# OSEKtime – Eine Softwareplattform für sicherheitsrelevante verteilte Applikationen im Automobil

**Thomas M. Galla**
**DECOMSYS GmbH**
**Stumpergasse 48/28**
**A-1060 Wien**
**Tel: +43 (1) 59983 15**
**FAX: +43 (1) 59983 715**
**galla@decomsys.com**

**Jochen Olig**
**3SOFT GmbH**
**Frauenweiherstrasse 14**
**D-91058 Erlangen**
**Tel: +49 (9131) 7701 120**
**FAX: +49 (9131) 7701 333**
**Jochen.Olig@3SOFT.de**

## Einleitung

In der Automobilindustrie herrscht der Trend, die Fahrsicherheit durch Applikationen zu erhöhen, die den Fahrer von Routineaufgaben entlasten. Diese Assistenzsysteme benötigen unmittelbare elektronische Kontrolle über die ihnen zugeordneten Aktuatoren. Bei diesen sogenannten X-by-Wire Applikationen werden mechanische und hydraulische Kopplungen durch fehlertolerante elektronische Systeme ersetzt. Typische Vertreter dieser Gruppe sind beispielsweise die vollelektronische Bremse (Brake-by-Wire) und Lenkung (Drive-by-Wire). Diese sicherheitskritischen Anwendungen verlangen nach zuverlässigen und sicheren Methoden der Realisierung, da das Versagen einzelner Komponenten bei diesen Anwendungen bekanntermaßen fatalste Folgen nach sich ziehen kann.

Seit knapp 10 Jahren setzt die Automobilindustrie mit dem OSEK/VDX Komitee ihre Bemühungen um, einheitliche Standards einer Software-Infrastruktur für die Fahrzeugelektronik zu schaffen. Diesen Anstrengungen liegt die Erkenntnis zu Grunde, dass der steigende Grad an funktionaler Komplexität nur über definierte Schnittstellen zu bewältigen ist. Die rasanten Innovationszyklen der Branche, besonders im Elektronikbereich, bringen Anforderungen hinsichtlich Kompatibilität und Wiederverwendbarkeit mit sich, die nur mit Schaffung von Standards auf den verschiedenen Systemebenen erfüllbar sind. Damit wird auch die Basis zur Erstellung fehlerfreier Software gelegt.

Der vom Komitee geschaffene Standard OSEK/VDX wird heute von den führenden Automobilherstellern flächendeckend eingesetzt. Dabei hat sich gezeigt, dass diese Systeme für die typischen Anforderungen des Komfortbus und des Antriebsstranges gut geeignet sind, in denen weitgehend autonome Steuergeräte lose Verbunde bilden. Für X-by-Wire Anwendungen hingegen ist ein koordiniertes Handeln in hoher zeitlicher Synchronität erforderlich. Dies machte die Schaffung eines neuen Standards nötig, da OSEK/VDX keine globale Zeit für alle Steuergeräte zur Verfügung stellt und somit Aktionen verschiedener Steuergeräte zu identischen Zeitpunkten nicht sichergestellt sind. Mit OSEKtime, seit Juli 2001 in der Version 1.0 verfügbar, liegt die Spezifikation eines zeitgesteuerten Betriebssystems inklusive zugehöriger Kommunikationsschicht vor, das den gestiegenen Anforderungen hinsichtlich Echtzeitfähigkeit und Fehlertoleranz genügt.

Im Wesentlichen besteht der neue Standard aus zwei Teilen: Zum einen das eigentliche OS, welches Betriebsmittel wie Tasks und Interrupt Service Routinen bereit stellt und deren zeitliche Überwachung übernimmt und zum anderen die Kommunikationsschicht FTCom. Letztere ist für die fehlertolerante Kommunikation zwischen den einzelnen Steuergeräten verantwortlich und gibt außerdem eine globale Zeit für alle beteiligten Knoten vor. Damit wird offensichtlich, dass erst durch das Zusammenwirken beider Teile die volle Funktionalität gegeben ist, wie sie von den oben beschriebenen Systemen gefordert wird.

# OSEKtime OS

OSEKtime OS ist als zeitgesteuertes Single-Chip Betriebssystem für verteilte Embedded Systeme konzipiert. Es unterscheidet zwei logische Einheiten, in denen Funktionen ausgeführt werden können. Interrupt Service Routinen (ISR) dienen der Ausführung Interrupt spezifischer Dienste. Tasks werden zu definierten Zeitpunkten gestartet und können jeweils einen von drei verschiedenen Zuständen haben: Im Zustand *running* ist eine Task der CPU zugeteilt und ihre Anweisungen werden ausgeführt. Nur jeweils eine Task kann zu jedem Zeitpunkt diesen Zustand einnehmen, während die beiden anderen Zustände gleichzeitig von mehreren Tasks angenommen werden können. Den Zustand *preempted* erreicht eine Task, die gerade im Zustand *running* ist, wenn sie von einer anderen Task, deren Aktivierungszeitpunkt erreicht ist, unterbrochen wird. Den Zustand *preempted* kann eine Task nur dann wieder verlassen, wenn die Task, die sie unterbrochen hat, in den Zustand *suspended* wechselt. *Suspended* heißt eine inaktive Task, die aktiviert werden kann.

Die Aktivierungszeitpunkte der Tasks werden vor Übersetzungszeit in einer so genannten Dispatchertabelle hinterlegt. Der Dispatcher als zentraler Bestandteil von OSEKtime OS aktiviert die einzelnen Tasks entsprechend der Aktivierungszeitpunkte. Ein vollständiger Durchlauf der Dispatchertabelle wird Dispatcherrunde genannt, wobei die Abarbeitung der Tabelle zyklisch erfolgt. Sollte eine zeitgesteuerte Task noch im Zustand *running* sein, wenn der Aktivierungszeitpunkt einer anderen erreicht ist, so geht die erste Task in den Zustand *preempted* über und bleibt solange in diesem Zustand, bis die sie unterbrechende Task beendet ist. Diese Art von Scheduling bezeichnet man als stack-basiertes Scheduling, wobei den Tasks keine Priorität zugeteilt ist. Einzig die Aktivierungszeitpunkte bestimmen die Vorrangsbeziehungen der Tasks untereinander, je später eine Task gestartet wird, desto weniger Tasks können diese in ihrer Ausführung verzögern. Für jede zeitgesteuerte Task kann über einen Eintrag in der Dispatchertabelle eine Deadline definiert werden. Das Überwachen der Deadline wird vom Dispatcher erledigt. Ist die betreffende Task zum entsprechenden Zeitpunkt noch nicht beendet, so wird das OS heruntergefahren. Wird für eine Task keine Deadline spezifiziert, so muss diese spätestens zum Ende der Dispatcherrunde beendet sein.

Das Einschalten eines Interrupts erfolgt ebenfalls über einen Eintrag in der Dispatchertabelle. Nach der Ausführung der ISR wird der zugehörige Interrupt vom OS wieder abgeschaltet und muss neu eingeschaltet werden.

Je nach Design können sich für eine Dispatcherrunde mehr oder weniger lange Zeiträume ergeben, in denen keine zeitgesteuerte Task oder ISR aktiv ist. In diesen Leerlaufzeiten ist eine vom OS zur Verfügung gestellte *ttIdleTask* der CPU zugeordnet, die nicht in der Dispatchertabelle eingetragen ist und für die keine Deadline definiert ist. In dieser Task können nicht-zeitkritische Aufgaben abgearbeitet werden.

Der Dispatcher wird typischerweise über einen nicht maskierbaren Interrupt getriggert, dessen Frequenz an die lokale Zeitbasis gekoppelt ist. Am Ende einer Dispatcherrunde, nachdem der letzte zeitgesteuerte Task beendet ist, erfolgt die Synchronisation der lokalen mit der globalen Zeit. Dies geschieht durch ein passendes Verlängern bzw. Verkürzen der aktuellen Dispatcherrunde.

Abbildung 1 zeigt die Funktionsweise des Dispatchers und das Prinzip des stack-basierten Scheduling.

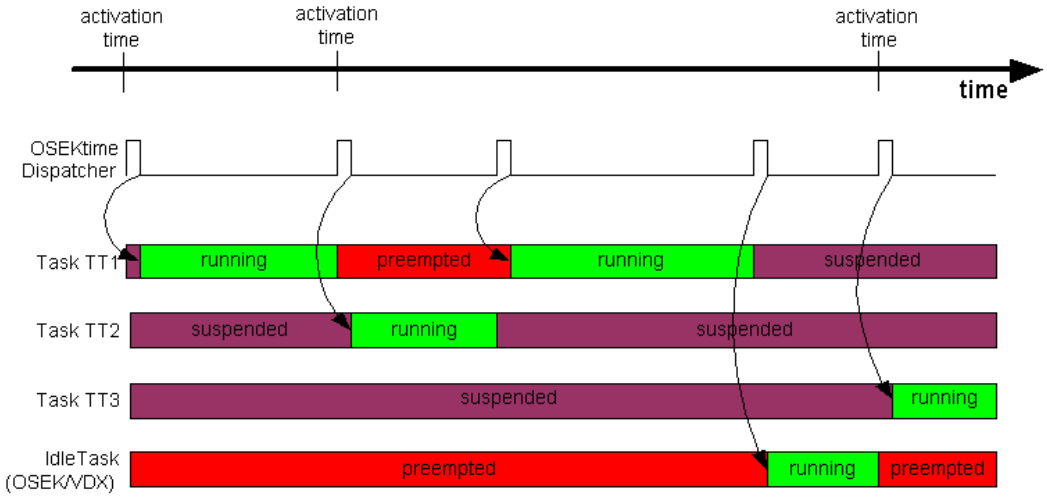


**Abbildung 1: Funktionsweise des OSEKtime Dispatchers**

In Abbildung 1 wird am Anfang der Dispatcherrunde zuerst der OSEKtime Dispatcher aktiv, welcher dann die Task TT1 aktiviert. Beim nächsten Aktivierungszeitpunkt unterbricht der Dispatcher die Task TT1, um die Task TT2 zu aktivieren. Die Task TT1 wechselt also in den Zustand preempted. Nach Beendigung der Task TT2 wird wiederum der Dispatcher aktiv und setzt die unterbrochenen Task (Task TT1) fort. Nach deren Beendigung ist keine weitere zeitgesteuerte Task abzuarbeiten, weshalb der Dispatcher die *ttIdleTask* ausführt. Sobald wieder eine zeitgesteuerte Task abgearbeitet werden muß (nächster Aktivierungszeitpunkt), wird die *ttIdleTask* unterbrochen und die zeitgesteuerte Task (Task TT3) aktiviert.

Nachdem sich die Aufgabe des Dispatchers zur Laufzeit auf das zyklische Abarbeiten einer Tabelle beschränkt, muß dem Konfigurationsprozess bei einem verteilten System mit stark synchronisierter Aufgabenteilung ein nicht unerhebliches Maß an Aufwand gewidmet werden. So ist beispielsweise die genaue Kenntnis der maximalen Abarbeitungszeit („Worst Case Execution Time“ (WCET)) sämtlicher Tasks und ISRs unerlässlich, um den Ablauf einer Dispatcherrunde zu planen. Die Dispatchertabelle selbst wird von einem Tool (siehe „Toolkette“) generiert.

## *OSEK/VDX Subsystem*

Wenn eine Applikation aus einem zeitkritischen Anteil und einem komplexeren, nicht-zeitkritischen Anteil besteht, so bietet OSEKtime eine weitere interessante Möglichkeit der Realisierung: An Stelle der erwähnten *ttIdleTask* kann ein OSEK/VDX Subsystem integriert werden, welches dem Anwender eine komfortable Palette von Betriebsmitteln zur Verfügung stellt. Zusammengefasst wären hier zu nennen:

### Tasks

Unter OSEK/VDX gibt es zwei Typen von Tasks, Basic und Extended Tasks. Beide kennen die Zustände *ready*, *running* und *suspended*. Extended Tasks können zusätzlich noch den Zustand *waiting* einnehmen, um auf ein Event zu warten. Im Gegensatz zu zeitgesteuerten Tasks haben OSEK/VDX Tasks Prioritäten, und zwar von 0 (niedrig) aufsteigend. Weiters unterstützen OSEK/VDX Tasks verschiedene Scheduling Strategien: Eine full-preemptive Task kann von einer anderen Task, die im Zustand *ready* und höher priorisiert ist, in ihrer Ausführung unterbrochen und in den Zustand *ready* gebracht werden. Dagegen gibt eine non-preemptive Task während ihrer Ausführung die CPU nicht frei, wenn eine höher priorisierte Task in den Zustand *ready* kommt.

### Interrupt Service Routinen

OSEK/VDX stellt zwei Kategorien von ISRs zur Verfügung: Kategorie 1 ISRs verwenden keine OSEK/VDX Betriebsmittel. Nach Beendigung einer solchen Routine wird die normale Programmausführung an der Stelle fortgesetzt, an der der zugehörige Interrupt aufgetreten ist. In ISRs der Kategorie 2 können OSEK/VDX Funktionen aufgerufen werden, die ein Rescheduling zur Folge haben. Dieses findet nach Beendigung der ISR statt, wenn die unterbrochene Task full-preemptive ist und kein anderer Interrupt aktiv ist.

### Events

Events dienen der Synchronisation zwischen OSEK/VDX Tasks und ISRs. Extended Tasks können auf ein oder mehrere Events warten, während beliebige Tasks und ISRs der Kategorie 2 sowie einige andere Mechanismen Events auslösen können. Somit lässt sich eine Ereignissteuerung aufbauen, die es Extended Tasks erlaubt, auf bestimmte Ereignisse zu warten, ohne dabei CPU-Zeit zu beanspruchen.

### Counter und Alarme

Counter dienen zum Zählen beliebiger Ereignisse und können beispielsweise mit einem zyklisch auftretenden Timer-Interrupt verknüpft werden. Ein Alarm stellt eine Aktion dar, die mit dem Erreichen eines bestimmten Counterwerts ausgeführt werden soll. Dies kann die Aktivierung einer OSEK/VDX Task oder das Setzen eines Events sein.

### Resources

Während eine Task eine Resource belegt, kann sie von keiner anderen Task, die Zugriff auf die gleiche Resource hat, unterbrochen werden. Durch diesen Mechanismus wird der Zugriff auf kritische Bereiche wie Speicher oder Ports limitiert.

## *Gemischtes System*

Bei einer Zusammenführung von OSEKtime und OSEK/VDX zu einem so genannten „Mixed System“ ist aus verständlichen Gründen bei der Konfiguration der Interrupts zu beachten, dass sämtliche OSEKtime Interrupts Vorrang vor OSEK/VDX Interrupts haben. OSEKtime kann so zwar unter Umständen die Prozessierung von OSEK/VDX IRQs verzögern, allerdings sollte der zeitkritische Anteil ausschließlich im OSEKtime Kontext bearbeitet werden. Ähnlich verhält es sich mit OSEK/VDX IRQs und OSEKtime Tasks: Wenn OSEK/VDX IRQs eine höhere Priorität haben als die OSEKtime Tasks, so verzögern die OSEK/VDX IRQs die Abarbeitung der OSEKtime Tasks und beeinflussen so das zeitliche Verhalten dieser zeitkritischen Tasks.

# OSEKtime FTCom

Um Determinismus im gesamten verteilten System erzielen zu können, ist es notwendig, dass nicht nur das Betriebssystem und die Applikation jedes Steuergerätes, sondern auch die Kommunikation zwischen den einzelnen Steuergeräten ein deterministisches Zeitverhalten aufweist. – Um diesen Determinismus auch auf Kommunikationsebene zu erzielen, werden speziell für sicherheitskritische Anwendungen zeitgesteuerte Kommunikationsprotokolle (wie z.B. FlexRay) eingesetzt.

Um von den Eigenheiten eines konkreten Kommunikationsprotokolls zu abstrahieren, hat das OSEK/VDX Komitee die Kommunikationsschicht OSEKtime FTCom definiert, welche eine standardisierte Schnittstelle für den Austausch von Nachrichten (d.h., Teilen von Botschaften) zwischen Tasks (auf eventuell verschiedenen Steuergeräten) zur Verfügung stellt.

## *Schichtenmodell von OSEKtime FTCom*

OSEKtime FTCom selbst ist in die Schichten unterteilt, welche in Abbildung 2 skizziert und in den folgenden Abschnitte genauer erläutert sind.

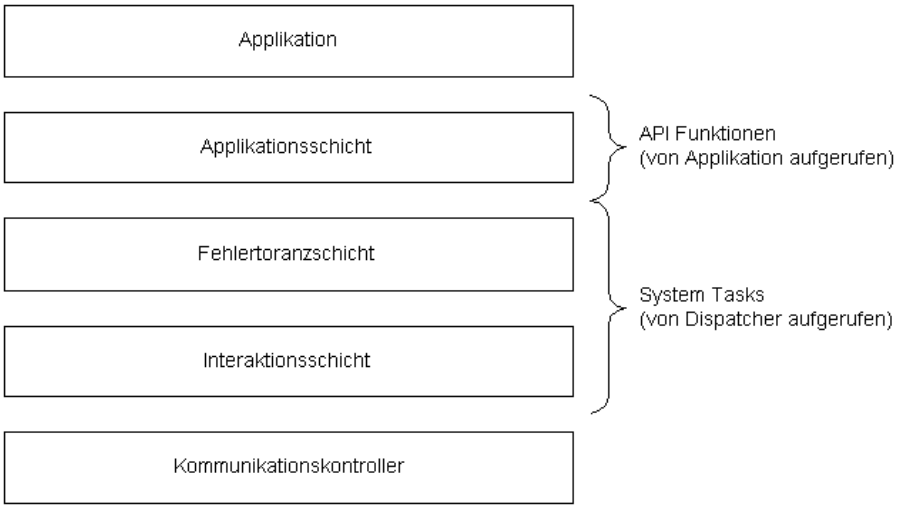


**Abbildung 2: Schichtenmodell von OSEKtime FTCom**

## Interaktionsschicht

Diese Schicht befasst sich mit den Repräsentationsbelangen von Nachrichten wie Nachrichtenausrichtung und Byte Ordnung. – Auf der Sendeseite kümmert sich diese Schicht darum, daß Nachrichten von der Byte Ordnung des Steuergerätes in die Byte Ordnung des Kommunikationsmediums umgewandelt werden. Weiterhin sorgt die Schicht beim Sender dafür, daß mehrere Nachrichten in eine einzelne Botschaft verpackt werden. – Auf der Empfängerseite extrahiert diese Schicht die Nachrichten aus den Botschaften und wandelt die Byte Ordnung der Nachrichten von der des Kommunikationsmediums in die des Steuergeräts um.

## Fehlertoleranzschicht

Aufbauend auf der Interaktionsschicht operiert die Fehlertoleranzschicht, deren Aufgabe es ist, sich um Fehlertoleranzbelange, nämlich Nachrichtenreplikation und Nachrichtenreduktion, zu kümmern. – Auf der Sendeseite repliziert diese Schicht eine einzige Applikationsnachricht und generiert daraus mehrere Nachrichteninstanzen. Diese Instanzen werden auf redundante Art und Weise (z.B., in verschiedenen Botschaften über verschiedene Kanäle bzw. zu verschiedenen Zeiten) versandt. – Auf der Empfängerseite werden dadurch mehrere Botschaften, die Instanzen derselben Nachricht beinhalten, empfangen. Nach der Extraktion dieser Nachrichteninstanzen durch die Interaktionsschicht, reduziert die Fehlertoleranzschicht die Instanzen und generiert dadurch eine einzige Applikationsnachricht für die höher liegenden Schichten.

Die Anzahl der Nachrichteninstanzen pro Applikationsnachricht sowie der verwendete Reduktionsalgorithmus hängt einerseits vom angenommenen Fehlermodell[1] und andererseits von der Anzahl der Fehler, die toleriert werden sollen[2], ab.

Neben einer fixen Anzahl an vordefinierten Reduktionsalgorithmen stellt OSEKtime FTCom auch die Möglichkeit bereit, eigene Reduktionsalgorithmen zu definieren.

## Applikationsschicht

Die oberste Schicht, die sogenannten Applikationsschicht, bildet die Applikationsschnittstelle von OSEKtime FTCom. Zur Nachrichtenübertragung bietet diese Schicht drei verschiedene Funktionen, nämlich zum Versenden, zum Empfangen und zum Invalidieren[3] von Nachrichten.

---

[1] Im Falle von konsistenten Fehlern im Wertebereich wäre zum Beispiel ein Majoritätsvotum ein geeigneter Reduktionsalgorithmus.

[2] Um einen konsistenten Fehler im Wertebereich zu tolerieren, sind beispielsweise 3 Nachrichteninstanzen notwendig.

### *Systemtasks und Schnittstellenfunktionen*

Während die Funktionalität der Interaktionsschicht und der Fehlertoleranzschicht vor dem Applikationsentwickler versteckt sind und durch sogenannte Systemtasks bewerkstelligt werden, ist die Funktionalität der Applikationsschicht in Schnittstellenfunktionen verpackt. Die Aktivierung der Systemtasks erfolgt hierbei durch das OSEKtime Betriebssystem, während die Schnittstellenfunktionen über Funktionsaufrufe vom Applikationsprogramm aufgerufen werden.

### *Globale Zeit – Synchronisation*

Zusätzlich zu den bisher vorgestellten Diensten zur Nachrichtenübertragung bietet OSEKtime FTCom auch zeitbezogene Dienste an. Zu diesen Diensten gehören Schnittstellenfunktionen, welche die Synchronisation zwischen OSEKtime Betriebssystem und dem zeitgesteuerten Kommunikationssystem erledigen, sowie Funktionen, die eine standardisierte Schnittstelle zum Zugriff auf die globale Zeit des Kommunikationssystems bereitstellen.

## Toolkette

Wie bei Betriebssystemen dieser Art üblich, wird eine statische Anpassung des Betriebssystems vor dem Übersetzungszeitpunkt vorgenommen. Dadurch kann der Betriebsystemkernel für die jeweilige Applikation optimiert werden.

Die Aktionen, die von OSEKtime FTCom durchgeführt werden, sind sehr rechenintensiv, bieten jedoch ebenfalls ein großes Optimierungspotential. Manche der möglichen Optimierungen jedoch können nur dann durchgeführt werden, wenn gewisse Parameter (wie zum Beispiel Nachrichtenlänge, Position der Nachricht in der Botschaft, ...) bereits zur Übersetzungszeit bekannt sind[4]. Bei Verwendung eines generischen Kommunikationsstacks ist dies im Allgemeinen nicht der Fall. – Wird nun jedoch in einem Generator basierten Ansatz Programmcode für jede einzelne Nachricht spezifisch generiert, dann sind genau diese Parameter bereits zur Übersetzungszeit bekannt. – Deshalb können in diesem Fall diese Optimierungen durchgeführt werden, was in einem besseren Laufzeitverhalten resultiert.

Die Information über die Nachrichten, die zwischen den verschiedenen Steuergeräten ausgetauscht werden, sowie die Zeitpunkte, wann so ein Nachrichtenaustausch passiert, sind in einer Entwurfsdatenbank (z.B. Bordnetzdatenbank) abgelegt. Diese Datenbank enthält weiterhin Informationen, wann Applikations- bzw. Systemtasks gestartet werden müssen. In der OSEKtime Toolkette verwaltet das Entwurfswerkzeug diese Entwurfsdatenbank und stellt eine graphische Benutzerschnittstelle zur Modifikation dieser Datenbank zur Verfügung.

Ausgehend von Vorgaben des Benutzers wie Periode von Nachrichten und Applikationstasks wird ein Kommunikationsschedule und eine entsprechende Konfiguration für die beteiligten Kommunikationskontroller erstellt. Basierend auf der Information, die in der Entwurfsdatenbank abgelegt ist, produziert der FTCom Generator eine OSEKtime FTCom kompatible Kommunikationsschicht und nimmt hierbei Rücksicht auf die Eigenheiten jeder einzelnen versandten Nachricht. Zusätzlich wird vom OIL Exporter eine Datei im OIL[5] Format generiert, welche Information über die Aktivierungszeiten der Applikations- und Systemtasks sowie eine Beschreibung des Kernels hinsichtlich der Betriebsmittel, die von der zu Grunde liegenden Applikation verwendet

---

[3] In zeitgesteuerten Kommunikationssystemen ist es nicht möglich, auf das Versenden von Botschaften zu verzichten. – Es ist nur möglich, Botschaften mit für den Empfänger erkennbar ungültigem Inhalt zu versenden.

[4] Der Nachrichtenextraktionsprozess zum Beispiel benötigt eine Schiebe- und eine Maskierungsoperation. Wenn die Nachricht allerdings an Bytegrenzen innerhalb der Botschaft liegt und die Nachrichtenlänge ein Vielfaches eines Bytes ist, dann kann sowohl die Maskierungsoperation als auch die Schiebeoperation entfallen. Diese Optimierung ist allerdings nur dann möglich, wenn Nachrichtenlänge und Nachrichtenposition bereits zur Übersetzungszeit bekannt sind.

[5] OSEK Implementation Language

werden, beinhaltet. Diese Informationen verwendet der OSEK Konfigurator, um einen bezüglich der aktuellen Erfordernisse optimierten Kernel sowie die zugehörige Dispatchertabelle zu generieren.

Sowohl die generierte Kommunikationsschicht, als auch das generierte Betriebssystem werden schließlich mit dem Applikationscode kompiliert und gelinkt und bilden letztendlich das ausführbare Programm. Abbildung 3 illustriert diesen Prozeß.

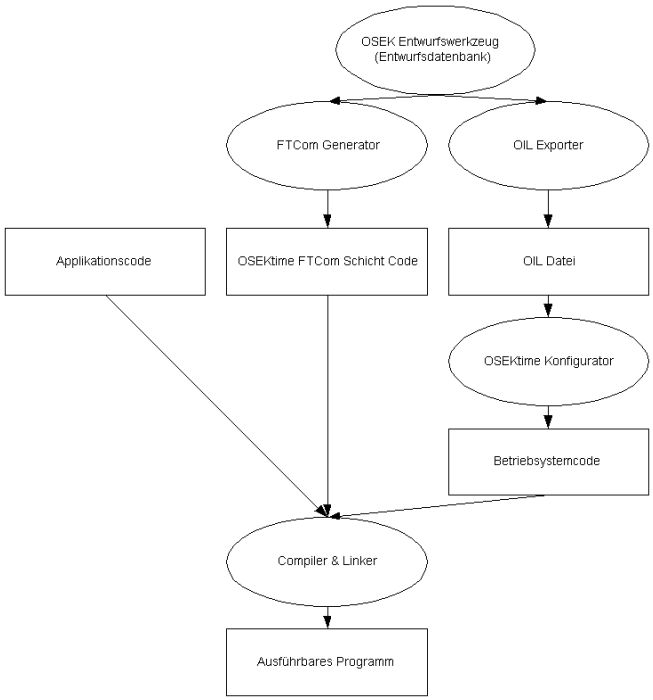


**Abbildung 3: Toolkette**

Auf diese Weise ist sichergestellt, daß das ausführbare Programm ein Betriebsystem und eine Kommunikationsschicht enthält, die speziell für die in der Entwurfsdatenbank modellierte Applikation optimiert ist.

# Zusammenfassung

OSEKtime bietet eine Kombination aus einem zeitgesteuerten Betriebssystem und einer fehlertoleranten Kommunikationsschicht. Aufgrund der durch Zeitsteuerung erreichten hohen zeitlichen Synchronität sowohl auf Betriebsystem- als auch auf Kommunikationsebene sowie der inherenten Fehlertoleranz von FTCom ist diese Kombination hervorragend für den Einsatz in sicherheitsrelevanten Applikation im Automobil geeignet. Mittels der integrierten OSEKtime Toolkette wird die Applikationsentwicklung vom Entwurf bis hin zum hochoptimierten ausführbaren Programm unterstützt.