TimeCore - Software Komponenten und Entwurfswerkzeuge 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

1 Einleitung

In der Automobilindustrie herrscht der Trend vor, bestehende mechanische oder hydraulische Teilsysteme durch elektronische Gegenstücke zu ersetzen. Bis dato beschränkten sich diese Bestrebungen jedoch ausschließlich auf nicht sicheheitsrelevante Applikationen. In den letzten Jahren wird jedoch vermehrt über den Einsatz von elektronischen Systemen in sicherheitsrelevatenen Anwendungsbereichen nachgedacht. Als Beispiel wären hier die vollelektronische Bremse (break-by-wire) oder Lenkung (steer-by-wire) zu nennen. Nachdem bei solchen sicherheitsrelevanten Applikationen der Ausfall einzelner Komponenten fatale Folgen nach sich ziehen kann, verlangen Anwendungen dieser Art nach hoher Zuverlässigkeit und sicherer Realisierung, was oft einen ausgesprochen hohen Grad an Komplexität impliziert.

Um dieser Komplexität in einem Umfeld von immer kürzer werdenden Entwicklungszyklen Herr zu werden, ist es unabdingbar, bei der Entwicklung solcher Systeme auf standardisierte Software Komponenten mit einheitlichen Schnittstellen zurückzugreifen. – Diese Entwicklung wird besonders durch die gemeinsamen Bestrebungen von Automobilfirmen, Zulieferern und Toolherstellern im Rahmen des AUTOSAR (AUTomotive Open System Architecture) Konsortiums deutlich.

TimeCore, ein gemeinschaftliches Produkt der Firmen DECOMSYS GmbH und 3SOFT GmbH, bietet eine abgestimmte Kombination aus solch standardisierten Software Komponenten wie zum Beispiel Betriebsystem [3], fehlertolerante Kommunikationsschicht [4], Netzwerkmanagement [2], oder Transportschicht [1] und den dazugehörigen Entwicklungswerkzeugen. Abbildung 1 illustriert die Software Komponenten von TimeCore.

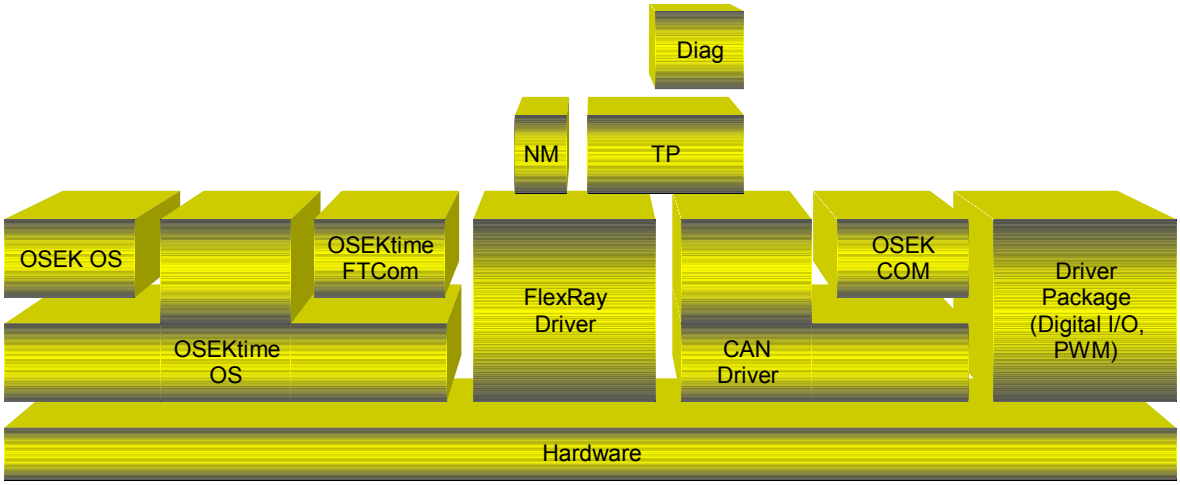

Abbildung 1: TimeCore Software Komponenten

Im Folgenden wird nun detailliert auf die Hauptkomponenten von TimeCore sowie auf den dazugehörigen Entwicklungsprozess eingegangen.

2 OSEKtime OS

OSEKtime OS ist ein zeitgesteuertes Betriebsystem, bei dessen Entwurf die Ziele Vorhersagbarkeit und Determinismus im Vordergrund standen. In OSEKtime OS wird zwischen zwei logischen Einheiten, in denen Aktionen ausgeführt werden können, unterschieden.

- OSEKtime Tasks werden zu definierten Zeitpunkten vom OSEKtime Dispatcher aktiviert und anschließend bis zu ihrer Fertigstellung abgearbeitet.
- Interrupt Service Routinen (ISRs) hingegen dienen zur Ausführung von asynchronen Diensten. Die Aktivierung dieser ISRs erfolgt durch Interrupts und erfolgt somit asynchron zur normalen Taskabarbeitung.

2.1 Taskmodell und Dispatcher

Im Gegensatz zu konventionellen Betriebsystemen sieht OSEKtime keine Betriebsystemdienste zur blockierenden Synchronisation zwischen Tasks zur Laufzeit vor, was die Analyse der maximalen Abarbeitungszeit (worst case execution time (WCET)) [5,6,7,8,9] einer Task sowie die Analyse des Laufzeitverhaltens des Gesamtsystems zur Entwurfszeit wesentlich vereinfacht.

OSEKtime Tasks können sich zu jedem Zeitpunkt in genau einem von drei Zuständen befinden. Für den Fall, dass eine Task gerade im Besitz der CPU ist, so befindet sie sich im Zustand *running*. Diesen Zustand kann immer nur genau eine Task einnehmen. Die beiden anderen Zustände können von mehreren Tasks gleichzeitig eingenommen werden. Im Zustand *preempted* befindet sich eine Task, wenn sie von einer anderen Task unterbrochen wurde. Dieser Zustand kann nur dadurch wieder verlassen werden, dass die unterbrechende Task ihre Abarbeitung beendet und selbst in den Zustand *suspended* überwechselt. Tasks, die sich im Zustand suspended befinden, sind inaktiv und warten auf ihre nächste Aktivierung durch den Dispatcher. Diese Taskzustände sowie die zugehörigen Zustandsübergänge sind in Abbildung 2 dargestellt.

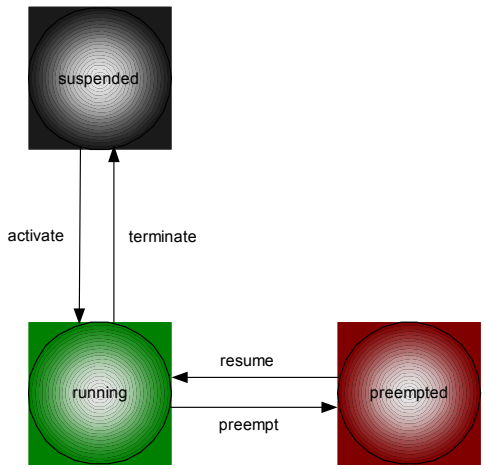

Abbildung 2: OSEKtime Task Zustände

Die Aktivierungszeitpunkte der Tasks werden zur Entwurfszeit des Systems in einer sogenannten Dispatchertabelle abgelegt. Diese Tabelle wird im Allgemeinen durch einen komplizierten Schedulingprozess erzeugt, der allerdings im Gegensatz zu konventionellen Betriebsystemen zur Entwurfszeit des Systems erfolgt und nicht zur Laufzeit. Zur Laufzeit aktiviert der OSEKtime Dispatcher als integraler Bestandteil des Betriebsystems basierend auf dieser Dispatchertabelle die einzelnen Tasks zu den definierten Zeitpunkten. Befindet sich eine Task noch im Zustand running, obwohl eine andere Tasks laut Disptachertabelle bereits aktiviert werden soll, so wird die gerade laufende Task durch die neu zu aktivierende unterbrochen und in den Zustand preempted versetzt. Diese Art von Dispatching wird als stack basiertes Dispatching bezeichnet. In dieser Art von

Dispatching ist keiner Task explizit eine Priorität zugeordnet. Einzig die zeitliche Abfolge der Tasks bestimmt ihre Vorrangbeziehung untereinander.

Der Dispatcher wird typischerweise über einen nicht maskierbaren Interrupt getriggert, dessen Frequenz an die lokale Zeitbasis gekoppelt ist. Am Ende einer Dispatcherrunde, nachdem die letzte zeitgesteuerte Task beendet ist, kann die Synchronisation der Dispatcherzeit mit einer externen Zeitreferenz (z.B. mit der globalen Zeit eines Kommunikationssystems wie FlexRay [10,11]) erfolgen. Dies geschieht durch ein passendes Verlängern bzw. Verkürzen der aktuellen Dispatcherrunde.

Abhängig von der Anordnung der Tasks innerhalb der Dispatchertabelle sowie von der maximalen Abarbeitungszeit dieser Task ist es möglich (und sogar sehr wahrscheinlich), dass Zeiträume quasi ungenutzt bleiben, d.h., dass in diesen Zeiträumen weder eine zeitgesteuerte Task noch eine Interrupt Service Routine aktiv ist. In diesen Leerlaufzeiträumen wird eine vom Betriebsystem zur Verfügung gestellte Task namens *ttIdleTask* ausgeführt, in der nicht zeitkritische Aufgaben abgearbeitet werden können.

In Abbildung 3 ist ein Beispiel für die zeitliche Abarbeitung von drei Tasks durch ein OSEKtime Dispatcher veranschaulicht.

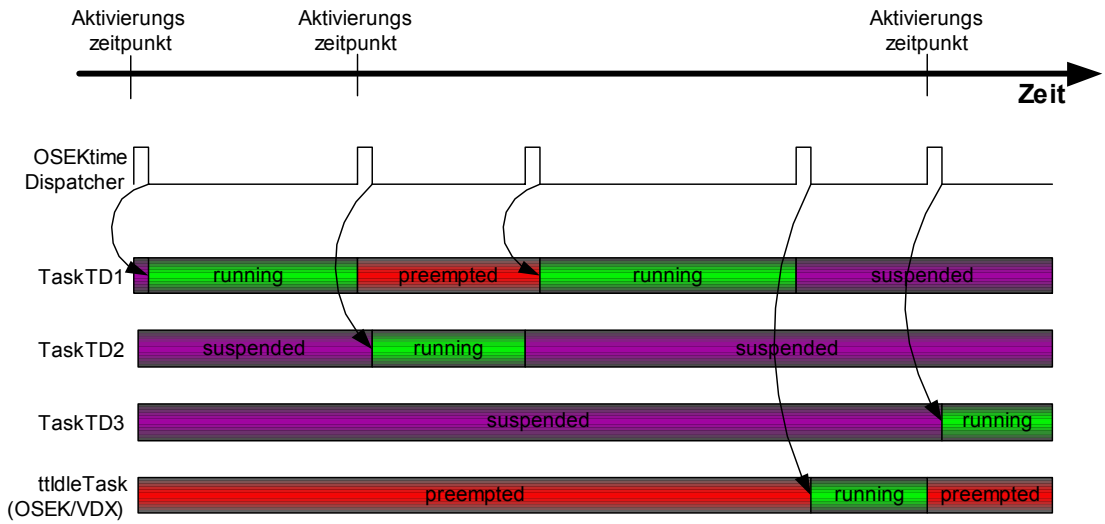

Abbildung 3: Arbeitsweise des OSEKtime Dispatchers

Am Anfang der Dispatcherrunde ist zuerst der OSEKtime Dispatcher aktiv, der dann die Task TD1 aktiviert. Beim nächsten Aktivierungszeitpunkt unterbricht der Dispatcher die Task TD1, um die Task TD2 zu aktivieren. Die Task TD1 wechselt also in den Zustand preempted. Nach Beendigung der Task TD2 wird wiederum der Dispatcher aktiv und setzt die unterbrochene Task (Task TD1) 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 TD3) aktiviert.

2.2 Deadline Überprüfung

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 durch das Betriebsystem eine entsprechende Fehlerbehandlung durchgeführt. Wird für eine Task keine explizite Deadline spezifiziert, so erfolgt ein automatischer Deadlinecheck am Ende der Dispatcherrunde. – Die einzige Ausnahme bildet hier die ttIdleTask, der keine Deadline zugeordnet ist.

2.3 Interrupt Überwachung

Sobald in OSEKtime ein Interrupt auftritt, wird dieser Interrupt durch das Betriebsystem deaktiviert. Erst anschließend wird die zugehörige Interrupt Service Routine aufgerufen. Ein Reaktivieren eines deaktivierten Interrupts erfolgt über einen speziellen Eintrag in der Dispatchertabelle.

2.4 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, das dem Anwender eine komfortable Palette von Betriebsmitteln zur Verfügung stellt.

3 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.

Die fehlertolerante Kommunikationsschicht OSEKtime FTCom von TimeCore bietet eine standardisierte Schnittstelle für den Austausch von Nachrichten (d.h. Teilen von Botschaften) zwischen Tasks (auf eventuell verschiedenen Steuergeräten) und abstrahiert so von den Eigenheiten eines konkreten Kommunikationsprotokolls.

3.1 Schichtenmodell von OSEKtime FTCom

OSEKtime FTCom selbst ist in die Schichten unterteilt, welche in Abbildung 4 skizziert und in den folgenden Abschnitten genauer erläutert sind.

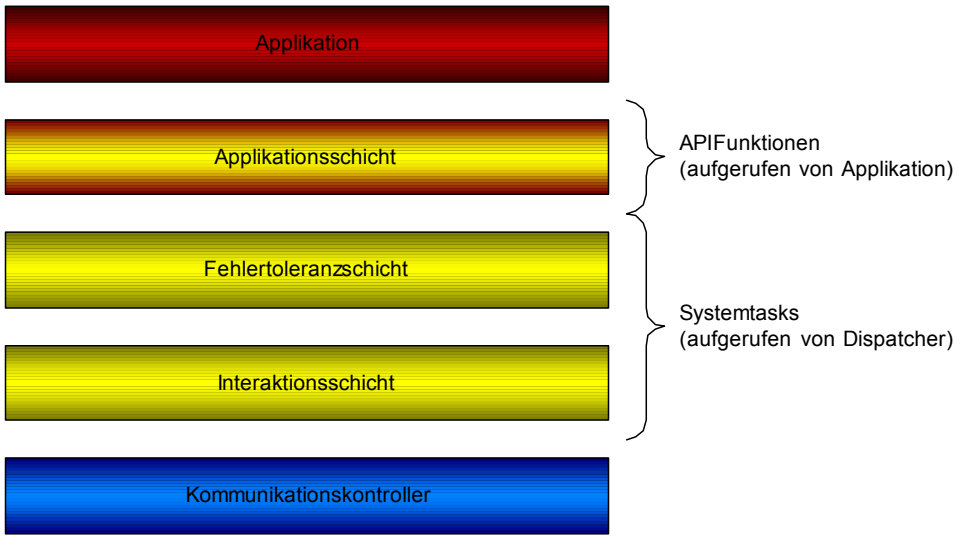

Abbildung 4: Schichtenmodell von OSEKtime FTCom

3.1.1 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.

3.1.2 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.

3.1.3 Applikationsschicht

Die oberste Schicht, die sogenannte 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.

3.2 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.

3.3 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.

4 Entwurfsprozess

Aufgrund der relativ simplen (und daher vorhersagbaren) Abarbeitung zur Laufzeit benötigen zeitgesteuerten Systeme einen besonders komplexen Entwurfsprozess. Sowohl das Erstellen eines geeigneten Kommunikationsschedules als auch die Planung eines Taskschedules, welches sämtliche erforderlichen Beziehungen zwischen verschiedenen Tasks (wie zum Beispiel gegenseitiger Ausschluss (Mutual Exclusion)) berücksichtigt, stellen für den Systemdesigner eine große Herausforderung dar. Um dem Systemdesigner beim Entwurf optimal zu assistieren, bietet TimeCore eine unterstützende Werkzeugkette sowie einen integrierten Entwurfsprozess an.

---

[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.

[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.

Ausgehend von einer sehr groben Planung der Systemfunktionen (z.B. Drive-By-Wire) erfolgt eine Partitionierung in Teilfunktionen (z.B. Steer-By-Wire und Brake-By-Wire). Eine solche Unterteilung wird so lange fortgesetzt, bis einzelnen Teilfunktionen eindeutig Hardwarekomponenten (z.B. einzelne Steuergeräte) zugeordnet werden können.

Anschließend werden diese Teilfunktionen noch weiter in Tasks untergliedert, für die dann bereits zur Entwurfszeit ein Task Schedule und eine zugehörige Dispatchertabelle erstellt wird. Abbildung 5 zeigt ein Werkzeug der TimeCore Werkzeugkette, das ein solches Task Schedule sowie das zugehörige Kommunikationsschedule darstellt.

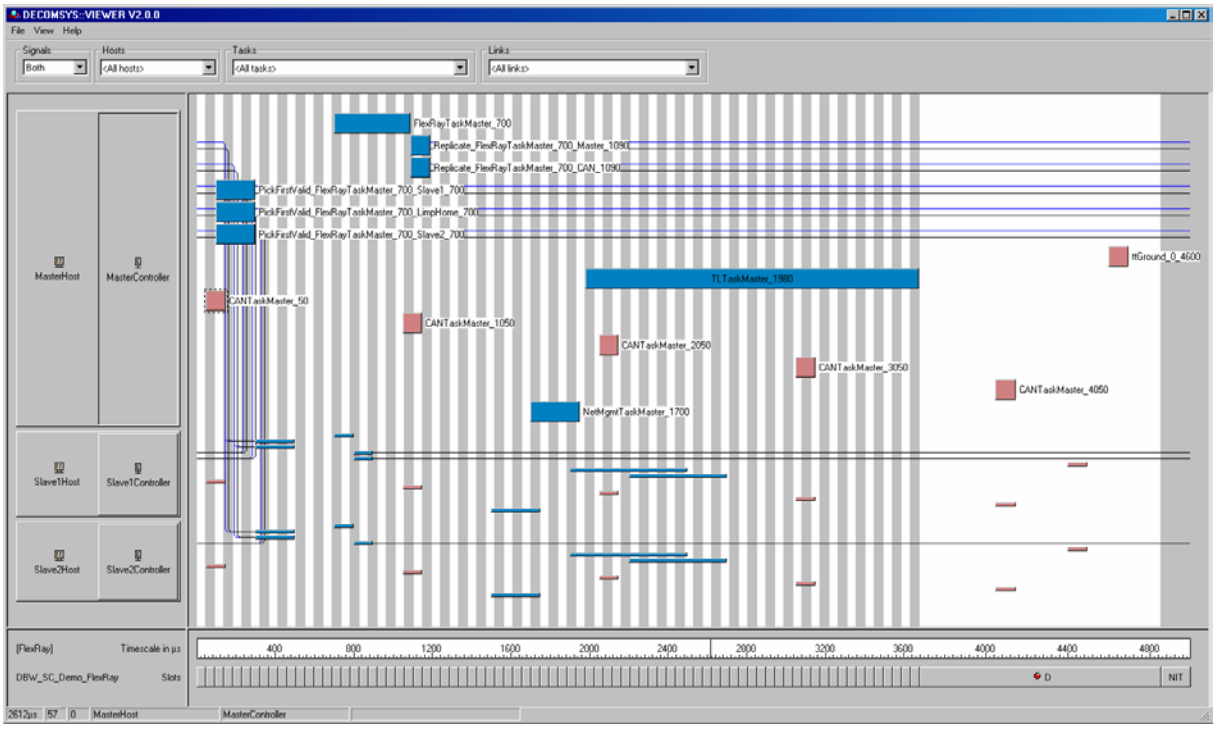

Abbildung 5: Task Schedule in der TimeCore Werkzeugkette

Hierbei sind die Tasks durch rote und blaue Blöcke veranschaulicht, während die Linien zwischen den Tasks die Signale, die zwischen diesen Tasks ausgetauscht werden, repräsentieren.

Um nun ein Taskschedule zu erstellen, das sämtliche Beziehungen zwischen den Tasks (z.B. gegenseitigen Ausschluss oder Vorrangbeziehungen) berücksichtigt und unter diesen Bedingungen noch die Deadlines der Tasks einhält, ist es notwendig, im Entwurfsprozess gewisse Annahmen über die zeitlichen Attribute von Tasks zu treffen. – Als wichtiges Beispiel sei hier die maximale Abarbeitungszeit (WCET) genannt, die man genau kennen muss, um effiziente Taskschedules erstellen zu können. Eine Unterschätzung dieser maximalen Abarbeitungszeit führt höchstwahrscheinlich zu einer Verletzung der Deadlines von Tasks, während eine Überschätzung eine Verschwendung von CPU Bandbreite bedeutet.

Für den Fall, dass die Taskabarbeitung durch Interrupts unterbrochen werden kann, ist es weiters notwendig, Annahmen darüber zu treffen, wie oft eine Task durch einen spezifischen Interrupt unterbrochen werden kann. Anschließend muss die maximale Abarbeitungszeit der zugehörigen Interrupt Service Routine entsprechend oft zur maximalen Abarbeitungszeit der unterbrochenen Task addiert werden.

Basierend auf diesen Annahmen kann schließlich, wie bereits angedeutet, unter Berücksichtigung des Laufzeitverhaltens des Dispatchers (z.B. Verhalten bei Taskunterbrechung) ein Taskschedule erstellt werden. – Zu diesem Zeitpunkt können bereits Aussagen getroffen werden, ob in dem dann vorliegenden Taskschedule alle Task ihre Deadlines einhalten oder nicht.

5 Laufzeitüberprüfungen

Wenn nun zur Laufzeit des Systems gewisse Annahmen, die zur Entwurfszeit getroffen wurden, nicht eingehalten werden (z.B. wenn eine Task ihre maximale Abarbeitungszeit überschreitet), dann weicht das tatsächliche Systemverhalten von dem während des Entwurfsprozess modellierten Verhalten ab. Im Allgemeinen ist es dann nicht mehr möglich, die zeitlichen Rahmenbedingungen (z.B. maximale Antwortzeit auf einen Stimulus) einzuhalten, was zu einem nicht vorhersagbaren Systemverhalten führt.

Insofern ist es essentiell, dass zur Laufzeit Mechanismen vorgesehen sind, die eine Einhaltung der zur Entwurfszeit getroffenen Annahmen sicherstellen.

Das OSEKtime OS Betriebsystem von TimeCore bietet hier zwei fundamentale Mechanismen, um diese Annahmen sicherzustellen.

- Mittels der Deadline Überprüfung ist es einerseits möglich für den Fall, dass ein Task seine maximale Abarbeitungszeit überschreitet, eine kontrollierte Fehlerbehandlung durchzuführen. Dies führt dazu, dass in diesem Fehlerfall die Reaktion des Systems nicht unvorhersagbar ist, sondern, dass das System hier ein klar definiertes Fehlerverhalten (z.B. kontrolliertes Abschalten und Neustart) aufweist.
- Der Mechanismus der Interrupt Überwachung dient dazu, die Annahmen über die maximale Unterbrechungen eines Tasks durch einen Interrupt zur Laufzeit zu erzwingen. Nachdem OSEKtime OS beim Auftreten eines Interrupts diesen sofort deaktiviert und eine Reaktivierung nur durch entsprechende Einträge in der Dispatchertabelle erfolgen kann, wird die gesamte Dispatcherrunde dadurch in Intervalle unterteilt, in denen der Interrupt garantiert maximal einmal auftritt.

Die Kombination dieser beiden Mechanismen stellt sicher, dass das Laufzeitverhalten des Systems von den Annahmen, die während des Systementwurfs getroffen wurden, nicht abweicht. Daher treffen sämtliche Aussagen über das Zeitverhalten (z.B. Antwortzeiten) des modellierten Systems die während des Entwurfsprozess getätigt wurden, auch für das reale System unter allen Umständen zu.

6 Zusammenfassung

TimeCore bietet eine Kombination aus standardisierten Software Komponenten wie zum Beispiel Betriebsystem und fehlertolerante Kommunikationsschicht sowie der dazugehörigen Entwicklungswerkzeugkette. Mittels dieser integrierten Werkzeugkette wird die Applikationsentwicklung vom Entwurf bis hin zum ausführbaren Programm unterstützt, wobei die Softwarekomponenten (z.B. OSEKtime OS) die notwendigen Mechanismen bereitstellen, um zur Laufzeit die Einhaltung der zur Entwurfszeit getroffenen Annahmen zu erzwingen.

Aufgrund der durch Zeitsteuerung erreichten hohen zeitlichen Synchronität sowohl auf Betriebsystem- als auch auf Kommunikationsebene, der inherenten Fehlertoleranz von OSEKtime FTCom und des erzielten vorhersagbaren Systemverhaltens ist TimeCore hervorragend für den Einsatz in sicherheitsrelevanten Applikation im Automobil geeignet.

7 Referenzen

[1] ISO (International Organization for Standardization), *Road Vehicles – Diagnostics on Controller Area Networks (CAN) – Part 2: Network Layer Services*, ISO/DIS 15765-2.2, 1, rue de Varembe, Case postale 56, CH-1211 Geneva 20, Switzerland, June 2003

[2] C. Hoffmann, J. Minuth, J. Krammer, J. Graf, K. J. Neumann, F. Kaag, A. Maisch, W. Roche, O. Quelenis, E. Farges, P. Aberl, D. John, L. Mathieu, M. Schütze, D. Gronemann, J. Spohr, *OSEK/VDX – Network Management - Concept and Application Programming Interface, Version 2.5.2*, January 16th 2003; Available at http://www.osek-vdx.org/mirror/nm252.pdf

[3] V. Barthelmann, A Schedl, E. Dilger, T. Führer, B. Hedentz, J. Ruh, M. Kühlewein, E. Fuchs, Y. Domaratsky, A. Krüger, P.Pelcat M. Glück, S. Poledna, T. Ringler, B. Nash, and T. Curtis, *OSEK/VDX - Time-Triggered Operating System, Version 1.0*, July 24th 2001; Available at http://www.osek-vdx.org/mirror/ttos10.pdf

[4] A. Schedl, E. Dilger, T. Führer, B. Hedenetz, J. Ruh, M. Kühlewein, E. Fuchs, T. M. Galla, Y. Domaratsky, A. Krüger, P. Pelcat, M. Taï-Leung, M. Glück, S. Poledna, T. Ringler, B. Nash, and T. Curtis, *OSEK/VDX - Fault-Tolerant Communication, Version 1.0*, July 24th 2001; Avalable at http://www.osek-vdx.org/mirror/ftcom10.pdf

[5] P. Puschner, *Timing Analysis for Real-Time Programs*, PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 3/3/182-1, 1040 Vienna, Austria, 1993

[6] Raimund Kirner, *Extending Optimising Compilation to Worst-Case Execution Time Analysis*, PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 3/3/182-1, 1040 Vienna, Austria, May 2003

[7] C. Y. Park and A. C. Shaw, *Experiments with a Program Timing Tool based on a Source-Level Timing Schema*, Computer, 24(5):48–57, May 1991

[8] P. Puschner and A. V. Schedl, *Computing Maximum Task Execution Times – A Graph-Based Approach*, The Journal of Real-Time Systems, 13:67–91, 1997

[9] A. Vrchoticky, *The Basis for Static Execution Time Prediction*, PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 3/3/182-1, 1040 Vienna, Austria, April 1994

[10] R. Mores, G. Hay, R. Belschner, J. Berwanger, C. Ebner, S. Fluher, E. Fuchs, B. Hedenetz, W. Kuffner, A. Krüger, P. Lohrmann, D. Millinger, M. Peller, J. Ruh, A. Schedl, and M. Sprachmann, *FlexRay - The Communication System for Advanced Automotive Control Systems*, SAE 2001 World Congress, Detroit, MI, USA, March 2001

[11] T. Führer, F. Hartwich, R. Hugel, H. Weiler, *FlexRay-The Communication System for Future Control Systems in Vehicles*, SAE 2003 World Congress & Exhibition, Detroit, MI, USA, March 2003