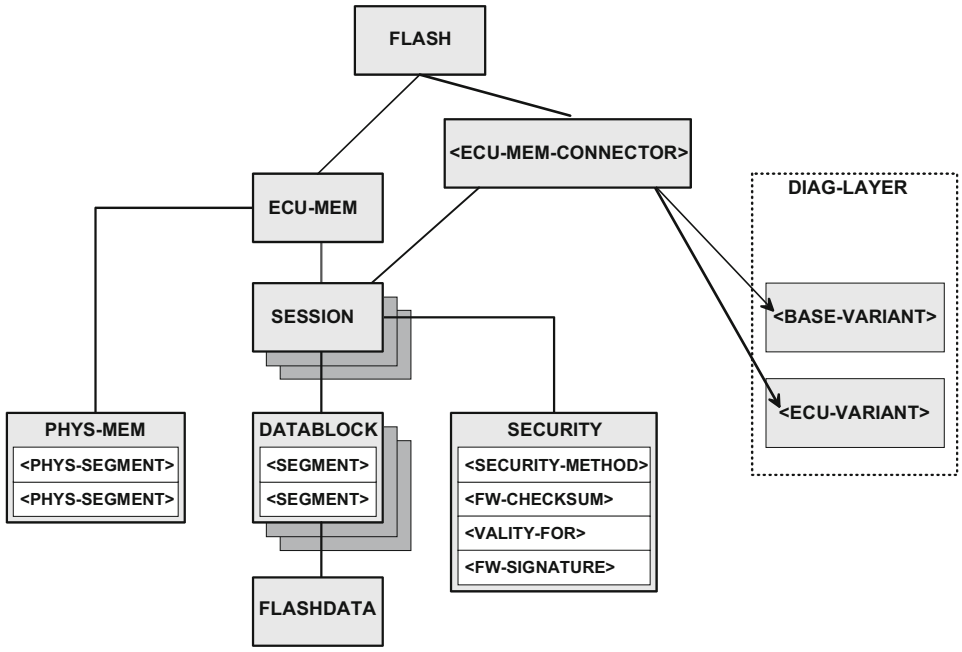

Abb. 6.40 ODX-Beschreibungsstruktur für die Flash-Programmierung (vereinfacht)

Tab. 6.25 ODX-Beschreibung von Speichersegmenten

```
<SEGMENTS>
   <SEGMENT ID="SEGMENT01">
         <SHORT-NAME>SEGMENT01</SHORT-NAME>
         . . .
         <SOURCE-START-ADDRESS>0000</SOURCE-START-ADDRESS>
         <SOURCE-END-ADDRESS>3FFF</SOURCE-END-ADDRESS>
   </SEGMENT>
   <SEGMENT ID="SEGMENT02">
         <SHORT-NAME>SEGMENT02</SHORT-NAME>
         . . .
         <SOURCE-START-ADDRESS>4000</SOURCE-START-ADDRESS>
         <SOURCE-END-ADDRESS>4FFF</SOURCE-END-ADDRESS>
   </SEGMENT>
   . . .
</SEGMENTS>
```

Die Flash-Programmierung wird durch mehrere Prüfmechanismen abgesichert. Um unbefugte Manipulationen zu verhindern, wird ein Zugriffsschutz implementiert (z. B. *Seed and Key*). Der Algorithmus für die Berechnung des *Keys* aus dem *Seed* muss nicht nur dem Steuergerät sondern auch dem Laufzeitsystem bekannt sein. Ebenso werden die Informationen zur Verifikation der Programmierung wie Prüfsummen und Signaturen benötigt. Diese Daten werden innerhalb von ODX in SECURITY-Elementen abgelegt und einer Session bzw. einem Datenblock zugewiesen.

Innerhalb von PHYS-MEM wird die real existierende physikalische Speichertopologie des Steuergerätes beschrieben. Ein Laufzeitsystem kann somit prüfen, ob die logischen Segmente der Flash-Daten mit der tatsächlich existierenden Speichertopologie identisch sind. Dazu wird der Flash-Speicher durch PHYS-SEGMENTe bestehend aus START-ADDRESS, END-ADDRESS und BLOCKSIZE modelliert.

Der eigentliche Programmiervorgang erfolgt durch die sequenzielle Abarbeitung von definierten Diagnosediensten (Programmiersequenz). Wie oben bereits erwähnt, beschreibt ODX eine solche Abfolge von Diagnosediensten in einem Job. Damit die Diagnosedienste des Flash-Jobs auf die Flash-Daten zugreifen können, muss eine Verbindung zwischen dem Flash-Job im Dokument DIAG-LAYER und einer SESSION im Dokument ECU-MEM hergestellt werden. Diese Verbindung wird durch den ECU-MEM-CONNECTOR vollzogen.

6.6.11 FUNCTION-DICTIONARY: Funktionsorientierte Diagnose

Die funktionsorientierte oder funktionsbezogene Diagnose stellt eine abstrakte Sichtweise der Diagnose dar. Dabei werden innerhalb eines Fahrzeugs nicht die einzelnen Steuergeräte betrachtet, sondern eine bestimmte Funktionalität im Gesamtsystem.

An der Umsetzung einer Funktion im Fahrzeug können durchaus mehrere Steuergeräte beteiligt sein. Die Funktionalität *Außenbeleuchtung* besteht beispielsweise aus einem

Steuergerät für die vorderen und einem für die hinteren Lampen. Die Schalterstellung am Armaturenbrett wird dabei von einem weiteren Steuergerät eingelesen. Für eine Funktion „LampenTest“, bei der die gesamte Außenbeleuchtung ein- und wieder ausgeschaltet wird, müssen also beide Lampensteuergeräte mit dem entsprechenden Diagnosedienst für das Ein- bzw. Ausschalten angesprochen werden. Ein Anwender muss daher exakte Kenntnisse haben, welche Steuergeräte bei der Umsetzung der Funktion involviert sind und welche Diagnosedienste die gewünschte Funktionalität erzeugen. Bei der funktionsorientierten Diagnose sind diese Informationen gekapselt. Der Anwender startet die Funktion „LampenTest“ und das Laufzeitsystem ermittelt über das ODX-Dokument FUNCTION-DICTIONARY die beteiligten Komponenten und die notwendigen Diagnosedienste für die Funktionsausführung. Speziell bei komplexen Systemen mit vielen beteiligten Komponenten (z. B. Fahrwerkregelung bzw. Assistenzsysteme wie ESP) bietet diese Sichtweise einen großen Vorteil. So werden bei einem System durch „Fehlerspeicher_Lesen“ alle Fehlerspeicher der beteiligten Komponenten ausgelesen. Durch entsprechende Analyse der Einträge kann so ein Fehlerbild auf Systemebene erstellt werden.

Die einzelnen Funktionen werden in FUNCTION-NODES organisiert (Abb. 6.41). Über Referenzen auf ausführbare (EXECUTABLE) Dienste (DIAG-COM) und MULTIPLE-ECU-JOBs Innerhalb des DIAG-OBJECT-CONNECTORs werden die Verbindungen zu den einzelnen Steuergeräten hergestellt.

6.6.12 Packaged ODX und ODX-Autorenwerkzeuge

ODX-Datensätze werden von verschiedenen Stellen bei Fahrzeugherstellern und Zuliefern erstellt und gepflegt, wobei jeder Datensatz aus einer ganzen Reihe von XML- und gegebenenfalls Binärdateien besteht. Außerdem „leben“ die Datensätze, d. h. sie werden laufend weiterentwickelt, so dass Änderungsstände dokumentiert und versioniert werden müssen.

ASAM hat zu diesem Zweck unter der Bezeichnung *Packaged ODX* (PDX) ein Archivformat definiert, mit dem verschiedene, zu einem Projekt gehörende ODX-Dateien zusammengefasst werden können. Im Wesentlichen handelt es sich dabei um das aus der Rechnerwelt bekannte ZIP-Format zur komprimierten Speicherung von Verzeichnis und Dateistrukturen ergänzt um den *PDX-Catalogue*, eine XML-Datei, die das Inhaltsverzeichnis der PDX-Datei enthält.

Die Erstellung von konsistenten ODX-Datensätzen für ein Fahrzeug ist eine komplexe Aufgabe, die ohne entsprechende Autorenwerkzeuge nicht sinnvoll bewältigt werden kann. Beispiele solcher Werkzeuge werden in Abschn. 9.6 dargestellt.

6.6.13 ODX Version 2.2 und ISO 22901

Die aktuellste ODX-Version, mittlerweile als ISO 22901 standardisiert, beschreibt insbesondere Anwendungsszenarien für die Flash-Programmierung und die Handhabung von

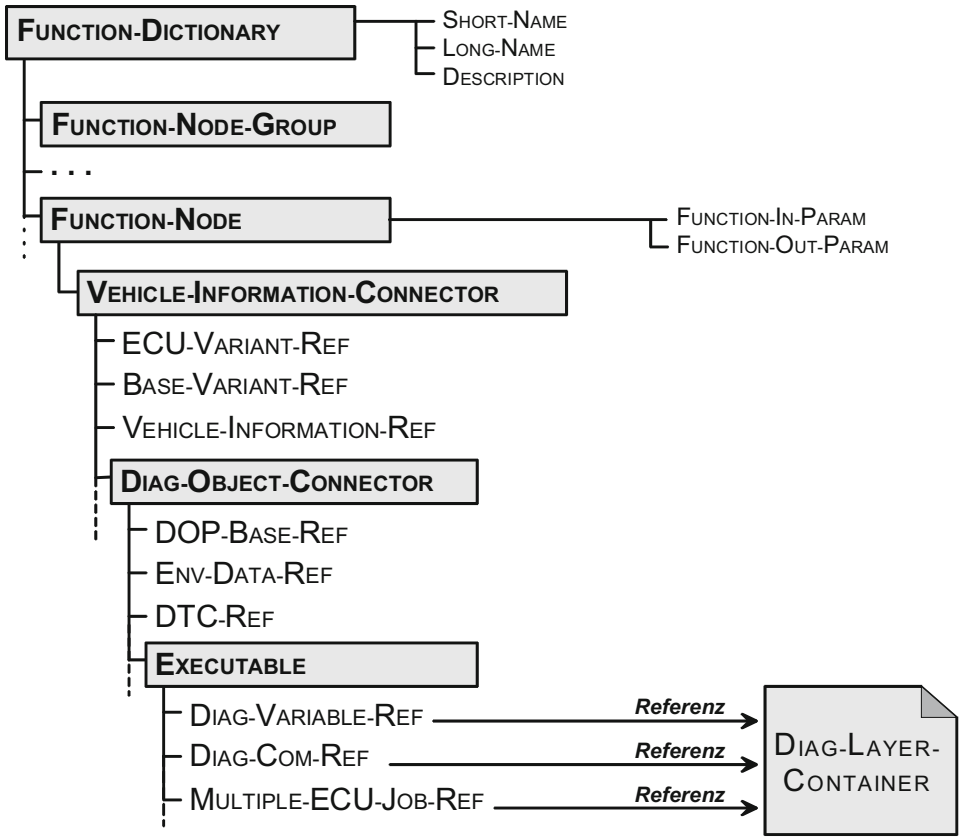

Abb. 6.41 Aufbau des FUNCTION-DICTIONARY-Dokumentes (vereinfacht)

Varianten ausführlicher als die Vorgänger. Zu den wenigen echten Neuerungen gehören LIBRARY-Elemente für ECU-JOBS, um Programmbibliotheken anzulegen und eine einfache Versions- und Zugriffsverwaltung für diese Bibliotheken.

6.7 ASAM AE MCD 3-Server

Die Grundstruktur eines MCD 3-Server wurde bereits in Abb. 6.22 dargestellt. Die Datenbasis des Servers ist in sogenannten Projekten organisiert, in denen beispielsweise die Daten eines bestimmten Fahrzeugmodells zusammengefasst werden.

Der MCD 3-Server kennt drei Grundzustände (Abb. 6.42). Zu Beginn wählt der Anwender über den entsprechenden Bedienclient eines der Projekte aus und kann dann zunächst die in der Datenbank für dieses Projekt gespeicherten Werte einsehen. Sobald die Verbindung zu einem Steuergerät hergestellt ist, können dann auch aktuelle Werte ausgelesen oder verändert werden.

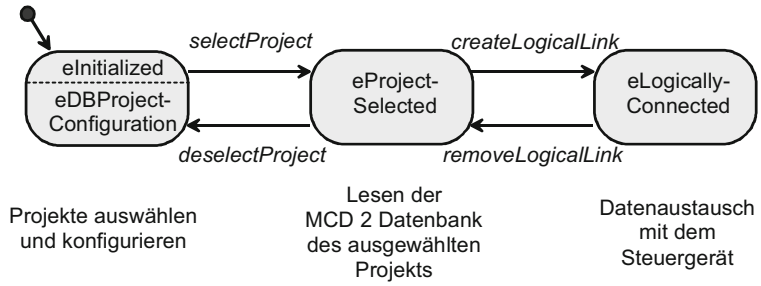

Abb. 6.42 Grundzustände des MCD 3-Servers (vereinfacht)

Wie Projekte innerhalb der Datenbank zu organisieren und über den Server zu verwalten sind, wird in der MCD 2/3-Spezifikation im Einzelnen nicht geregelt, sondern ist herstellerabhängig.

Das MCD 3-Objektmodell sieht eine strenge Trennung zwischen Datenobjekten (*Database Object DB*) und Laufzeitobjekten (*Runtime Object*) vor (Abb. 6.43). Die Datenobjekte repräsentieren den Inhalt der MCD 2-Datenbanken und sind statisch, d. h. sie können nicht verändert werden, während die Laufzeitobjekte die Interface-Methoden für den oder die Bedienclients bereitstellen. Die statischen Objekte bilden im Wesentlichen die Elemente der ODX-Diagnose- bzw. ASAP2-Applikationsdatensätze ab.

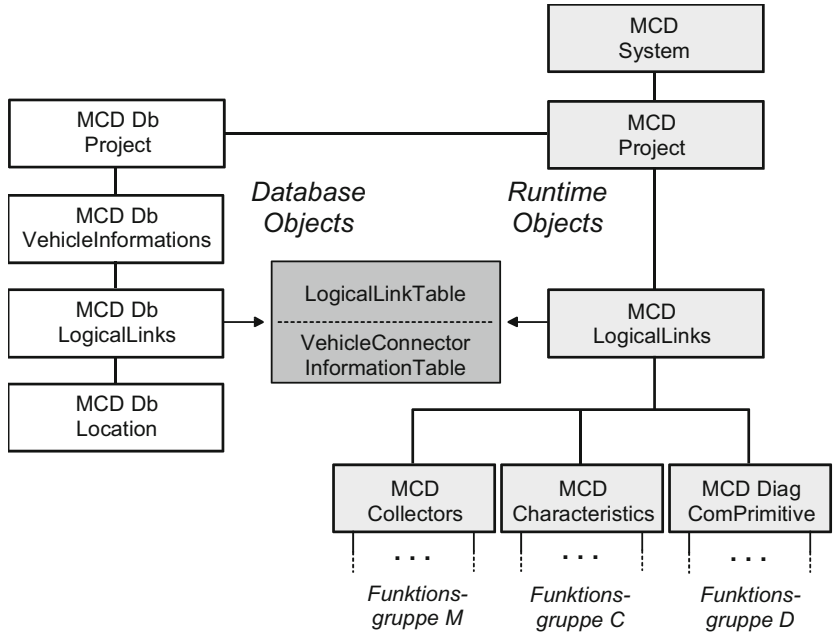

Abb. 6.43 Vereinfachte Grundstruktur des MCD 3-Objektmodells

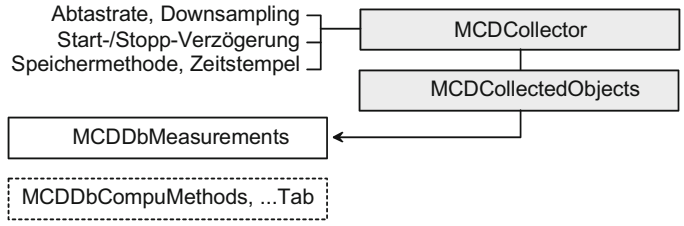

Abb. 6.44 Hauptelemente der Funktionsgruppe M – Messen

Viele der Objekte existieren mehrfach und werden in Listen (*Collections*) verwaltet, auf die über einen laufenden Index bzw. den Namen (ODX-Element SHORT-NAME) der Objektinstanz zugegriffen werden kann. Als Name einer solchen Liste wird üblicherweise der Name der enthaltenen Elemente im Plural verwendet, z. B. *MCDDbLogicalLinks* für die Liste aller im Projekt vorhandenen *MCDDbLogicalLink* Elemente aus der ODX-Datei (vgl. Abschn. 6.6.3).

Clients können über Schnittstellenmethoden der Laufzeitobjekte nicht nur selbst Werte abfragen und setzen sondern sich durch das Installieren von sogenannten Event-Handlern auch über auftretende Ereignisse, z. B. die Unterbrechung einer logischen Verbindung, automatisch vom Laufzeitsystem benachrichtigen lassen.

6.7.1 Funktionsgruppe M – Messen

Zentrales Objekt dieser Funktionsgruppe ist der *Collector* (Abb. 6.44). Seine Aufgabe besteht darin, selbstständig Messungen durchzuführen und Messwerte abzuspeichern. Alle innerhalb eines Collector-Objektes durchgeführten Messungen werden mit derselben Abtastrate durchgeführt, wobei jeder Messwert in einem Measurement-Abschnitt der ASAP2-Datei (vgl. Abschn. 6.5) definiert sein muss. Innerhalb eines Collectors können nur Messwerte über einen einzigen Logical Link, d. h. aus einem einzigen Steuergerät, erfasst werden, es ist aber möglich, mehrere Collector-Objekte gleichzeitig anzulegen. Die Messwerte können wahlweise mit einem Zeitstempel abgespeichert werden, so dass nachträglich eine zeitliche Korrelation mehrerer Messdatensätze möglich ist. Nachdem ein Collector-Objekt erzeugt und die gewünschten Messungen konfiguriert wurden, werden die Messungen aktiviert und starten bzw. stoppen nach Vorliegen der in der ASAP2-Datenbank definierten Triggerbedingungen sowie der über Collector-Attribute einstellbaren Verzögerungszeiten automatisch.

Die Messwerte werden in einem Ringpuffer mit einstellbarer Größe abgelegt und können vom Client durch Polling oder mit Unterstützung von Events fortlaufend ausgelesen werden. Alternativ kann das Laufzeitsystem die Messdaten auch selbstständig in einer Datei ablegen. Die Umrechnung zwischen geräteinternen und physikalischen Werten über-

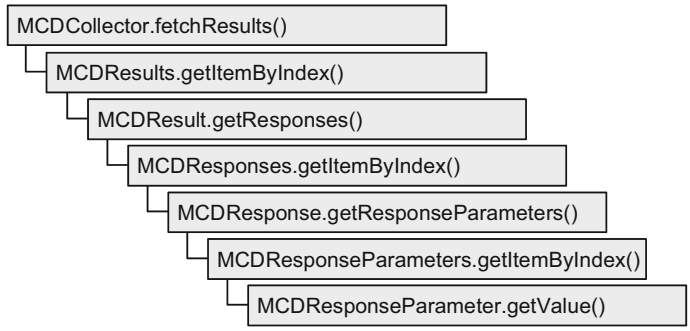

Abb. 6.45 Zugriff auf Messergebnisse über *MCDResult* und *MCDResponse*

nehmen die in der ASAP2-Datenbank als *CompuMethods* und *CompuTabs* definierten Umrechnungsformeln.

Optional können von einem ASAM MCD 3-Laufzeitsystem auch Kennlinien und Kennfelder (*Characteristics*) gemessen werden (im Abb. 6.44 nicht dargestellt).

Der Zugriff auf die Messergebnisse erfolgt über eine Kette von Funktionsaufrufen (Abb. 6.45). *MCDCollector::fetchResults()* liefert eine Liste der Ergebnisse aller Messungen des Collector-Objekts, aus denen dann das Ergebnis einer einzelnen Messung (*MCDResult*), daraus dann eine Liste der zum selben Abtastzeitpunkt erfassten Messwerte (*MCDResponse*) und damit schließlich ein einzelner Wert (*MCDResponseParameter*) extrahiert werden kann.

6.7.2 Funktionsgruppe C – Kalibrieren

Die Funktionsgruppe C erlaubt den Zugriff auf die steuergeräteinternen Parameter (Abb. 6.46). Diese sind in der ASAP2-Datenbank (vgl. Abschn. 6.5) als skalare Werte (*Scalar*), Kennlinien (*Curve*) und Kennfelder (*Map*) beschrieben.

Die unabhängigen Größen werden dabei als Achsen (*Axis*) bezeichnet, die abhängigen als Werte (*Value*). Bei Kennlinien und Kennfeldern bestehen beide aus mehreren Daten-

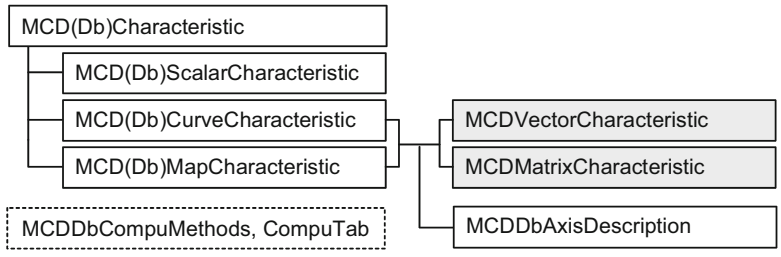

Abb. 6.46 Hauptelemente der Funktionsgruppe C – Kalibrieren

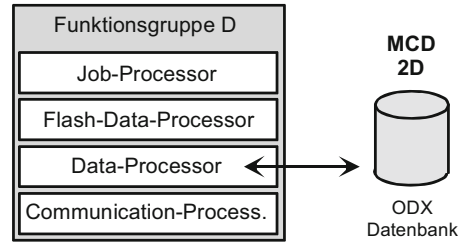

Abb. 6.47 Struktur der Funktionsgruppe D – Diagnose

werten, die zur einfacheren Handhabung zu Vektoren (*Vector*) oder Matrizen (*Matrix*) zusammengefasst werden. Für jeden in der Datenbank vordefinierten Wert, jede Kennlinie oder jedes Kennfeld kann ein Laufzeitobjekt angelegt werden. Dessen Eigenschaften können über *get…()*-Methoden abgefragt und dessen Werte über *read()* bzw. *write()*-Methoden gelesen und gesetzt werden. Die Umrechnung zwischen geräteinternen und physikalischen Werten übernehmen die in der ASAP2-Datenbank als *CompuMethods* und *CompuTabs* definierten Umrechnungsformeln.

6.7.3 Funktionsgruppe D – Diagnose

Da diese Funktionsgruppe wesentlich komplexer ist als die beiden anderen, gibt die MCD 3-Spezifikation nicht nur die äußere Schnittstelle sondern auch den inneren Aufbau präziser vor (Abb. 6.47). Der Diagnose-Block besteht aus sogenannten Prozessoren, die jeweils eindeutige Aufgaben besitzen:

- Der **Communication Processor** erzeugt Diagnosebotschaften, sendet diese zum Steuergerät und verarbeitet dessen Antworten.
- Der **Data Processor** bildet die einzige Schnittstelle zur ODX-Datenbank. Sämtliche Datenabfragen erfolgen über den Daten-Processor. Außerdem ist er für die Umwandlung der steuergeräteinternen Werte in die physikalischen Werte und umgekehrt zuständig.
- Der **Job Processor** führt vom Anwender bereitgestellte Funktionen aus, mit denen Sequenzen von Diagnosebotschaften erzeugt und deren Ergebnisse verarbeitet werden (vgl. Abschn. 6.6.7). Die Funktionen selbst werden als Java-Quellprogramme (*Java File*), als vorübersetzte Java-Bytecode-Programme (*Class File*) oder als Java-Archiv (*JAR File*) über die ODX-Datenbank bereitgestellt.
- Der **Flash Data Processor** bildet das Laufzeitsystem für die Abwicklung von Flash-Programmieraufgaben, wie bereits in Abschn. 6.6.10 beschrieben. Die Programmierdatensätze werden wiederum über die ODX-Datenbank bereitgestellt.

Hauptelement der Funktionsgruppe D ist *MCBDiagComPrimitive*, von dem, wie von den meisten anderen Objekten, sowohl ein Datenbank- als auch ein Laufzeitobjekt existiert (Abb. 6.48). Dahinter verbergen sich Diagnosedienste (*MCDDiagService*) und Jobs (*MCD-*

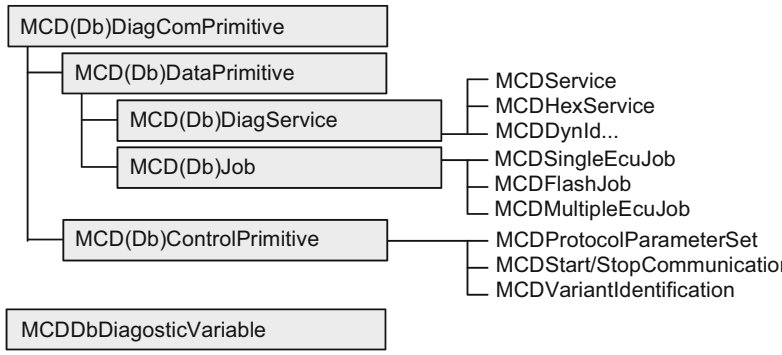

Abb. 6.48 Hauptelemente der Funktionsgruppe D

Job) sowie Schnittstellen zur Steuerung der Kommunikation zu den Steuergeräten (*MCDControlPrimitive*). Diagnosedienste können mit Hilfe der Methoden *Execute(A)Sync()* bzw. *StartRepetition()* einfach, mehrfach oder periodisch ausgeführt werden. Die einfache Ausführung erfolgt in der Regel synchron, d. h. nach dem Starten des Dienstes wartet der MCD 3-Server, bis der Diagnosedienst ausgeführt wurde und liefert das Ergebnis als Rückgabewert. Die mehrfache und die periodische Ausführung müssen asynchron erfolgen, d. h. der Server arbeitet nach dem Start des Dienstes sofort weiter und der Client wird mit Hilfe eines Events nach jeder Ausführung des Diagnosedienstes über das aktuelle Ergebnis informiert.

Auf die Parameter der Request Botschaft eines Diagnosedienstes kann über *MCDService::GetRequest()* und *MCDRequest::GetRequestParameters()* zugegriffen werden, so dass die in der ODX-Datenbank enthaltenen Werte verändert werden können. Das Auslesen der Ergebnisse erfolgt über ein *MCDResult*-Objekt wie bereits in Abb. 6.45 dargestellt, wobei statt des nur in der Funktionsgruppe M vorhandenen *MCDCollector*-Objekts das *MCDDiagService*-Objekt als Ausgangspunkt für die *fetchResults()*-Methode dient.

Alternativ zu Diagnosediensten kann auf einzelne Steuergerätegrößen auch direkt zugegriffen werden, sofern diese in der ODX-Beschreibung als Diagnosevariable (DIAG-VAR) definiert sind (vgl. Abschn. 6.6.5).

Jobs werden vom MCD 3-Server im Prinzip wie einfache synchrone Diagnosedienste aufgerufen, d. h. der Job wird gestartet und der MCD 3-Server wartet, bis der Job vollständig ausgeführt ist. Jobs müssen in der Programmiersprache Java geschrieben werden, wobei aber nur ein kleiner Teil der Java-Bibliotheken (Java.lang, Java.util, Java.math, Java.text) verwendet werden darf und der gesamte Ablauf innerhalb eines Jobs sequenziell erfolgen muss. Die Übergabe der Ein- und Ausgangsparameter erfolgt wie bei den Diagnosediensten über *MCDRequest* bzw. *MCDResult*-Strukturen. Hierfür und für das Zusammenspiel mit dem MCD 3-Laufzeitsystem stellt ASAM das Java-Package *asam.d.** zur Verfügung, das jeder Java-Job importieren muss. Komplexe Aufgaben können in mehrere Jobs zerlegt werden, wobei ein *äußerer* Job dann den Aufruf der *inneren* Jobs veranlasst. Falls das

Client-Programm ebenfalls in Java realisiert ist und die Verbindung zwischen Client und Server das Java-RMI-(Remote-Method-Invocation)-Verfahren einsetzt, ist es zu Testzwecken in der Entwicklungsphase möglich, einen Job nicht auf dem Server sondern im Client auszuführen.

Flash-Jobs sind aus Sicht des Laufzeitsystems gewöhnliche (Single-ECU)-Jobs, denen über *MCDDbFlashSession* und weitere *MCDDbFlash...*-Objekte zusätzlich die Informationen aus der ODX-Beschreibung ECU-MEM über den Aufbau des Steuergerätespeichers und des Flash-Programmierdatensatzes zur Verfügung steht.

6.8 MVCI-Schnittstelle für Diagnosetester nach ISO 22900

Ähnlich dem amerikanischen Ansatz der *Pass-Through Programmierung* nach SAE J1534 (siehe Abschn. 5.3.6) wird auch in Europa versucht, den Aufbau von Diagnosetestsystemen zu vereinheitlichen. Unter dem Begriff *Modular Vehicle Communication Interface* (MVCI) sind die ASAM-Diagnosekonzepte als ISO 22900 bzw. ISO 22901 standardisiert. Das Testsystem besteht aus dem Diagnosetestrechner und einem für verschiedene Bussysteme modular aufgebauten Businterface für die Diagnoseschnittstellen des Fahrzeugs (Abb. 6.49). Das Businterface übernimmt die Umsetzung der protokoll- und busunabhängigen *Diagnostic Protocol Data Unit* (D-PDU-API) Programmierschnittstelle auf das vom jeweiligen Fahrzeug genutzte Diagnoseprotokoll und Bussystem. Die eigentliche Diagnoseanwendung kann theoretisch direkt auf der D-PDU-Schnittstelle aufsetzen, soll aber vorzugsweise die darüber liegende *Diagnostic Server* (D-Server) Schicht verwenden, die gleichzeitig die Schnittstelle zu den ODX-Diagnosedaten kapselt. Die Normen für *ODX* (ISO 22901) und *D-Server* (ISO 22900-3) entsprechen im Wesentlichen den ASAM-Spezifikationen MCD 2D (siehe Abschn. 6.6) und MCD 3D (siehe Abschn. 6.7). Bei der *D-PDU API* (ISO 22900-2) dagegen handelt es sich um einen Neuentwurf, wobei die prinzipielle Aufgabenstellung der ASAM MCD 1 Schnittstelle entspricht (siehe Abschn. 6.1).

Die hardwareseitigen Vorgaben für das MVCI Businterface sind relativ vage. Das Businterface soll eines oder mehrere der derzeit gängigen oder zukünftigen Diagnoseprotokolle und Bussysteme unterstützen und über den bekannten OBD-Diagnosestecker nach ISO 15031-3 mit dem Fahrzeug verbunden werden. Als Verbindung zum Diagnosetester wird Ethernet bzw. USB vorgeschrieben, optional dürfen beliebige andere Schnittstellen, z. B. WLAN, unterstützt werden. Diese Vorgaben können relativ großzügig gehalten werden, da sämtliche Implementierungsunterschiede auf der Hardwareseite durch die *D-PDU API*, die der Hersteller zum Businterface mitliefern muss, verborgen bleiben. Die Schnittstelle ist so angelegt, dass mehrere MVCI Module und damit mehrere Protokolle und Bussysteme parallel unterstützt werden können.

Die *D-PDU API* unterstützt das gängige *Request-Response*-Kommunikationsmodell, bei dem der Diagnosetester eine Diagnoseanfrage sendet und auf eine Diagnoseantwort wartet (siehe Kap. 5). Die übergeordnete Anwendung fordert das Versenden einer Diagnosebot-

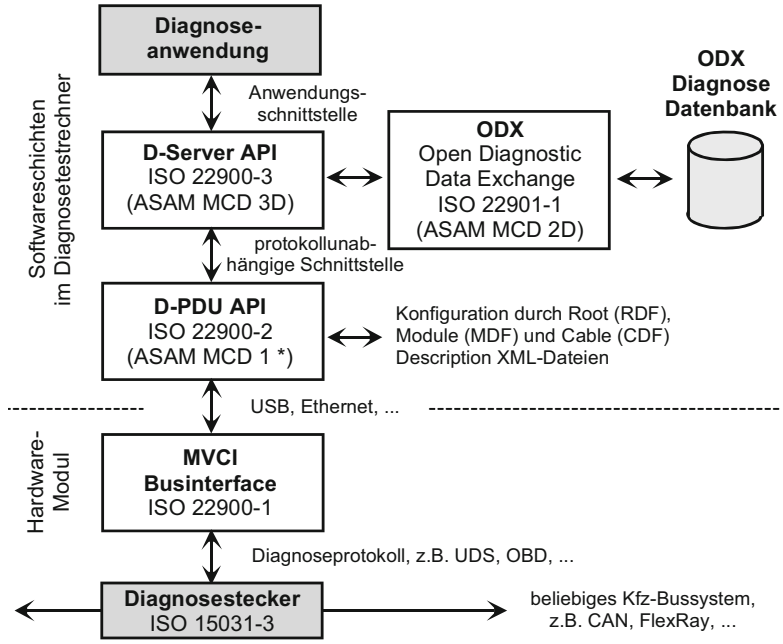

Abb. 6.49 Struktur eines MVCI-Diagnosetestsystems

schaft an (*Communication Primitive*) und wartet dann entweder in einer Abfrageschleife (*Polling*), bis die Diagnoseantwort empfangen wurde, oder lässt sich durch eine Rückruffunktion (*Callback Event Notification*) über die empfangene Antwortbotschaft informieren. Außerdem kann die *D-PDU* Schicht Empfangsbotschaften zwischenspeichern, um Kommunikationsszenarien behandeln zu können, bei denen mehrere Antwortbotschaften erwartet werden, z. B. wenn bei der funktionalen Adressierung bei KWP 2000 oder UDS mehrere Steuergeräte auf eine einzelne Anfrage antworten oder wenn ein Steuergerät spontan Botschaften an den Tester sendet wie beim UDS-Diagnosedienst *Response on Event*. Umgekehrt kann die *D-PDU* Schicht Botschaften, z. B. *Tester Present*, auch selbstständig mehrfach oder periodisch versenden.

Botschaften und sonstige Ereignisse werden beim Empfangen und Versenden im Businterface mit einem Zeitstempel versehen, so dass die darüber liegende Anwendung die zeitlichen Abläufe nachvollziehen kann.

Die *D-PDU API* benutzt das bereits von ODX bekannte Konzept der *logischen Verbindungen* (LOGICAL-LINK, siehe Abschn. 6.6.3) und der Definition von Protokollparametern (COMPARAM, siehe Abschn. 6.6.4). Beim Aufbau einer Verbindung (Tab. 6.26) werden die notwendigen Daten aus den MVCI-Konfigurationsdateien entnommen. Die zentrale Konfigurationsdatei, das *Module Description File MDF*, beschreibt, welche Bussysteme und Protokolle das Businterface unterstützt und legt Minimal-, Maximal- und Defaultwerte für die einzelnen Protokollparameter fest. Die Defaultwerte werden dann bei

Tab. 6.26 Vereinfachter Ablauf einer Verbindung mit *D-PDU API* Funktionen

Verbindungsaufbau	
`PDUConstruct()`	Laden der *D-PDU API* Bibliothek, Erkennen der angeschlossenen Businterface-Module
`PDURegisterEventCallback()`	Registrieren von Rückruffunktionen (optional)
`PDUCreateComLogicalLink()`	Initialisieren einer logischen Verbindung
`PDUGet/SetComParam()`	Abfragen und Setzen von Protokollparametern
`PDUConnect()`	Herstellen der Verbindung vom Tester zum Steuergerät
Senden und Empfangen von Daten mit Hilfe von ComPrimitives	
`PDUStartComPrimitive()`	Senden einer Diagnosebotschaft
`PDUGetEventItem()`	Lesen der Antwortbotschaft
Verbindungsabbau	
`PDUDisconnect()`	Beenden der Verbindung zum Steuergerät
`PDUDestroyComLogicalLink()`	Löschen der logischen Verbindung
`PDUDestruct()`	Entladen der *D-PDU API* Bibliothek

Bedarf durch diejenigen Parameter überschrieben, die für die spezielle Fahrzeugkonfiguration gültig sind und in der zugehörigen ODX COMPARAM-SPEC des Fahrzeugs festgelegt wurden.

Das Senden und Empfangen von Botschaften erfolgt mit Hilfe der sogenannten *Communication Primitives*. Dabei ergänzt die D-API beim Senden die von der übergeordneten Anwendung gelieferten „Nutzdaten“ (inkl. *Service Identifier*) der Diagnosebotschaften um die entsprechenden Protokollheader und liefert beim Empfangen nach Entfernen der Protokollinformationen die entsprechenden „Nutzdaten“ zurück. Eine Interpretation der Nutzdaten findet nicht statt, sondern erfolgt in der übergeordneten Anwendung z. B. mit Hilfe der ODX DIAG-SERVICE Definitionen (siehe Abschn. 6.6.5). Spezielle *Communication Primitives* führen protokollspezifische Abläufe wie etwa die KWP 2000 *Fast Initialisation* durch (siehe Abschn. 2.2.3).

Die *D-PDU API* übernimmt also im Normalfall die Abwicklung des Übertragungsprotokolls inklusive der Einhaltung der entsprechenden Zeitbedingungen. Im Sonderfall, dem sogenannten *Raw* oder *Pass-Through Mode*, dagegen wickelt die übergeordnete Anwendung das Diagnoseprotokoll selbst ab und verwendet die *D-PDU API* lediglich zum Versenden und Empfangen der Busbotschaften. Somit kann das Businterface auch für Protokolle genutzt werden, die nicht direkt durch MVCI unterstützt werden, z. B. XCP. Das Protokoll muss dann im Diagnosetestrechner selbst implementiert werden, was bei den üblichen, nicht echtzeitfähigen Betriebssystemen allerdings die Einhaltung der Timing-Bedingungen gegebenenfalls erheblich erschwert.

6.9 OTX-Beschreibung von Testabläufen nach ISO 13209

Diagnoseabläufe bestehen aus aufeinanderfolgenden Diagnosebotschaften, die üblicherweise von einem menschlichen Benutzer über die Bedienoberfläche des Diagnosetesters ausgelöst und deren Ergebnisse visualisiert werden. Nachfolgende Botschaftssequenzen hängen häufig von den Resultaten der vorigen Sequenz sowie weiteren Eingaben des Benutzers ab. ODX erlaubt es, derartige Abläufe über Java-Jobs zu realisieren. Ein Java-Job ist allerdings bereits die in eine spezielle Programmiersprache umgesetzte Implementierung eines derartigen Ablaufs. In der Praxis ist der Diagnoseentwickler allerdings ein Fachmann für Diagnoseprozesse, der solche Aufgaben auf einer abstrakteren Ebene spezifiziert und sich nicht mit programmiertechnischen Details befassen will und kann. Hierfür stellt ODX leider keine geeigneten Hilfsmittel bereit.

Um diese Lücke zu schließen, wurde mit dem *Open Test Sequence Exchange Format OTX* nach ISO 13209 ein Datenformat zur abstrakten Beschreibung von Diagnoseabläufen definiert. Logisch ist OTX im ASAM-Diagnosemodell nach Abb. 6.22 oberhalb der MCD 3 Schicht angeordnet.

Wie bei ODX soll der Anwender nicht direkt mit den XML-Datensätzen von OTX arbeiten, sondern seine Aufgaben mit Hilfe eines geeigneten Eingabewerkzeugs erledigen, welches das XML-Datenformat erstellt und daraus die notwendige Implementierung generiert. In diesem Kapitel werden die Grundstrukturen von OTX dargestellt. Ein typisches OTX-Werkzeug wird dann in Abschn. 9.7 vorgestellt.

6.9.1 Grundkonzepte und Aufbau von OTX

OTX unterscheidet das Grundsystem und die Systemerweiterungen (Abb. 6.50):

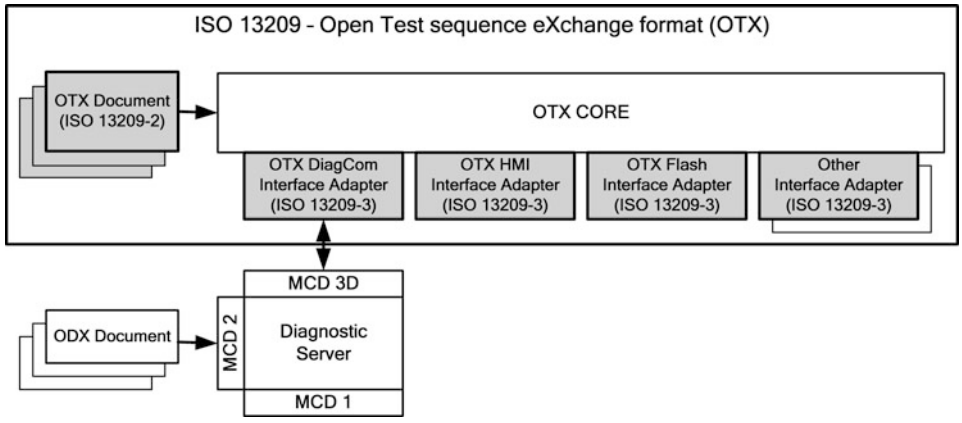

Abb. 6.50 Struktur eines OTX-Diagnosetestsystems

- Das OTX-Core-Datenmodell in ISO 13209-2 beschreibt die Grundelemente einer OTX-Testsequenz. Dabei werden die typischen Elemente einer Programmiersprache wie Datentypen, Deklarationen, Zuweisungen, Operatoren, Kontrollflussstrukturen und Methoden als XML-Strukturen definiert.
- Die in ISO 13209-3 beschriebenen OTX-Extensions definieren die Schnittstellen zur Außenwelt, z. B. den Zugriff auf Diagnosedienste und ODX-Daten (DiagCom), die Flash-Programmierung oder die Anbindung von Benutzerschnittstellen (HMI).

OTX beruht auf vier Grundkonzepten, die das Prozess- und Variantenmanagement bei der Diagnoseentwicklung vereinfachen sollen:

- *Specification and Realisation*
  Testsequenzen werden häufig in einem dreistufigen Prozess entwickelt. In der *Spezifikationsphase* werden die Testfälle allgemein spezifiziert. Testsequenzen können bereits logisch erfasst, in einzelne Schritte aufgeteilt und beschrieben werden, ohne die exakten Details zur Realisierung zu kennen. Bereits zu diesem Zeitpunkt ist ein Austausch der Testfallbeschreibungen zwischen Fahrzeughersteller, Zulieferern und Prüforganisationen wie TÜV oder DEKRA möglich.
  In der *Zwischenphase* wird die Spezifikation schrittweise implementiert. Dabei entstehen *Aktivitäten*, ein Ablaufcode im XML-Format, der Steuerkonstrukte klassischer Programmiersprachen abstrakt nachbildet. Ein Interpreter für OTX-Abläufe kann diese Sequenzen simulieren und mit einem Codegenerator in realen Programmcode für ein Testsystem übersetzt. In dieser Phase ist ein Testen im Mischbetrieb möglich. Bereits implementierte Anteile können real ausgeführt, noch nicht realisierte Teile in der OTX-Laufzeitumgebung (*OTX-Runtime*) simuliert werden.
  In der *Realisierungsphase* schließlich sind alle Testsequenzen vollständig implementiert und die Entwicklung ist abgeschlossen.
- *Context*
  Nicht alle für einen Test- und Prüfablauf relevanten Informationen können aus einem Steuergerät ausgelesen werden. Ein Steuergerät kann nicht erkennen, ob es sich im Entwicklungs-, Produktions- oder Serviceumfeld befindet. Um die Testsequenz im richtigen Kontext ausführen zu können, müssen solche Informationen durch das OTX-Laufzeitsystem durch sogenannte Kontextvariablen zur Verfügung gestellt werden. Kontextvariablen werden vom Diagnoseautor definiert und können Fahrzeugdaten, Benutzereingaben oder Informationen über den Diagnosetester enthalten.
- *Validity*
  OTX kann Testabläufe oder Teile davon in Abhängigkeit von bestimmten Bedingungen variabel ausführbar gestalten. Das *Validity*-Konzept ist vergleichbar mit den aus Programmiersprachen bekannten Compiler-Direktiven wie #ifdef, mit denen Programmteile ausgeblendet werden können. In Verbindung mit den Kontextvariablen können so Testschritte festgelegt werden, die nur in der Produktion oder nur in der Werkstatt ausgeführt werden. Das *Validity*-Konzept für ganze Testsequenzen ist übersichtlicher als

wenn solche Entscheidungen über viele einzelne Programmverzweigungen implementiert werden müssen.

- *Signature*
  Eine Signatur ist die Schnittstelle einer OTX-Testsequenz. Sie entspricht dem Funktionsprototyp einer Prozedur in einer gewöhnlichen Programmiersprache. Über den Signaturnamen und die Übergabe der entsprechenden Parameter kann eine Testsequenz aus einer anderen Testsequenz heraus aufgerufen werden, ohne dass Implementierungsdetails bekannt sind. In Verbindung mit *Validity*-Informationen lassen sich so generische Testsequenzen erzeugen, die sich zur Laufzeit an die entsprechende Umgebung anpassen.

6.9.2 OTX-Core-Datenmodell

Ein OTX-Datensatz wird in einem *OTX-Dokument* angelegt, das mit dem XML-Wurzelelement `<otx>` beginnt und in seinem Aufbau der Struktur eines Java-Programms ähnelt (Abb. 6.51). Am Anfang stehen Verwaltungsinformationen wie Name (`name`), Identifikationskennung (`id`), Versions- (`version`) und Zeitangaben (`timestamp`), eine Inhaltsbeschreibung (`specification`) und Historien- und Metainformationen (`adminData`). OTX unterstützt die Strukturierung, indem inhaltlich zusammengehörende OTX-Dokumente zu Paketen (`package`) gruppiert werden.

Der Import von Elementen aus anderen OTX-Dokumenten erfolgt über eine `imports`-Sektion. Die Kombination aus `package` und `name` stellt die eindeutige Bezeichnung der importierten OTX-Dokumente sicher und ermöglicht den Zugriff via eines *OTX-Links*. Die Definition eines kurzen `prefix`-Namens erleichtert den späteren Zugriff auf die importierten Elemente. Über diesen Mechanismus werden auch die Erweiterungen aus ISO 13209-3 eingebunden.

Innerhalb der XML-Struktur werden globale oder lokale Deklarationen von Variablen, Konstanten und Parameter in entsprechenden `declarations`-Sektionen angelegt. Auf der obersten Ebene werden globale Elemente wie Context-Variablen deklariert, die vom Laufzeitsystem gesetzt und innerhalb des OTX-Dokuments nur gelesen werden. Eine typische Context-Variable ist z. B. die Fahrzeugidentifikationsnummer VIN, die vom Laufzeitsystem über einen Diagnosedienst vom Fahrzeug abgefragt wird. Für die Realisierung lauffähiger Programme muss den Konstanten, Variablen oder Ausdrücken im `realisation`-Segment ein Datentyp wie `BOOLEAN`, `INTEGER`, `FLOAT`, `STRING` usw. und gegebenenfalls ein Initialisierungswert zugeordnet werden. Tabelle 6.27 zeigt ein Beispiel für eine Variablen-Deklaration. Die Sichtbarkeit von Variablen und Funktionen bestimmt OTX mit den Ebenen `PRIVATE`, `PACKAGE` und `PUBLIC`.

Nach den Variablendeklarationen folgen die bereits oben beschriebenen `validities` für die bedingte Ausführung von Testschritten. Unter `signatures` werden Funktionsprototypen und anschließend unter `procedures` die eigentlichen Funktionen definiert (Tab. 6.28).

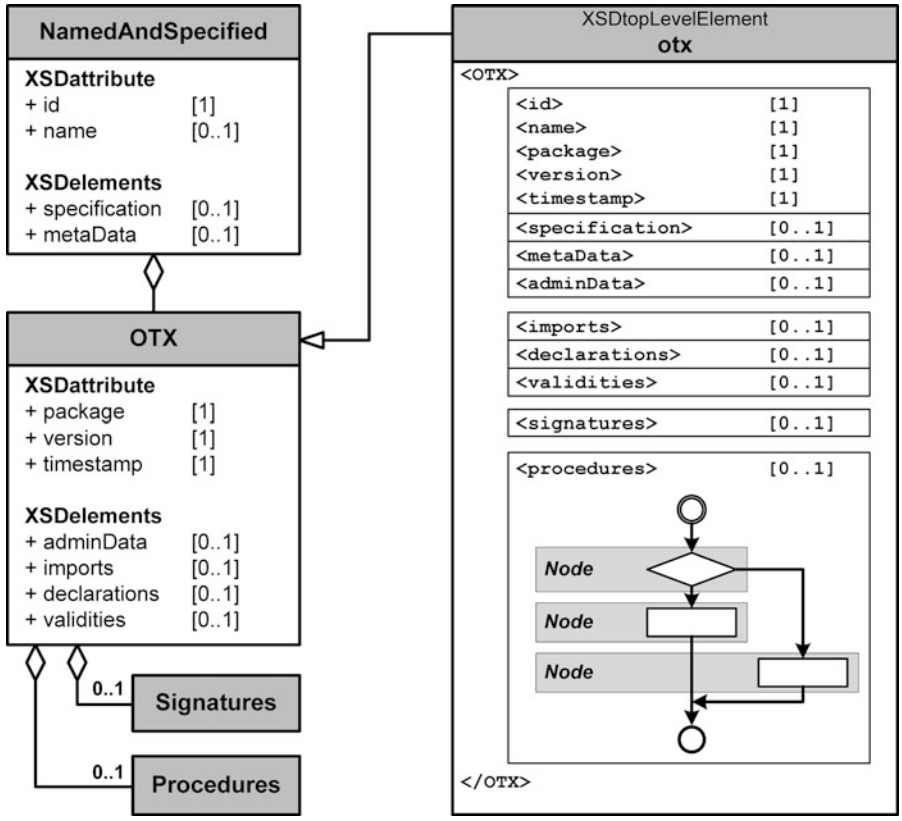

Abb. 6.51 Aufbau eines OTX-Dokumentes

6.9.3 OTX-Core-Programmelemente

Die von Programmiersprachen bekannten Elemente wie Zuweisungen, Blöcke und Kontrollflussstrukturen werden im OTX-Datenmodell als *Nodes* bezeichnet (Abb. 6.52). Eine Testprozedur besteht aus vielen *Nodes*, die nacheinander abgearbeitet werden. Jeder *Node* hat eine `id` zur Identifikation, optional sind ein Name (`name`) und eine Beschreibung (`specification`). Durch einen Schalter (`disabled`) kann ein *Node* deaktiviert werden. Dies entspricht dem *Auskommentieren* in einer klassischen Programmiersprache. OTX unterscheidet nach einfachen *AtomicNodes* und Elementen, die weitere *Nodes* enthalten (*CompoundNodes*).

Sogenannte *Action-Nodes* implementieren Zuweisungen und Funktionsaufrufe. Funktionen (OTX-Prozeduren) werden durch *ProcedureCall-Node*-Elemente mit den entsprechenden Argumenten aufgerufen. Tabelle 6.29 zeigt beispielhaft eine Zuweisung (`assignment`). Als Ergebnis (`result`) wird hier einer Variablen der Rückgabewert eines Ausdrucks (`term`) zugewiesen. Ausdrücke können mathematische Formeln, Lite-

Tab. 6.27 Vergleich einer Deklaration in einer Programmiersprache und in OTX

Deklaration in der Programmiersprache Java:

```
int i = 42; // A simple Integer
```

Deklaration in OTX:

```
<declarations>
  <variable>
    <name="i" id="001">
    <specification>A simple Integer</specification>
    <realisation>
      <dataType xsi:type="Integer"> <init value="42"/></dataType>
    </realisation>
  </variable>
</declarations>
```

rale, logische oder relationale Anweisungen, Konvertierungen, Dereferenzierungen und anderes enthalten.

Kontrollflusselemente in *Compound-Nodes* steuern den sequentiellen Ablauf von Programmen (Abb. 6.53). Das OTX-Core-Datenmodell bietet die Kontrollflusselemente Gruppierung (`group`) mit und ohne wechselseitigem Ausschluss (`mutex`), Schleifen (`loop`), Verzweigungen (`branch`), Parallelisierung (`parallel`) und Ausnahmebehandlung (`handler`) an. *End-Nodes* beenden einen Programmzweig, z. B. eine Schleife, vorzeitig und kehren auf eine höhere Ebene des Programmflusses zurück. Interessant ist dabei vor allem die Möglichkeit, mit *Parallel-Node* parallele Verarbeitungsstränge (`lanes`) zu erzeugen. Ein solcher Node wird erst verlassen, wenn alle parallelen Verarbeitungsstränge komplett durchlaufen sind. Ein einzelner Strang beendet sich normalerweise mit `return`, die gesamte Parallelstruktur kann mit `terminateLanes` vorzeitig abgebrochen werden. Kritische Bereiche (*critical section*) und wechselseitiger Ausschluss (*mutual exclusion*) können definiert werden, Ausnahmebehandlungen sind mit `throw` möglich.

Neben den vom OTX-Autor definierten Ausnahmebehandlungen (`UserException`) bietet OTX mit Bereichsüberwachungen bei Arrays und Listen (`OutOfBoundsException`), Typprüfungen (`TypeMismatchException`), der Erkennung von Fehlern in Formeln (`ArithmeticException`) oder der Überprüfungen auf nicht initialisierte Variable (`InvalidReferenceException`) schon standardmäßig eine große Zahl von Überwachungen, um Laufzeitfehler abzufangen.

6.9.4 OTX-Erweiterungen

Die OTX-Erweiterungen definieren die Schnittstellen des OTX-Kernsystems zur Außenwelt. Die wichtigsten Schnittstellen sind (Abb. 6.50):

Tab. 6.28 Attribute und Elemente von Prozeduren und Signaturen (Auswahl)

Attribut/Element	Prozedur, Signatur	Multiplizität	Beschreibung
`name`	Prozedur Signatur	1	Name der Prozedur oder Signatur innerhalb eines OTX-Dokumentes. `main` signalisiert die Einstiegsprozedur einer Testsequenz.
`visibility`	Prozedur Signatur	1	Definiert die Sichtbarkeit der Prozedur oder Signatur (`PRIVATE, PACKAGE, PUBLIC`).
`implements`	Prozedur	0..1	Verweis (*OTX-Link*) zwischen Signatur und Prozedur.
`validFor`	Prozedur	0..1	Definiert die Bedingungen, unter denen die Prozedur ausführbar ist. Die Ausführbarkeit kann in Abhängigkeit des Wertes einer globalen Konstanten (`constant`), einer `context`-Variablen oder eines `validity`-Wertes eingeschränkt werden.
`specification`	Prozedur Signatur	0..1	Für den Menschen lesbare Beschreibung der Prozedur oder Signatur.
`realisation`	Prozedur Signatur	0..1	Implementierungsblock
`parameters`	Prozedur Signatur	0..1	Deklarationsblock für Parameter von Prozeduren und Signaturen mit den Ein- und Ausgangsparametern `inParam`, `inOutParam` und `outParam`
`declarations`	Prozedur	0..1	Lokale Konstanten und Variablen
`flow`	Prozedur	1	Legt die einzelnen Programmelemente (Nodes) einer Prozedur fest.
…	…	…	…

Tab. 6.29 Beispiel einer Zuweisung in OTX

```
<action id="A-001">
  <specification>Example for an assignment action node</specification>
  <realisation xsi:type="Assignment">
    <result  xsi:type="IntegerVariable" name="i" />
    <term    xsi:type="IntegerLiteral" value="171174" />
  </realisation>
</action>
```

- *DiagDataBrowsing*, *DiagCom* und *DiagComRaw* erlauben den Zugriff auf ODX-Daten und die Abwicklung von Diagnosediensten (Abb. 6.54). Die Erweiterungen kapseln den Fahrzeugzugang, die physikalische oder funktionale Adressierung der Geräte, den Umgang mit Request- und Response-Parametern sowie die Variantenidentifikation. Grundsätzlich kann jede Art von Diagnose-Laufzeitsystem unterstützt werden, empfohlen jedoch wird ein standardisierter ODX/MVCI-Server.

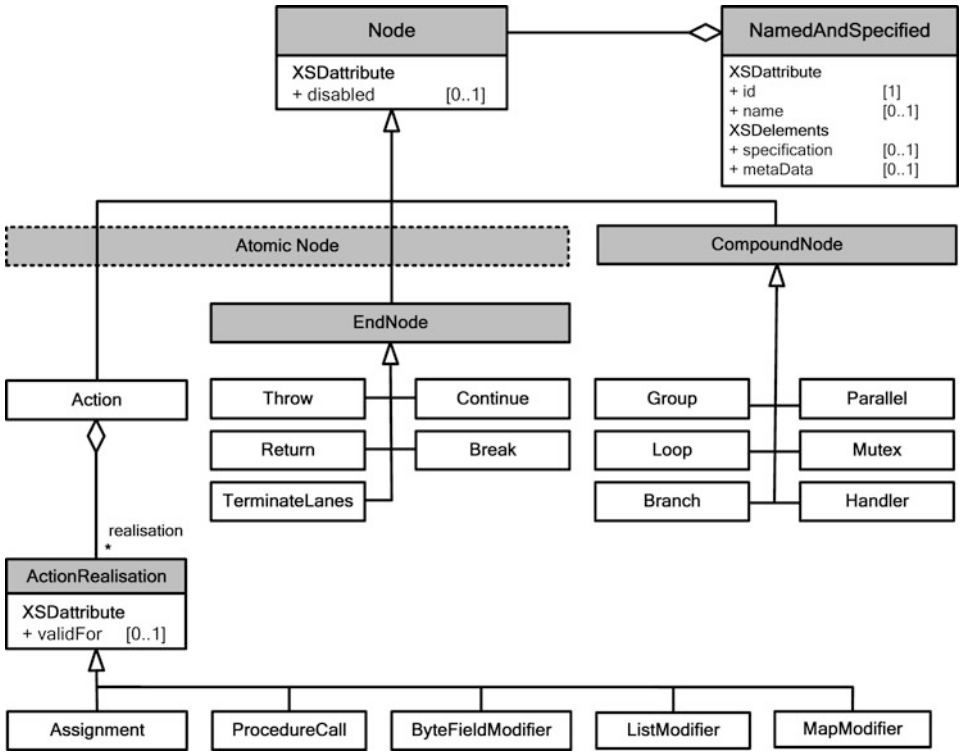

Abb. 6.52 Aufbau der Programmelemente (Nodes)

- *Human Machine Interface HMI* stellt Elemente zum Aufbau einer Bedienoberfläche für den Diagnosetester bereit. Die Erweiterung enthält Standarddialoge sowie frei parametrierbare Bildschirmausgaben und Tastatureingaben. OTX unterstützt im Grundumfang zwei Ausgabemasken. Der *BasicScreen* bietet modale Dialoge, mit denen der

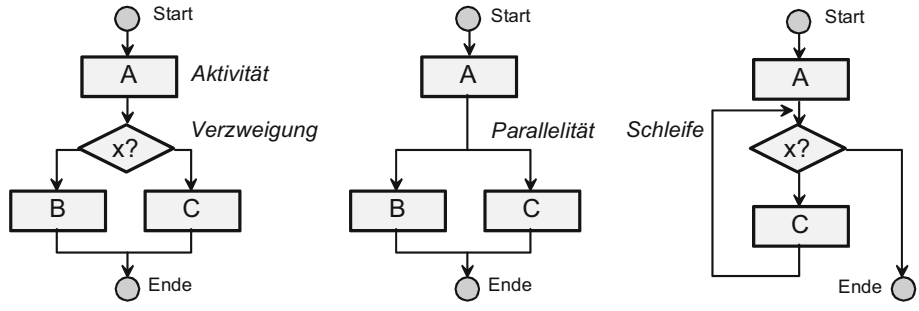

Abb. 6.53 Typische Kontrollflussstrukturen

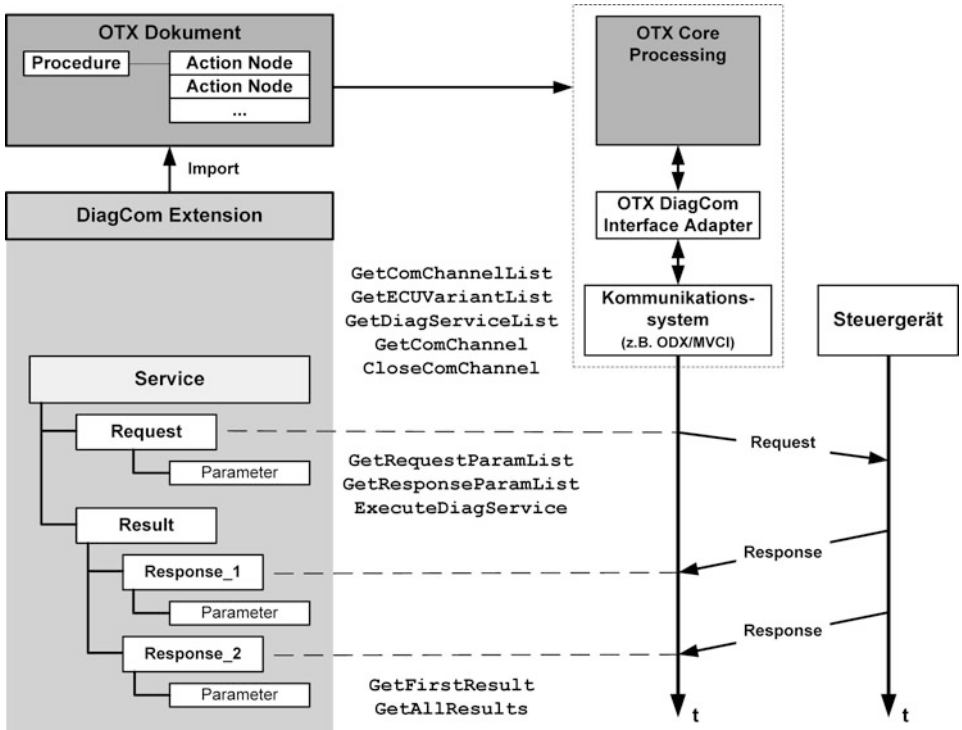

Abb. 6.54 Diagnose Request und Response in OTX

Anwender zu einfachen Eingaben aufgefordert werden kann bzw. Meldungen vom System erhält. Mit dem *CustomScreen* können beliebig komplexe Oberflächen gebaut werden. Da sich die verschiedenen Diagnosetester vom einfachen Handheld-Gerät über Geräte mit Touchscreens bis zum vollwertigen Computer mit hochauflösendem Farbbildschirm aber sehr stark unterscheiden, sieht OTX keine direkte Beschreibung des Bildschirmlayouts vor, sondern benötigt hierfür Tool-spezifische Erweiterungen wie beispielsweise das im Abschn. 9.7 beschriebene *Open Diagnostic Framework ODF*, dessen Daten standardkonform als OTX-Metadaten integriert werden.

- *Flash* enthält die Funktionen für die Flash-Programmierung von Steuergeräten einschließlich des Managements der Diagnosesitzungen und der Authentifizierung und Verifikation (s. h. Abschn. 9.4). Das OTX-Datenmodell (Abb. 6.55) ist an den ODX ECU-MEM Container (Abb. 6.40) angelehnt.
- *Measure* kapselt die Durchführung von Mess- und Steueraufgaben, die nicht ausschließlich über die Diagnosekommunikation sondern unter Einbindung externer Messgeräte durchgeführt werden.
- *EventHandling* ergänzt die sequenzielle Ablaufsteuerung des OTX-Kerns um Elemente zur Ereignisbehandlung, z. B. die Reaktion auf Benutzeraktionen wie Mausklicks, Tas-

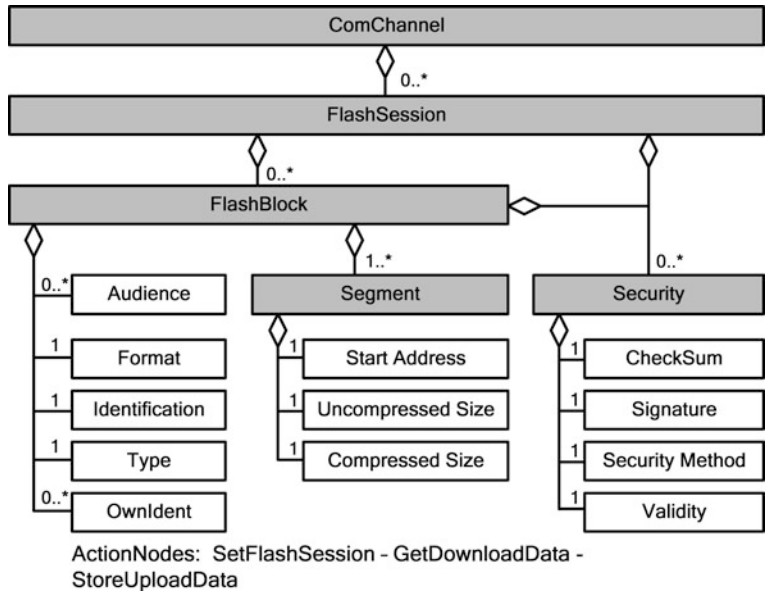

Abb. 6.55 OTX Flash Erweiterung

tatureingaben oder den Ablauf von Timern, aber auch die Veränderung von Variablen oder das Überschreiten eines Grenzwertes. Die Ereignisbehandlung erfolgt dabei synchron, d. h. die Ereignisquelle (*EventSource*) wird durch OTX definiert, danach wird auf das Auftreten des Ereignisses gewartet (*WaitForEvent*) und sequenziell weitergearbeitet, sobald das Ereignis (*Event*) auftritt.

- Mit *Internationalization*, *StringUtil*, *Math, DateTime* oder *Quantities*, die bei der Anpassung an unterschiedliche Landessprachen oder beim allgemeinen Umgang mit Texten, Umrechnungsformeln und Einheiten unterstützen, wird die praktische Implementierung der Diagnosefunktionen erleichtert. Darüber hinaus kann der Anwender oder Toolhersteller OTX um eigene Bibliotheken erweitern.

6.10 Normen und Standards zu Kapitel 6

ASAM 1	ASAP Standard CAN Calibration Protocol CCP Version 2.1, 1999, www.asam.net ASAP2 Meta Language für CCP – AML Version 2.6, 2002, www.asam.net ASAP Interface Specification Interface 1b Version 1.2, 1998, www.asam.net XCP The Universal Measurement and Calibration Protocol Family – Part 1: Overview Version 1.0, 2003, www.asam.net XCP The Universal Measurement and Calibration Protocol Family – Part 2: Protocol Layer Specification Version 1.0, 2003, www.asam.net XCP The Universal Measurement and Calibration Protocol Family – Part 3: XCP on CAN Transport Layer Specification Version 1.0, 2003, www.asam.net XCP The Universal Measurement and Calibration Protocol Family – Part 3: XCP on FlexRay Transport Layer Specification Version 1.0, 2006, www.asam.net XCP The Universal Measurement and Calibration Protocol Family – Part 3: XCP on USB Transport Layer Specification Version 1.0, 2004, www.asam.net XCP The Universal Measurement and Calibration Protocol Family – Part 3: XCP on Ethernet Transport Layer Specification Version 1.0, 2003, www.asam.net XCP The Universal Measurement and Calibration Protocol Family – Part 3: XCP on Sxl Transport Layer Specification Version 1.0, 2003, www.asam.net XCP The Universal Measurement and Calibration Protocol Family – Part 4: Interface Specification Version 1.0, 2004, www.asam.net XCP The Universal Measurement and Calibration Protocol Family – Part 5: Example Communication Sequences Version 1.0, 2003, www.asam.net XCP Version 1.1 – What's new, 2008, www.asam.net. Vollständige Spezifikationen der Version 1.1 derzeit nur für ASAM-Mitglieder zugänglich
ASAM 2	MCD 2 FIBEX Field Bus Exchange Format Version 2.0, 2006, www.asam.net MCD 2 NET FIBEX Field Bus Exchange Format Version 3.1.1, 2010 MCD 2 NET FIBEX Field Bus Exchange Format Version 4.1, 2013 MOST Fibex4MOST Cookbook Version 1.0, 2009, www.mostcooperation.com ASAM MCD 2MC/ASAP2 Interface Specification Interface 2 Version 1.6, 2009, www.asam.net ASAM MCD 2MC Calibration Data Format User's Guide Version 2.0, 2006, www.asam.net ASAM MCD 2MC Calibration Data Format Reference Guide Version 2.0, 2006, www.asam.net ASAM AE Meta Data Exchange Format for Software Sharing MDX Version 1.0, 2006, www.asam.net ASAM MCD 2D (ODX) Data Model Specification Version 2.0.1, 2005, www.asam.net ASAM MCD 2D (ODX) Data Model Specification Version 2.1.0, 2006 ASAM MCD 2D (ODX) Data Model Specification Version 2.2, 2008, www.asam.net ASAM MCD 2D (ODX) Authoring Guidelines Version 1.0, 2011, www.asam.net ODX ist mittlerweile als ISO 22901 standardisiert (siehe unten)

ASAM 3	ASAM MCD 3 Application programmer's interface specification Version 2.2.0, 2008, www.asam.net ASAM MCD 3 Programmer's reference guide Version 2.2.0, Teil 1: MCD, MD und D, Teil 2: MCD, MC, M und C, 2008, www.asam.net ASAM MCD 3 Application programming interface specification Version 3.0.0, Teil D und Teil MC, 2013, www.asam.net MCD3 ist mittlerweile als ISO 22900 standardisiert (siehe unten)
ASAM MDF	ASAM MDF Measurement Data Format – Programmer's guide Version 4.1.0, 2012, www.asam.net
ISO	ISO 22900-1 Road vehicles - Modular vehicle communication interface (MVCI) Part 1: Hardware design requirements. 2008, www.iso.org ISO 22900-2 Road vehicles - Modular vehicle communication interface (MVCI) Part 2: Diagnostic protocol data unit application programmer interface (D-PDU API). 2009, www.iso.org ISO 22900-3 Road vehicles - Modular vehicle communication interface (MVCI) Part 3: Diagnostic server application programmer interface (D-Server API). 2012, www.iso.org ISO 22901-1 Road vehicles - Open diagnostic data exchange (ODX) – Part 1: Data model specification. 2008, www.iso.org ISO 22901-2 Road vehicles - Open diagnostic data exchange (ODX) – Part 2: Emissions-related diagnostic data in ODX format. 2011, www.iso.org ISO 22901-3 Road vehicles - Open diagnostic data exchange (ODX) – Part 3: Fault symptom exchange description (FXD). Noch nicht veröffentlicht ISO 13209-1 Road vehicles – Open test sequence exhange format (OTX) – Part 1: General information and use cases. 2011, www.iso.org ISO 13209-2 Road vehicles – Open test sequence exhange format (OTX) – Part 2: Core data model specification and requirements. 2012, www.iso.org ISO 13209-3 Road vehicles – Open test sequence exhange format (OTX) – Part 3: Standard extensions and requirements. 2012, www.iso.org

Literatur

[1] C. Marscholik, P. Subke: Datenkommunikation im Automobil. VDE-Verlag, 2. Auflage, 2011