Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Seminar

Modellierung von Event- und Time-Triggered Systemen

Yegor Yefremov

Fakultät Technik und Informatik
Studiendepartment Informatik
Datum: 15. Februar 2007
Masterkurs Informatik
Betreuer: Prof. Dipl.-Inf. Heiner Kaltenhäuser

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Einführung

Die Fortschritte in der Entwicklung vom Computer- und Kommunikationssystemen machen es möglich neue sicherheitskritische Funktionen wie z.B. X-by-wire in Kfz- und Flugzeugindustrie auf die eingebetteten Kontroller zu übertragen. Somit erhöht sich die Anzahl an *Electronic Control Units (ECUs)* und damit die Anzahl der auszutauschenden Signale. In einem modernen Fahrzeug werden z.B. bis zu 2500 Signale zwischen bis zu 70 ECUs ausgetauscht, was die heutigen PKWs zu hoch verteilten Systemen macht [1]. So ein System stellt ein Gemisch aus sicherheitskritischen und nicht sicherheitskritischen Funktionen, die auf verschiedene Art und Weise verarbeitet werden. Die verteilten Echtzeitsysteme werden je nach der Betrachtungsweise folgender Maßen unterteilt [13, 15]:

- Schwerpunkt der Steuerung
  - *Ereignisgesteuerte Architektur (event-triggered architecture)*
  - *Zeitgesteuerte Architektur (time-trigered architecture)*
- Grad der Integration
  - *föderierte Systeme (federated systems)*
  - *integrierte Systeme (integrated systems)*

Die Systemaktivitäten, wie Senden einer Nachricht oder Starten einer Berechnung, werden in ereignisgesteuerter Architektur vom Eintreten eines Ereignisses in der Umgebung oder Computersystem angestossen. Dagegen werden die Aktivitäten in den zeitgesteuerten Architekturen durch zyklisch auftretende Zeitmarker „angestoßen“, die von Timern erzeugt werden.
In föderierten Systemen bekommt jede Anwendung wie z.B. Autopilot oder Brake-by-wire System ein eigenes verteiltes Rechnersystem mit möglicher interner Redundanz. Die Subsysteme werden mittels Gateways miteinander verbunden. Vorteile von solchen Systemen sind zu einem gegenseitige hardwaremäßige Trennung von Anwendungssystemen, zu anderem erhöht sich die Transparenz von Interaktionen und Abhängigkeiten zwischen autonomen Systemen. Ein integriertes System integriert dagegen mehrere Anwendungssubsysteme innerhalb eines verteilten Systems. Ein Hauptvorteil dabei ist die Kostenreduktion durch die Mehrfachnutzung der Hardwareressourcen.
Ein aktueller Trend in der Kfz-Industrie ist ein neues System zu schaffen, dass die Vorteile von föderierten und integrieren Systemen nutzt. Dafür muss ein integriertes System den gleichen Grad an *Fehlerisolation (fault isolation)* und *Fehlereingrenzung (error containment)* von föderierten Systemen erreichen. Dieses Konzept verspricht eine bessere Koordination von Steuerungsfunktionen und eine erhebliche Reduktion von Hardwareressourcen und eine Gesamtverbesserung der Zuverlässigkeit des Systems. Um das Konzept realisieren zu können, bedarf es ein geeignetes Framework, das eine Integration von ereignis- und zeitgesteuerten Architekturen ermöglicht. Um ein solches System modellieren und verifizieren zu können, bedarf es ein hybrides Modellierungskonzept für ereignis- und zeitgesteuerte Systeme, was ein Thema dieser Ausarbeitung darstellt.
Die ereignisgesteuerten und zeitgesteuerten Konzepte sind für sich alleine gut erforscht und es existieren jeweils mehrere Modellierungsansätze. Von den vielen möglichen Modellierungsansätzen werden hierarchische Automaten zunehmend für die Modellierung von ereignisgesteuerten Systemen eingesetzt. Die Gründe sind hohe Verbreitung und die Möglichkeit ein komplexes dynamisches System kompakt und übersichtlich modellieren zu können. Das Ziel der Masterarbeit ist ein zeitgesteuertes Modellierungskonzept in das Konzept von hierarchischen Automaten einzubauen. Daher soll eine an der Hochschule für Angewandte Wissenschaften Hamburg entwickelte Sprache AIRA als Vertreter der hierarchischen Automaten um das zeitgesteuertes Konzept erweitert.

Wenn ein hybrides Modellierungskonzept existiert, dann bedarf es ein Framework, das die Modellelemente unterstützt, und eine Laufzeit, die Ausführung von ereignis- und zeitgesteuerten Reaktionen organisiert. Folgende zur Zeit in der Industrie eingesetzte Architekturen für die verteilten Echtzeitsysteme sind unter Anderem MAFT[1][10], IMA[2][13] in der Flugzeugindustrie und BASEMENT[7], OSEK/VDX und OSEKtime[16, 6, 12] in der Kfz-Industrie. In dieser Ausarbeitung wird speziell auf das OSEKtime-Betriebsystem eingegangen, weil darin vorgestelltes Scheduling-Konzept den Anforderungen an die hybride Laufzeit sehr nahe kommt.
Neben den modellbasierenden Methoden gibt es auch die formalen Methoden, um hybride Systeme modellieren zu können. Diese sind unter Anderem *LTTA (Loosely Time-Triggered Architecture)*[4, 3] und *timed automata*[2]. *LTTA* ist ein mathematisches Framework für heterogene Modellierung von reaktiven und Echtzeitsystemen, dass eine Freiheit der Wahl zwischen verschiedenen Synchronisationsmethoden an verschiedenen Entwurfsphasen erlaubt. Der Fokus des Frameworks liegt in Behandlung der Kommunikation und Koordinierung zwischen heterogenen Prozessen auf dem mathematisch sicheren Wege. Ein *timed automata* ist ein endlicher Automat, das das Verhalten des Echtzeitsystems über die Zeit modelliert. Dabei werden die Zustandstransitionsgraphen mit zeitlichen Begrenzungen auf Basis von mehreren Uhren ergänzt. Ein hierarchischer *timed automata* wird in [5] vorgestellt. Mit einem in [11] vorgestellten Verfahren lässt sich ein *timed automata* in einen *time-triggered automata* konvertieren. Die oben kurz vorgestellten Formalismen könnten bei Bedarf in der Masterarbeit als Alternative oder Ergänzung zur AIRA oder Erweiterung des Konzeptes um die verteilte Ausführung in Betracht gezogen werden.

2 Echtzeitsysteme: Schwerpunkt der Steuerung

2.1 Verteilte Echtzeitsysteme

Ein Echtzeitkomputersystem ist ein System, in dem die Korrektheit des Systemverhaltens nicht nur von logischen Ergebnissen von Berechnungen abhängt, sondern der Einhaltung von bestimmten Zeitgrenzen (deadlines) zur Ermittlung der logischen Ergebnissen. Falls ein Ergebnis nutzbar bleibt, sogar nachdem die Zeitgrenze überschritten ist, so wird die Zeitgrenze als *weich (soft)* bezeichnet. Zeitgrenze ist *hart (hard)*, falls sich die fatalen Folgen wegen der Überschreitung der Zeitgrenze ergeben können [13].

2.2 Ereignisgesteuerte Systeme

Bei einem *ereignisgesteuerten System* wird das nächste anstehende Ereignis gewählt. Dieses Ereignis aktiviert das System. Die Zeitvariable wird gemäß der Zeit des nächsten Eintreffens eines Ereignisses aktualisiert [9].

[1] Multicomputer Architecture for Fault-Tolerance
[2] Integrated Modular Avionics

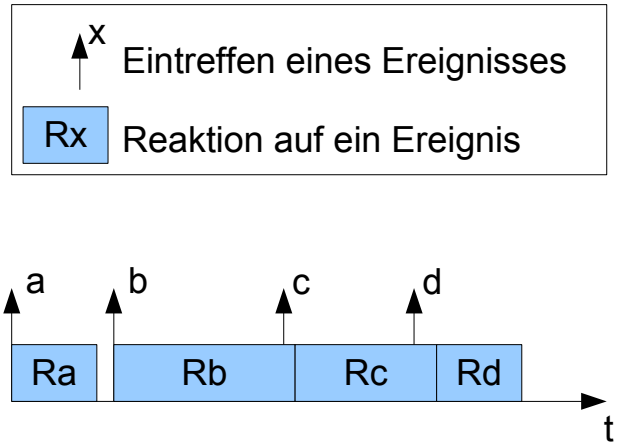


Abb. 1: Ereignisgesteuertes System

In der Abbildung 1 wird ein mögliches Scenario für ein ereignisgesteuertes System dargestellt. Auf Ereignis *a* wird sofort reagiert. Somit kann beim Eintreffen von Ereignis *b* ebenfalls sofort reagiert werden. Beim Eintreffen von Ereignis *c* dagegen ist das System immer noch mit der Bearbeitung des Ereignisses *b* beschäftigt, daher muss Ereignis *c* von der Laufzeit gepuffert werden. Deswegen verschiebt sich auch die Reaktion auf Ereignis *d*. Die Notwendigkeit Ereignisse zwischenzupuffern verlangt den Einsatz von *Ereignisqueues*. Bei einer fehlerhaften Konfiguration des Systems kann eine Ereignisqueue den Schwachpunkt darstellen, da beim Überlauf ein Verlust der Nachrichten erfolgen kann. Außerdem wird aus der Abbildung 1 ersichtlich, das solche Systeme einen Jitter aufweisen.

2.3 Zeitgesteuerte Systeme

In einem *zeitgesteuerten System* veranlasst das Fortschreiten der Zeit das System oder Teile davon sich zu aktivieren. Wenn die Zeitvariable sich erhöht, werden alle Ereignisse, die zu dieser Zeit anstehen, an die entsprechenden Eingänge angewendet und das System oder dessen Teile werden aktiviert mittels Verbrauchens dieser Ereignisse. Falls kein Ereignis zur bestimmten Zeit ansteht, wird ein „null“ Ereignis angenommen, das keine Zustandsänderung hervorruft [9].

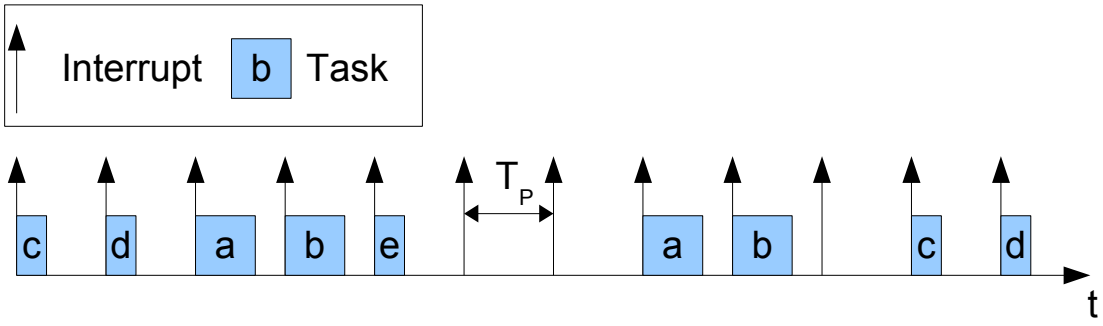


Abb. 2: Zeitgesteuertes System

In der Abbildung 2 wird ein mögliches Scenario für ein zeitgesteuertes System nach Pont [14] dargestellt. Die periodischen Time-Ticks lassen sich mit HW-Timer Interrupts genau reproduzieren. Auch stark unterschiedliche Perioden sind mit diesen Time-Ticks realisierbar. Eine Schachtelung von Tasks ist nicht erforderlich. Die Länge der Perioden bestimmt nicht den Code-Umfang. Eine variable Task-Ausführungsdauer hat keinen Einfluss auf die Periodizität anderer Tasks, daher entsteht dadurch kein Jitter.

3 Modellierung von dynamischen Systemen

3.1 Modellierungskonzepte

Es werden verschiedene Verfahren für die Modellierung von Echtzeitsystemen eingesetzt, wie z.B.: endliche Automaten wie Mealy und Moore, Petrinetze, temporäre Logik etc. Viele Autoren scheinen geeinigt zu sein, dass Zustände und Ereignisse sind *a priori* natürliches Medium um dynamisches Verhalten von komplexen Systemen zu beschreiben [8]. Die komplexe Systeme können aber wegen der exponentiell steigender Anzahl an Zuständen nicht mehr effizient mit z.B. Mealy/Moore Automaten beschrieben werden. Um das Zustand/Ereignis Konzept doch nutzen zu können, muss es modular, gut strukturiert und hierarchisch aufgebaut werden. Es soll unter Anderem das Problem des exponentiellen Wachstums durch die Auflockerung der Bedingung alle Zustandskombinationen explizit darzustellen gelöst werden. Solch ein Konzept von hierarchischen Automaten wird in [8] vorgestellt. Die oben genannten Vorteile machen hierarchische Automaten zu einer geeigneten Wahl für die hybride Modellierung. Als Vertreter von hierarchischen Automaten wird AIRA näher betrachtet.

3.2 AIRA-Modellierungssprache

Automaten **I**n **R**eaktiven **A**nwendungen (AIRA) ist zuerst eine Modellierungssprache entwickelt von Professor Kaltenhäuser an der HAW-Hamburg. Der Grundkonzept basiert auf dem Konzept der hierarchischen Automaten [8]. Nach den zahlreichen Einsätzen der hierarchischen Automaten wurde gemerkt, dass die Semantik in der Praxis nicht alle Situationen deckt. In AIRA werden diese Erkentnisse durch folgende Veränderungen reflektiert:

- besondere λ-Übergänge, die nur an das Beenden von eingebetteten Aktivitäten gebunden sind
- Abschaffung von Wiederentrittsgedächtnis, das durch die frei wählbaren Eintrittspunkte ersetzt wurde
- determinierende Semantik der Ausführungsreihenfolge
- erweiterte Zeitsteuerung

Die λ-Übergänge gibt es in AIRA in zwei Ausprägungen (s. Abbildung 3): λ_{first} und λ_{last}.

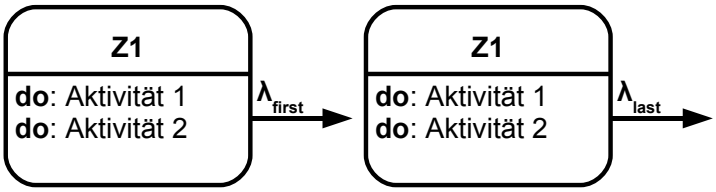


Abb. 3: λ-Übergänge

λ_{first}-Ereignis wird generiert, nachdem entweder *Aktivität 1* **oder** *Aktivität 2* beendet ist. λ_{last}-Ereignis wird generiert, nachdem *Aktivität 1* **und** *Aktivität 2* beendet ist.

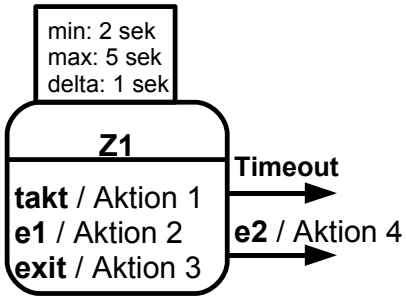


Abb. 4: Zeitsteuerung in AIRA

Die drei Möglichkeiten Zeitsteuerung in AIRA zu realisieren sind in der Abbildung 4 zu sehen:

- **min**: zustandsändernde Reaktionen werden erst nach 2 Sekunden freigeschaltet
- **max**: *Aktion 3* wird nach 5 Sekunden ausgeführt
- **delta(takt)**: *Aktion 1* wird jede Sekunde ausgeführt

Außer das Verhalten des Systems zu definieren beschreibt AIRA auch die Verteilung und Nebenläufigkeit des Systems durch Kontroller und Interfaces (s. Abbildung 5).

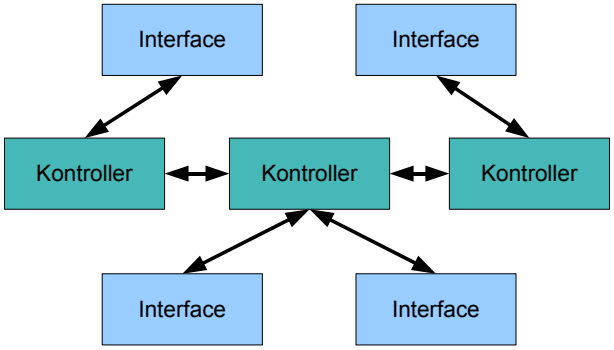


Abb. 5: Kontroller und Interfaces in AIRA

Die Kontroller sind Automatenrealisierungen und stellen Threads dar. Interfaces sind Schnittstellen zwischen dem technischen Prozess (unten) und dem Benutzer (oben). Sie setzten die Nachrichten des technischen Prozess bzw. des Benutzers in die für Automaten verständliche Nachrichten um und umgekehrt.
Somit stellt AIRA eine geeignete Basis für das geplante Framework:

- Es können vorherige ereignisgesteuerte Anwendungen ohne Veränderungen übernommen werden
- AIRA ist nicht proprietär und wird ständig erweitert
- AIRA ist Betriebsystemunabhängig

4 Scheduling von Tasks in der Laufzeit

Damit die ereignis- und zeitgesteuerten Tasks in einem System laufen können, muss die AIRA-Laufzeit um die zeitgesteurte Einheit mit einem geeigneten Schedulingverfahren erweitert werden. Zwei verschiedenen Schedulingverfahren sind in [13] vorgestellt. Dies sind *time sharing* und *two-level scheduling*. Der letzte Schedulingverfahren lässt sich einfacher implementieren. In dem *two-level scheduling* belegt der erste Scheduler die CPU für die zeitgesteuerten Tasks. Falls keine

zeitgesteuerten Tasks ausgeführt werden sollen, wird die CPU dem zweiten Scheduler für die ereignisgesteuerten Tasks übergeben. Dieser Verfahren ist in OSEKtime implementiert, daher wird hier unten dieses Betriebssystem beschrieben.

4.1 Konzepte des OSEKtime Betriebsystems

Seit 1993 arbeitet die Automobilindustrie im Rahmen der OSEK/VDX-Initiative an der Standardisierung wichtiger Elemente einer fahrzeugweiten Elektronik-Infrastruktur. Das Portfolio der inzwischen verabschiedeten Standards umfasst Betriebsystem (OSEK/VDX-OS), Kommunikationssystem (OSEK/VDX-COM), Netzmanagement (OSEK/VDX-NM) und OSEK Implementation Language (OSEK/VDX-OIL) [16].
Der OSEK/VDX-Standard wird inzwischen bei zahlreichen Automobil-Herstellern und Zuliefern genutzt. Ein großer Teil aktueller Steuergeräte wird bereits auf der Basis verschiedener OSEK-Implementierungen realisiert. Dabei hat sich gezeigt, dass diese Systeme für typischen Anforderungen am Innenraumbus und Powertrain sehr gut geeigent sind. Hier findet man heute lose Verbünde weitgehend autonomer Steuergeräte.
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.

4.1.1 Die Kommunikationsschicht FTCOM

OSEKtime besteht aus zwei Komponenten (s. Abbildung 6):

- Betribsystemkern OSEKtime
- spezielles Kommunikationssystem FTCOM (Fault Tolerant COMunication)

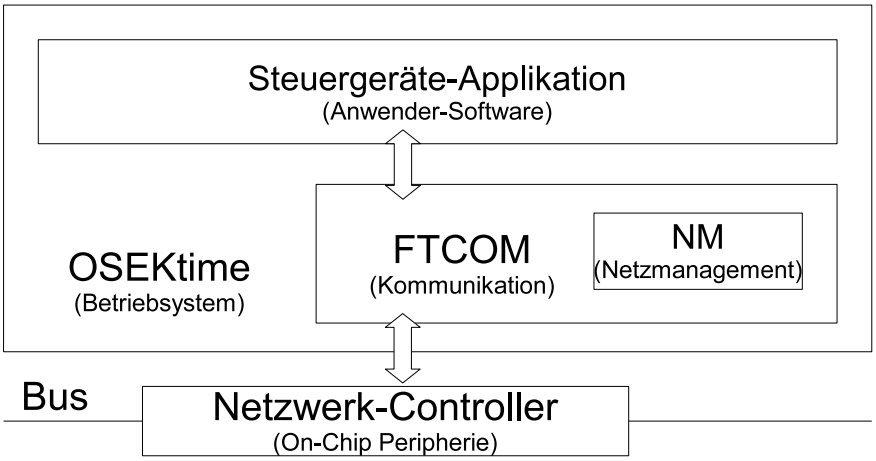


Abb. 6: Schematischer Aufbau eines OSEKtime-Systems

FTCOM ist außer Kommunikationsaufgaben auch für die Bereitstellung einer globalen Zeit in allen Knoten vernatwortlich. Es ist über verschiedene Ausbaustufen skalierbar und bietet alle seine Vorteile in der höchsten dieser Stufen an, während niedrigere Stufen nur Teilfunktionalitäten anbieten. Die niedrigste Stufe von FTCOM wird auch von heute gebräuchlichen Kommunikationsschichten bereits erfüllt.

4.1.2 Betriebsmittel unter OSEKtime

In seiner Grundcharakteristik ist OSEKtime ein Multitasking-Betriebsystem. Es beinhaltet zwei Instanzen, die die Funktionen ausführen können:

- *interrupt service routines (ISRs)*
- *tasks*

In OSEKtime gibt es sowohl nicht maskierbare ISRs, die am Betriebssystem vorbei laufen, als auch maskierbare, die das Betriebssystem kontrollieren kann. OSEKtime erlaubt sogar solche ISRs, die Tasks nicht unterbrechen können und so erst beim nächsten Dispatching zum Zuge kommen. Diese Möglichkeit ist besonders im Zusammenspiel mit der noch zu diskutierenden Deadline-Überwachung wichtig, da so sichergestellt werden kann, das bestimmte ISRs eine Task-Ausführung nicht beeinflüssen.
Jede Task wird zu einem definierten Zeitpunkt gestartet und kann einen von drei möglichen Zuständen annehmen (s. Abbildung 7). Im Zustand *suspended* ist die Task inaktiv, im Zustand *running* wird sie gerade ausgeführt und im Zustand *preemted* ist ihre Ausführung durch eine andere Task unterbrochen worden. Die Task kann den Zustand *preemted* nur dann verlassen, wenn die unterbrechende Task zum Zustand *suspended* gewechselt hat. Tasks besitzen in OSEKtime keine Prioritäten.

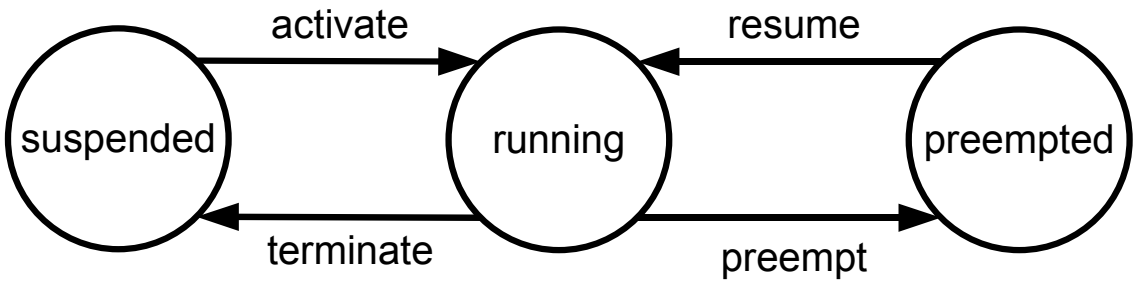


Abb. 7: OSEKtime Taskzustände

Die Aktivierungszeitpunkte der Tasks werden in einer Dispatcher-Tabelle vor der Kompillierung gespeichert. Der Dispatcher ist eine zentrale Komponente des OSEKtime Betriebssystems und ist für die Ausführung der Task gemäß ihren Aktivierungszeitpunkten zuständig. Die Abarbeitung der Dispatcher-Tabelle ist zyklisch organisiert. Ein komplettes Zyklus stellt eine Dispatcher-Runde dar. Falls eine zeitgesteuerte Task sich immer noch im Zustand *running* befindet, obwohl der Aktivierungszeitpunkt der anderen zeitgesteurten Task erreicht ist, wird die erste Task in Zustand *preempted* überführt und bleibt in dem Zustand bis die unterbrechende Task abgeschlossen wird (s. Abbildung 11). So ein Scheduling wird *stack-based scheduling* genannt. Den Tasks werden keine Prioritäten zugewiesen, sondern nur die Aktivierungszeitpunkte definieren den Vorrang von verschiedenen Tasks.
Voraussetzung für die Berechnung eines statischen Schedules ist die Kenntnis der maximal möglichen Laufzeit (worst case execution time, WCET) jeder einzelnen Task. Um diese tatsächlich angeben zu können, wurde in OSEKtime auf einen Wartezustand für Tasks verzichtet, da nicht vorhergesagt werden kann, nach welcher Zeit das erwartete Ereignis eintritt.
Natürlich müssen gegebenenfalls auch ISRs bei dieser Berechnung berücksichtigt werden. Da ihre Anforderung durch Interrupt Requests (IRQs) asynchron zur laufenden Applikation erfolgt, ist hier die Angabe der WCET nicht ausreichend. Zusätzlich muss bekannt sein, mit welcher Häufigkeit jeder IRQ höchstens auftreten kann. Diese wird durch die minimale Zeit zwischen zwei identischen IRQs (minimum interarrival time, MINT) ausgedrückt. Bei bekannter WCET und MINT

kann in der Dispatcher Round genügend Zeit für jede ISR eingeplant werden. Ein Unterschreiten der MINT ist vom System zu verhindern. Für nicht maskierbare ISRs ist dies natürlich nicht möglich, weshalb die MINT entsprechend konservativ zu wählen ist. Abhängig vom Entwurf einer Applikation kann eine Dispatcher-Runde die Zeiten beinhalten, wo keine Tasks oder ISRs laufen. In diesen Zeiten wird eine *idle-Task* der CPU zugewiesen. Diese Task wird vom Betriebsystem zur Verfügung gestellt, hat keinen Eintrag in der Dispatcher-Tabelle sowie zugewiesene Zeitgrenze. Die nicht zeitkritischen Aufgaben können in dieser Task ausgeführt werden.

5 Vereinigung von ereignis- und zeitgesteuerten Konzepten

Während der Masterarbeit soll ein Modellierungskonzept und ein Framework erstellt werden. Die oben beschriebene Modellierungssprache AIRA und deren Komponenten bieten sich als Grundlage an. Dafür muss sowohl der Sprachumfang als auch Laufzeit erweitert werden.

5.1 Modellierung

Für die Zeitsteuerung mussen in einem Zustand die TDMA und die dazugehörigen Tasks konfigurierbar sein.

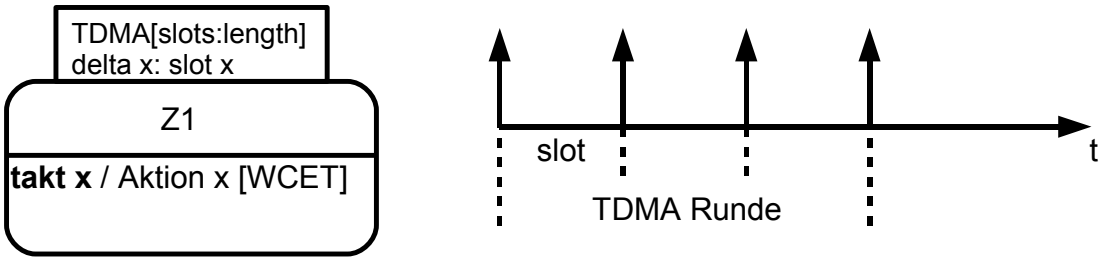


Abb. 8: Modellierung einer TDMA in AIRA

In der Abbildung 8 ist ein möglicher Modellierungskonzept vorgestellt. Debei besteht eine TDMA-Runde aus mehreren *Slots* der gleichen Länge. Jedem Slot kann eine Aktion zugeordnet werden. Um die Ausführbarkeit solchen Zeitplans prüfen zu können, müssen alle Aktivitäten mit WCET's angegeben werden. Aus diesen Angaben wird ein Zeitplan für den statischen Scheduler während der Übersetzung erstellt.

5.2 Modellierung von Hierarchien

Die ersten Komplikationen erscheinen bei der Benutzung von Hierarchien (s. Bils 9). Wenn im Zustand Z1 eine Aktion mit der Periodizität von 1 ms ausgeführt werden soll und im Zustand Z11 eine Aktion mit der Periodizität von 1 ms und WCET von 1,5 ms ausgeführt werden soll, gibt es einen Konflikt: die Aktion 1 kann ihre Ausführungszeitpunkt nicht einhalten. Als mögliche Lösung kann dabei die Vererbung von TDMA-Runden bei den Subautomaten eine Abhilfe schaffen. Anhand der WCET Angaben von Haupt- und Subautomaten kann dann ein Zeitplan geprüft und anschließend erstellt werden.

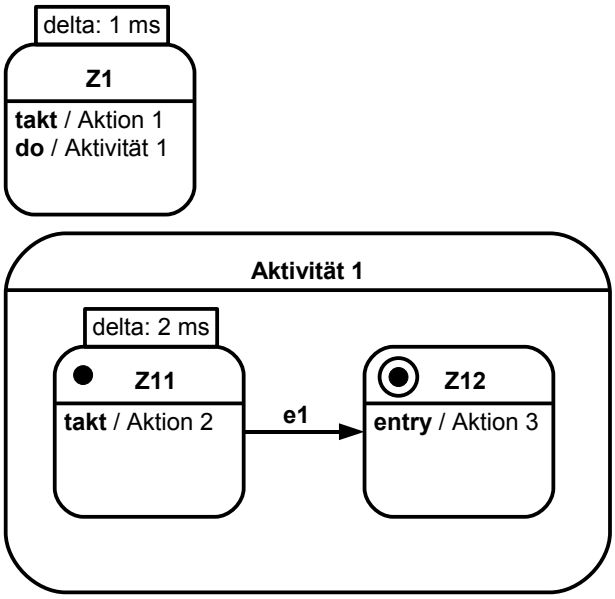


Abb. 9: Hierarchie in TDMA's

5.3 Modellierung von Nebenläufigkeiten

Bei Nebenläufigkeit (s. Abbildung 10) wird die Situation noch komplizierter. Einerseits kann jeder der Kontroller Hierarchie enthalten, anderseits könnten es zwei Kontroller gleichzeitig in einem zeitgesteuerten Zustand befinden,was ein Konflikt zwischen zwei Zeitplänen auslösen würde. Daher müssen in der Entwurfsphase alle Kombinationen geprüft werden, um den gemeinsamen Zeitplan zu garantieren.

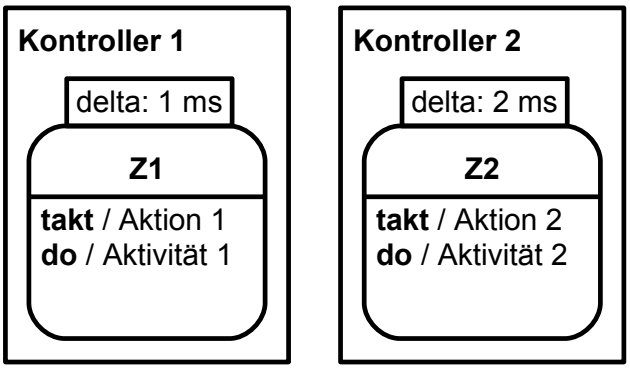


Abb. 10: Nebenläufigkeiten in TDMA's

5.4 Laufzeit

Für die Laufzeit kann OSEKtime-Konzept eingesetzt werden. Dabei kann die *idle-Task* dazu verwendet werden ereignisgesteuerte AIRA-Laufzeit laufen zu lassen. Solange keine zeitgesteuerten Zustände betreten sind, kann ereignisgesteuerte AIRA-Laufzeit laufen wie bisher, sobald ein zeitgesteuerter Zustand betreten ist, wird eine zeitgesteuerte AIRA-Laufzeit eingeschaltet und ereignisgesteuerte AIRA-Laufzeit läuft in *idle-Task*. In der bisherigen Implementierung würde für jeden Kontroller eigene reaktive und zeitgesteuerte Laufzeit geben (s. Abbildung 12a). Um den gemeinesamen Zeitplan garaniteren zu können, müssen alle Kontroller eine zeitgesteuerte Laufzeit, die im Betriebssystem verankert ist, untereinander teilen (s. Abbildung 12b).

6 Zusammenfassung

Die sicherheitskritischen Funktionen wie z.B. X-by-wire in Kfz- und Flugzeugindustrie werden zunehmend auf die eingebetteten Kontroller übertragen. Dies erhöht die Anzahl der installierten ECUs und somit erhöht sich die Anzahl der zu übertragenden Signale. Auf Grund dessen wird nach neuen Möglichkeiten geforscht, ein hybrides System zu entwicklen, dass den gleichen Grad an Fehlerisolation und Fehlereingrenzung von föderierten Systemen erreicht und zugleich die Hardwareressourcen durch Integration von mehreren Anwendungssubsysteme innerhalb eines verteilten Systems reduziert. Um solch ein hybrides System realisieren zu können, bedarf es ein Framework und eine Laufzeit, die das sichere Zusammenleben von den ereignisgesteurten und den zeitgesteuerten Architekturen erlauben. Dabei werden in den ereignisgesteurten Architekturen die Systemaktivitäten, wie Senden einer Nachricht oder Starten einer Berechnung vom Eintreten eines Ereignisses in der Umgebung oder Computersystem angestossen. Dagegen werden die Aktivitäten in den zeitgesteuerten Architekturen durch zyklisch auftretende Zeitmarker „angestoßen“, die von Timern erzeugt werden. An den Beispielen wurden die Eigenschaften von den ereignis- und zeitgesteurten Systemen verdeutlicht.
Aus mehreren Modellierungkonzepten für die ereignisgesteuerten Systeme wurden die hierarchischen Automaten ausgewählt. Einerseits sind die Zustände und Ereignisse *a priori* natürliches Medium, um dynamisches Verhalten von komplexen Systemen zu beschreiben, anderseits lösen die hierarchischen Automaten das Problem des exponentiellen Wachstums an Zuständen durch die Auflockerung der Bedingung alle Zustandskombinationen explizit darzustellen. Eine an der Hochschule für Angewandte Wissenschaften Hamburg entwickelte Sprache AIRA wurde als Vertreter der hierarchischen Automaten näher betrachtet. Neben der Möglichkeit die dynamischen Systeme hierarchisch modellieren zu können, bietet AIRA als Erweiterung eine Zeitsteuerung. Es können **min** (die zustandsändernden Reaktionen werden erst nach min-Zeit freigeschaltet), **max** (eine maximale Verweilzeit in einem Zustand) und **delta** (ein interner Takt) für einen Zustand angegeben werden. Außer das Verhalten des Systems zu definieren beschreibt AIRA auch die Verteilung und Nebenläufigkeit des Systems durch Kontroller (Automaten-Threads) und Interfaces (Kommunikationsschnittstellen der Kontroller zum Benutzer und technischen Prozessen). Als nicht proprietäre, betriebsystemunabhängige Sprache stellt AIRA eine geeignete Basis für das geplante Framework. Ein weiterer Vorteil von dem Framework ist, dass die alten ereignisgesteuerten Anwendungen, die in AIRA modelliert wurden, nicht verändert werden müssen.
Das in OSEKtime implementierte Schedulingverfahren *two-level scheduling* wurde vorgestellt. In dem *two-level scheduling* belegt der erste Scheduler die CPU für die zeitgesteuerten Tasks. Falls keine zeitgesteuerten Tasks ausgeführt werden sollen, wird die CPU dem zweiten Scheduler für die ereignisgesteuerten Tasks übergeben. Des weiteren wurden die Struktur und Komponente des OSEKtime-Betriebssystems vorgestellt.
Ein Vorschlag für die Integration von den ereignis- und zeitgesteuerten Modellierungskonzepten wurde gemacht. Dabei könnte eine TDMA-Runde ähnlich wie **delta** bei einem Zustand angegeben werden. Eine TDMA-Runde wird durch die Anzahl von Zeitslots und deren Länge definiert. Die Aktionen, die beim Eintreffen eines Zeitslots ausgeführt werden, sind mit worst case execution time (WCET) anzugeben. Die Probleme von hierarchisch aufgebauten Zuständen sowie Nebenläufigkeit wurden diskutiert und die Lösungen vorgestellt.
Die zeitgesteuerte Laufzeit soll nach dem OSEKtime-Prinzip aufgebaut werden: die zeitgesteuerten Aktionen werden nach dem während der Modellierung erstellten Zeitplan ausgeführt und die ereignisgesteuerten Aktionen werden in einer *idle-Task* ausgeführt.
In der Masterarbeit soll der hier vorgestellte Ansatz gegen alternative Methoden wie *LTTA* oder *timed automata* geprüft und bewertet werden und daraus resultierendes Konzept soll prototypisch implementiert werden.

7 Abbildungsverzeichnis

8 Abkürzungsverzeichnis

AIRA . .	**A**utomaten **I**n **R**eaktiven **A**nwendungen
ECU . . .	**E**lectronic **C**ontrol **U**nit
ET	**E**vent-**T**riggered
FTCOM	**F**ault **T**olerant **COM**unication
IMA . . .	**I**ntegrated **M**odular **A**vionics
LTTA . .	**L**oosely **T**ime-**T**riggered **A**rchitecture
MAFT .	**M**ulticomputer **A**rchitecture for **F**ault-**T**olerance
MINT . .	**M**inimum **I**nterarrival **T**ime
TT	**T**ime-**T**riggered
WCET .	**W**orst **C**ase **E**xecution **T**ime

9 Literatur

[1] ALBERT, Amos: Comparison of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems. In: *Proceedings Embedded World Conference*, 2004

[2] ALUR, Rajeev ; DILL, David L.: A theory of timed automata. In: *Theor. Comput. Sci.* 126 (1994), Nr. 2, S. 183–235. – ISSN 0304–3975

[3] BENVENISTE, Albert ; CARLONI, Luca P. ; CASPI, Paul ; SANGIOVANNI-VINCETELLI, Alberto L. *Heterogeneous Reactive Systems Modeling and Correct-by-Construction Deployment*. Springer-Verlag Berlin Heidelberg. 2003

[4] BENVENISTE, Albert ; CASPI, Paul ; GUERNIC, Paul le. *A Protocol for Loosely Time-Triggered Architectures*. Springer-Verlag Berlin Heidelberg. 2002

[5] DAVID, Alexandre ; MÖLLER, M. O. *From HUppaal to Uppaal: A translation from hierarchical timed automata to flat timed automata*. Aarhus: BRICS, Dept. of Computer Science, Univ. 2001

[6] GALLA, Thomas M. ; OLIG, Jochen. *Standard Software Components for X-by-Wire Networks*. Proceedings of the Embedded World 2004 Conference, Nürnberg, Germany. 2004

[7] HANSSON, H. A. ; LAWSON, H. W. ; STROMBERG, M. ; LARSSON, S.: BASEMENT: a distributed real-time architecture for vehicle applications. In: *RTAS '95: Proceedings of the Real-Time Technology and Applications Symposium*. Washington, DC, USA : IEEE Computer Society, 1995. – ISBN 0–8186–6980–2, S. 220

[8] HAREL, David. *Statecharts: A Visual Formalism for Complex Systems*. Elsevier Science Publishers B.V. 1987

[9] JANTSCH, Axel: *Modelling Embedded Systems and SOC's*. Morgan Kaufmann Publishers, 2004. – ISBN 1–55860–925–3

[10] KIECKHAFER, Roger M. ; WALTER, Chris J. ; FINN, Alan M. ; THAMBIDURAI, Philip M.: The MAFT Architecture for Distributed Fault Tolerance. In: *IEEE Trans. Comput.* 37 (1988), Nr. 4, S. 398–405. – ISSN 0018–9340

[11] KRČAL, Pavel ; MORKUSHIN, Leonid ; THIAGARAJAN, P.S. ; YI, Wang: Timed vs. Time-Triggered Automata. In: *Proceedings of CONCUR 2004*. London UK, 2004, S. 340–354

[12] METZNER, Alexander ; STIERAND, Ingo. *Analyzing Mixed Event Triggered/Time Triggered Systems*. IEEE RTAS'06. 2006

[13] OBERMAISSER, Roman: *Event-Triggered and Time-Triggered Control Paradigms*. Springer Science, 2005. – ISBN 0–387–23044–0

[14] PONT, Michael J.: *Patterns for Time Triggered Embedded Systems*. Addison Wesley, 2003. – ISBN 1–55860–925–3

[15] POP, Paul ; ELES, Petru ; PENG, Zebo: *Analysis and Synthesis of Distributed Real-Time Embedded Systems*. Kluwer Academic Publishers, 2004. – ISBN 1402028725

[16] SCHOOF, Jochen. *OSEKtime - Standard für zeitgesteuerte Betriebssysteme*. 3SOFT GmbH Erlangen. 2002

A Abbildung: OSEKtime Tasks

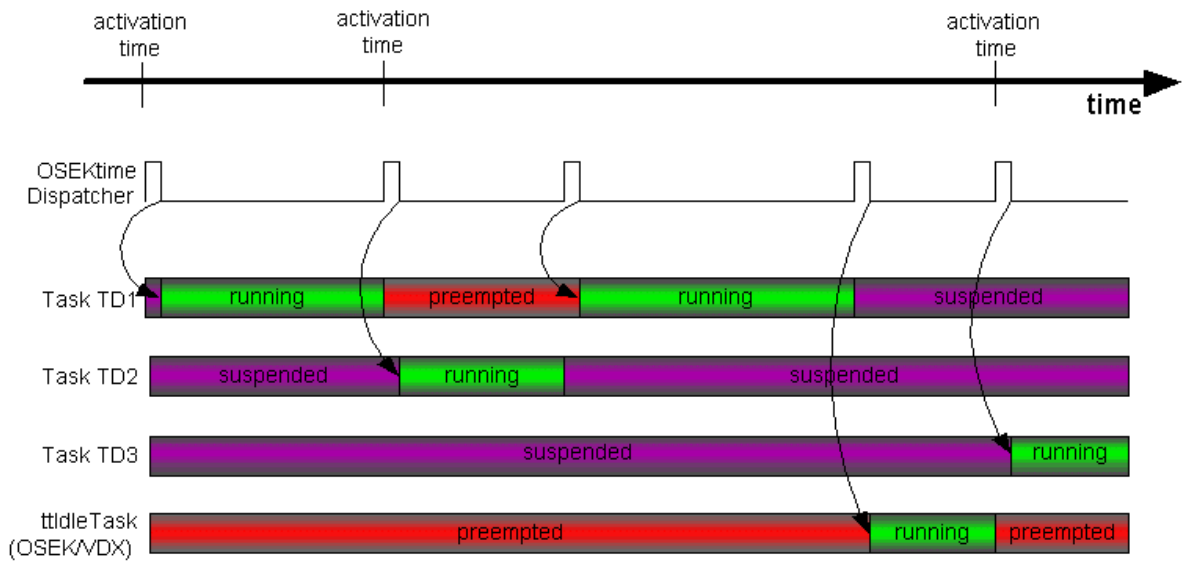
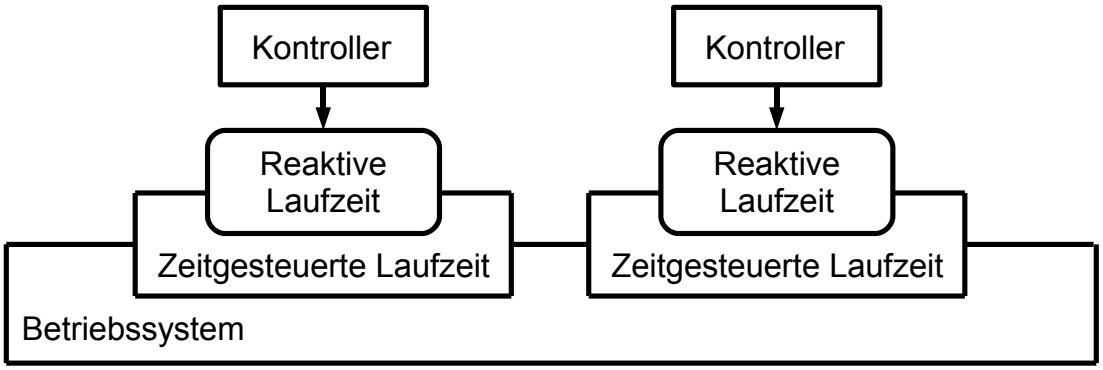
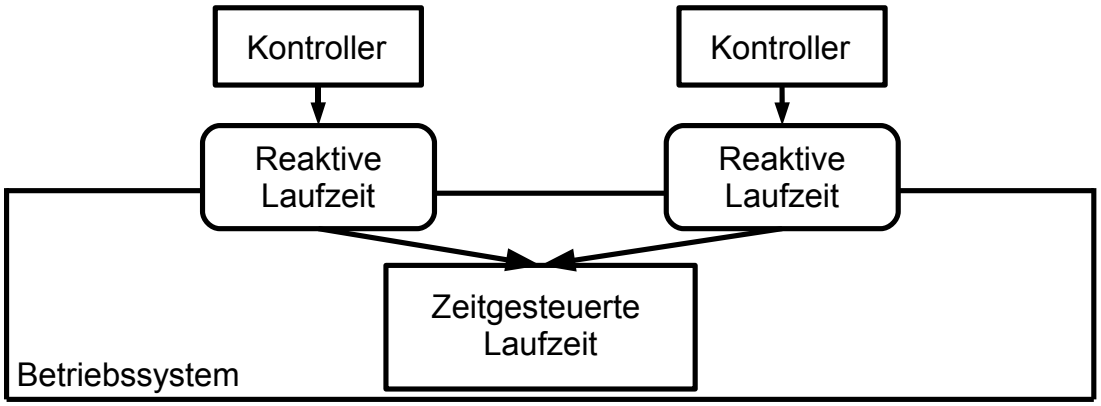


Abb. 11: OSEKtime Tasks

B Abbildung: Erweiterte Laufzeit



(a)



(b)

Abb. 12: Erweiterte Laufzeit