Der Betriebssystemstandard OSEK/VDX fürs Automobil

Die Initiative *Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug*, kurz: OSEK, wurde 1993 von den Firmen BMW, Bosch, Daimler-Benz, Opel, Siemens, Volkswagen und des Instituts für industrielle Informationstechnik der TH Karlsruhe ins Leben gerufen. Ziel der Initiative war es, einen bis dahin noch nicht vorhandenen plattformunabhängigen Standard für verteilte, elektronische Systeme im Automobil zu schaffen. Standardisierte Schnittstellen sollten die Entwicklungskosten senken, die Systemintegration erleichtern und die Systemqualität, insbesondere die der Software, verbessern.

Bereits 1994 vereinigte sich das OSEK-Konsortium mit der französischen VDX-Initiative, bestehend aus dem PSA-Konzern (Peugeot und Citroën) und Renault, die in Frankreich ähnliche Ziele verfolgte, und heißt seitdem OSEK/VDX. Heute ist OSEK/VDX weltweit eine feste Größe in der Automobilbranche.

Der OSEK-Standard setzt sich aus den Bestandteilen Echtzeitbetriebssystem (OSEK-OS), Netzwerkmanagement (OSEK-NM) und Kommunikationsschnittstelle (OSEK-COM) zusammen (*Abbildung 1*). Im folgenden werden die Bestandteile des OSEK-OS näher betrachtet.

OSEK Implementation Language

Das OSEK-OS, das seit 1997 standardisiert ist, ist ein statisches Echtzeitbetriebssystem. D.h., sämtliche benötigten Betriebsmittel (Tasks, Interrupts, zu verschickende Nachrichten,...) müssen vom Entwickler in der Entwurfsphase festgelegt werden und sind während der Laufzeit einer Anwendung nicht mehr veränderbar. Der Vorteil, der sich daraus ergibt, liegt neben der einfachen Portierbarkeit auf alternative Plattformen darin, daß das gesamte Betriebsystem sehr schlank gehalten werden kann. Auf diese Weise wird mit dem in einem zu entwickelndem *Automotive System* immer noch knappen Gut Speicherplatz sparsam umgegangen. Der Nachteil liegt in einem Mangel an Flexibilität im Hinblick auf die Erweiterungsfähigkeit.

Für die Festlegung der von der Anwendung benötigten Betriebsmittel und zur Konfiguration des Betriebssystems stellt OSEK die *OSEK Implementation Language* (OIL) bereit. Mit der OIL werden sämtliche Eigenschaften des OSEK-OS und der Applikation definiert, Applikationsmodi bestimmt, Tasks und deren Parameter wie preemptives oder nicht-preemptives Verhalten definiert, der Zugriff auf gemeinsam genutzte Ressourcen geregelt und vieles mehr. Eine plattformunabhängige Entwicklung, z.B. für OSEK-Derivate verschiedener Hersteller, und eine einfache Portierbarkeit auf verschiedene Hardwareplattformen werden so vereinfacht.

Eine OIL-Datei dient einem Codegenerator als Basis zur Generierung der dazugehörigen C-Sourcen. OIL-Dateien müssen nicht notwendigerweise von Hand erzeugt werden, sondern sie können auch mit Hilfe eines OIL-Builders – eines CASE-Tools – weitestgehend automatisch generiert werden.

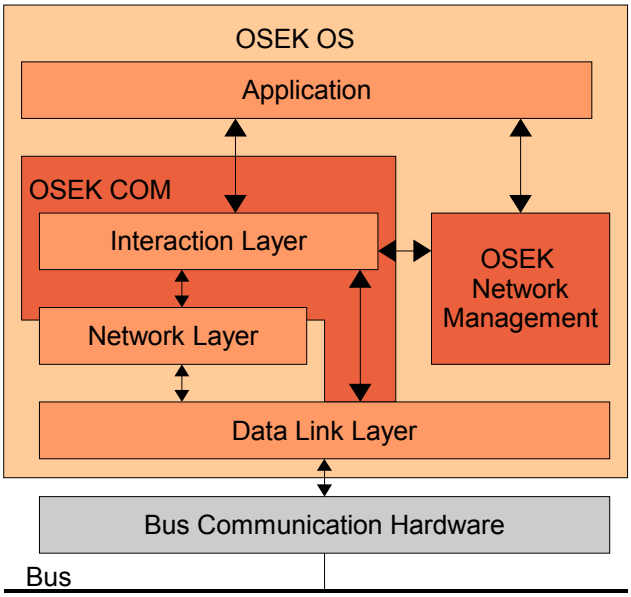

Abbildung 1: OSEK/VDX-Komponenten

Abbildung 2 zeigt den Workflow zur Entwicklung einer Applikation auf Basis von OSEK. Die aus der OIL-Datei generierten C-Files werden zusammen mit weiteren Sourcefiles und den Quelldateien des OSEK-OS kompiliert und mit den zur Anwendung gehörigen Libraries und Treiberdateien durch den Linker zur endgültigen Anwendung zusammengefügt. Diese wird dann etwa in das Flash-Memory eines Motorsteuergeräts geladen.

Tasks

Jede OSEK-Anwendung besteht aus mehreren Tasks, deren Instanzen im laufenden Betrieb um den Prozessor einer elektronischen Steuereinheit konkurrieren. Beispielsweise kann eine bestimmte Task in der Steuersoftware eines Elektronischen Stabilitätsprogramms (ESP) für die Erfassung und Weiterverarbeitung der Daten, die von

den Radsensoren übermittelt werden, zuständig sein, eine andere für die Auswertung der Informationen des Giersensors und des Lenkwinkelsensors und eine weitere Task für gezielte Bremseingriffe an einem Rad, um in einer Notsituation das Ausbrechen des Fahrzeugs zu verhindern. Jeder Task wird im OIL-File eine feste Priorität zugewiesen und kann verschiedene Zustände besitzen.

OSEK/VDX kennt zwei unterschiedliche Arten von Tasks: *Basic Tasks* und *Extended Tasks*. Der Unterschied zwischen beiden Arten liegt in ihren möglichen Zuständen. Sowohl eine *Basic Task* als auch eine *Extended Task* ist unmittelbar nach dem Systemstart im Zustand **inaktiv**, sie ist **bereit**, wenn sie auf Zuteilung des Prozessors wartet, und sie ist **laufend**, wenn sie im Besitz des Prozessors ist und ausgeführt wird. Eine in Ausführung befindliche *Extended Task* kann darüber hinaus in den Zustand **wartend** wechseln. Das geschieht auf ihren eigenen Wunsch immer dann, wenn sie auf das Eintreten eines externen Ereignisses warten muß, etwa ein Interrupt oder die Resultate der Berechnung einer anderen Task. Bis dahin gibt sie den Prozessor ab.

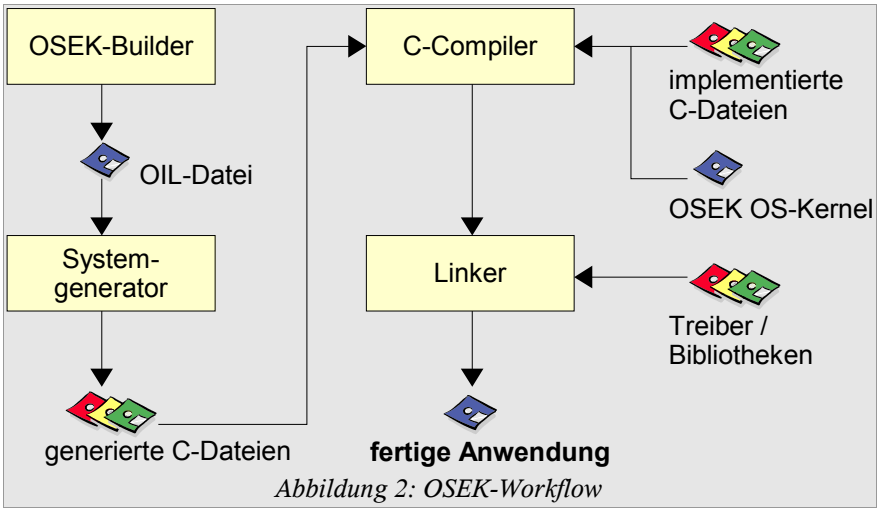

Abbildung 2: OSEK-Workflow

Ereignisse und Nachrichten

OSEK-Tasks (genauer: die *Extended Tasks*) können zur Synchronisation des Kontrollflusses Events an andere Tasks verschicken und von anderen empfangen. Für zyklische Aktivitäten kennt das OSEK-OS sogenannte Alarme.

Ein Alarm ist an einen Zähler (Counter) gebunden, der, wenn der Counter einen vorgegebenen Wert erreicht hat, den Alarm auslöst. In diesem Fall wird ein Event an eine bestimmte Task abgesetzt.

Tasks können Daten in Form von Nachrichten (Messages) an andere Tasks verschicken, die sich auch auf anderen Steuergeräten, z.B. angeschlossen am CAN-Bus, befinden dürfen. Der Versand erfolgt asynchron, d.h. der Absender wird nicht blockiert bis der Adressat die Nachricht erhalten hat. Nachrichten können gleichzeitig an einen oder mehrere Empfänger abgesetzt werden.

Interrupt Service Routinen

Eine *Interrupt Service Routine* (ISR) wird in OSEK durch einen Hardware-Interrupt getriggert. Eine ISR besitzt eine höhere Priorität als eine Task. D.h., während der Ausführung einer ISR werden die Tasks unterbrochen. Eine ISR sollte deshalb nicht zu umfangreich sein und nur wenige Operationen enthalten. Auch die ISRs sind in der Lage, Events an Tasks zu verschicken. Diese können dann statt dessen komplexere Aufgaben bewältigen. In OSEK können ISRs sowohl sporadisch als auch zyklisch angestoßen werden. Für die zyklischen ISR-Aufrufe sieht OSEK den Einsatz von *Timer-Interrupts* vor.

Hook-Funktionen

Das OSEK-OS unterstützt für eine einfachere Fehlersuche während der Software-Entwicklung Hilfsfunktionen (Hook-Funktionen). Dabei handelt es sich um spezielle Funktionen, die das Debugging, Tracing und Testen einer Anwendung unterstützen.

Eine Hook-Funktion wird in bestimmten (Ausnahme-)Situationen aufgerufen. So gibt es eine Hook-Funktion (ErrorHook), die immer dann aufgerufen wird, wenn eine Funktion des *Application Programming Interface* (API) nicht korrekt beendet wurde und einen Fehlercode zurückliefert. Eine andere (PreTaskHook) wird gestartet, wenn die Task vom Scheduler der Prozessor zugewiesen wird. Ebenso wird eine Hilfsfunktion (PostTaskHook) aufgerufen, sobald der Task der Prozessor wieder entzogen wird. Außerdem gibt es Hook-Funktionen (StartupHook und ShutdownHook), die beim Start bzw. Herunterfahren des Systems aufgerufen werden.

Ressourcen

Zum wechselseitigen Ausschluß verschiedener Tasks beim Zugriff auf gemeinsam genutzte Betriebsmittel wie Speicherbereiche (globale Variablen) oder Hardware-Komponenten, die nur einen exklusiven Zugriff gestatten, stellt das OSEK-OS entsprechende Synchronisationsmittel zur Verfügung: *Ressourcen*.

Um beim wechselseitigen Zugriff verschiedener Tasks, Verklemmungen (Deadlocks) und Prioritätsverschiebungen zu vermeiden, was bedeutet, daß eine Task überhaupt keinen Zugriff bekommt oder die Ressource durch eine Task niedrigerer Priorität blockiert wird, stellt OSEK mit dem *OSEK Priority Ceiling Protocol* eine entsprechende Zugriffsstrategie zur Verfügung.

OSEK Conformance Classes

Ein OSEK-OS muß nicht vollständig implementiert werden. Vielmehr reicht es aus, wenn einzelne, nur benötigte Features implementiert werden. Zu diesem Zweck gibt OSEK vier sog. *OSEK Conformance Classes* mit technischen Anforderungen vor:

- Klasse BCC_1: Es sind nur *Basic Tasks* erlaubt. Von jeder Task darf nur eine Instanz vorhanden sein. Jede Task muß eine andere Priorität besitzen.
- Klasse BCC_2: Es sind auch hier nur *Basic Tasks* erlaubt. Allerdings darf es auch mehrere Instanzen derselben Task geben. Verschiedene Tasks dürfen auch dieselbe Priorität besitzen.
- Klasse ECC_1: Es sind sowohl *Basic Tasks* als auch *Extended Tasks* zulässig. Von jeder Task ist nur eine Instanz erlaubt. Jede Task muß eine andere Priorität zugewiesen bekommen.
- Klasse ECC_2: Es gibt *Basic Tasks* und *Extended Tasks*. Für die *Basic Tasks* sind mehrere Instanzen pro Task zulässig, für *Extended Tasks* nur eine. Mehrere Tasks (das gilt für *Basic Tasks* und *Extended Tasks*) dürfen dieselbe Priorität haben.

Jede OSEK-Applikation muß die Eigenschaften einer *Conformance Class* erfüllen.

OSEKtime

Hochsicherheitskritische Anwendungen, wie die für die nahe Zukunft geplanten *X-by-Wire*-Systeme, bei denen hydraulische und mechanische Komponenten von Lenkung, Bremse und Antriebsstrang eines Automobils durch rein elektronische Systeme ersetzt werden sollen, müssen in ihrem Verhalten vorhersagbar und fehlertolerant sein. Ein ereignisgesteuertes System wie OSEK war bisher dafür nicht geeignet. Mit OSEKtime wurde deshalb eine zeitgesteuerte und damit deterministische Variante des Echtzeitbetriebssystems vorgestellt.

Kernstück ist eine fehlertolerante Kommunikationsschicht (FTCOM). Sie sorgt in einem verteilten System für eine fehlertolerante Kommunikation zwischen den Steuergeräten und stellt eine globale Zeitbasis – Voraussetzung für verteilte Anwendungen mit harten Echtzeitanforderungen – zur Verfügung an der sämtliche Aktivitäten der Tasks ausgerichtet werden.

Die Zukunft heißt AUTOSAR

Die Weiterentwicklung von OSEK erfolgt künftig unter dem Dach der AUTOSAR-Initiative, die im Jahre 2003 startete. Mit AUTOSAR wird bis 2006 eine offene Standardarchitektur für künftige elektronische Systeme im Automotive-Bereich geschaffen.

Die Hauptziele, die mit dem Projekt verfolgt werden sind: Die Entwicklung eines industrieweiten „Standard Core“ mit Basisfunktionen zum Bau künftiger Anwendungen im Baukastenprinzip; Skalierbarkeit für verschiedene Fahrzeugtypen und Plattformvarianten; einfache Integrationsmöglichkeit elektronischer Komponenten, die von verschiedenen Zulieferern kommen; Betrachtungen bezüglich Verfügbarkeit, Sicherheit und Redundanz der Systeme; einfachere Wartbarkeit und Möglichkeiten zum Einspielen von Software-Updates und -Upgrates während des gesamten Lebenszyklus eines Automobils, u.v.a. (bar)

Weitere Informationen

Bei weiteren Fragen stehen wir Ihnen gerne zur Verfügung.

Ingenieurbüro Barheine
Albstraße 47
76275 Ettlingen

Tel.: 0 72 43 / 52 37-67
Fax.: 0 72 43 / 52 37-68

E-Mail: kontakt@barheine.de

Web: http://www.barheine.de