Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung - Anwendungen 2 - SS 12

Jan Kamieth

Scheduling in einem Time-Triggered Ethernet System

Fakultät Technik und Informatik
Department Informatik

Jan Kamieth

Thema der Ausarbeitung - Anwendungen 2 - SS 12
Scheduling in einem Time-Triggered Ethernet System
Stichworte
Echtzeit Ethernet, TTEthernet, Bussysteme, Automotive Anwendungen, Scheduling

Kurzzusammenfassung
Die steigende Komplexität und neuen Anforderungen im Bereich der automobilen Kommunikatinssysteme erfordern neue Konzepte für die interne Vernetzung. Das TT-Ethernet stellt eine Lösung für diese Probleme dar, wodurch aber auch neue Probleme entstehen. Solch ein System unterliegt einem komplexen Scheduling, das durch entsprechende Algorithmen gelöst werden muss. Diese Arbeit stellt einige verwandte Probleme und deren Lösungen vor und zeigt die Gemeinsamkeiten und Unterschiede zum eigenen Problem auf.
Betreuende Prüfer
Prof. Dr. Kai von Luck, Prof. Dr. Gunter Klemke

Betreuer
Prof. Dr. Franz Korf

Title of the paper
Scheduling of a Time-Triggered Ethernet System
Keywords
Real-time Ethernet, TTEthernet, Bussystems, Automotive Applications, Scheduling

Abstract
The increasing complexity and new requirements in the area of automobile communication systems require new concepts for the in-vehicle networks. The TT-Ethernet is such a solution for these problems, by what arises new problems. Such a system is subject to a complex scheduling, which have to be solved by corresponding algorithms. This paper presents some related problems and solutions and reveals the commonalities and differences to the own problem.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Entwicklung des Automobils unterliegt seit Jahrzehnten einem stetigen Wandel, der derzeit mit dem Wechsel in die Elektroautomobilität den momentanen Höhepunkt erreicht hat. Die Anzahl der elektronischen Komponenten war in dieser Zeit stark ansteigend, so dass diese bereits für mehr als 20% der Herstellungskosten eines Fahrzeugs verantwortlich sind und es ist zu erwarten, dass in Zukunft 80% der kommenden Innovationen auf der Weiterentwicklung dieser elektronischen Komponenten beruhen (vgl. Leen und Heffernan (2002)).

Eine besondere Rolle spielen dabei die neuen X-By-Wire Systeme, wie zum Beispiel Steer-By-Wire und Break-By-Wire, die neue Anforderungen an die Bussysteme innerhalb des Automobils stellen, wie Fehlertoleranz, Determinismus, Flexibilität und Sicherheit (vgl. Nolte u. a. (2005)). Hinzu kommen Anforderungen an die Bandbreite des Systems, die für Multimediasysteme und Rückfahrkameras benötigt wird.

Aktuelle Feldbussysteme, wie der CAN-Bus, der MOST-Bus oder der LIN-Bus, werden diesen modernen Anforderungen nicht mehr gerecht, so dass ein neues Kommunikationsprotokoll für die Zukunft gefunden werden muss. Eine Lösung, die diesen Anforderungen gerecht wird, ist das Time-Triggered Ethernet (TTE) Protokoll (vgl. Kopetz u. a. (2005)), dass derzeit von der Firma TTTech entwickelt wird (vgl. Steiner (2008)).

Um der Anforderung des Determinismus gerecht zu werden, setzt das TTE Protokoll auf mehrere Nachrichtenklassen, von denen die Time-Triggered Klasse die größte Rolle spielt, um das Netzwerk Echtzeit fähig zu machen. Durch diese Nachrichten lässt sich festlegen, zu welchem Zeitpunkt die Nachrichten gesendet werden, so dass es nicht, wie beim klassischen Best-Effort Verkehr, zu Verdrängungen kommen kann. Diese Komponente fügt dem System aber auch eine weitere Stufe der Komplexität hinzu, nämlich das nicht mehr nur die Tasks in dem verteilten System einem Scheduling unterliegen, sondern auch die Nachrichten. Dieses auftretende Schedulingproblem gilt es effizient zu lösen.

1.2 Zielsetzung

In der Fachliteratur gibt es eine Vielzahl von Arbeiten, die sich mit dem Scheduling in verteilten Systemen und dem Multiprozessor Scheduling beschäftigen. Die in dieser Ausarbeitung vorgestellten Arbeiten befassen sich mit dem Schedulingproblem in solch verteilten Systemen und bieten unterschiedliche Herangehensweisen für eine Lösung. Dabei werden Systeme mit und ohne Kommunikationsprotokoll betrachtet. Ziel dieser Arbeit ist es, zu untersuchen, inwiefern sich die vorgestellten Lösungen in einen Zusammenhang mit dem eigenen Problem bringen lassen. Um dies zu erreichen wird auch eine klare Abgrenzung zu dem eigenen Problem gezogen, um die Verwendbarkeit der Lösung zu verifizieren und um die Unterschiede aufzuzeigen.

1.3 Inhaltlicher Aufbau der Arbeit

Im folgenden Kapitel werden die ausgewählten drei Arbeiten näher vorgestellt. Dafür werden die Kernkomponenten der Lösung beleuchtet. Im Anschluss zu jeder Arbeit folgt das Ergebnis der Lösung und die Abgrenzung zum eigenen Problem des TTE Schedulings. Im letzen Kapitel erfolgt dann eine Zusammenfassung der Arbeiten und das Fazit.

2 Verwandte Arbeiten

In diesem Kapitel werden verschiedene Ansätze vorgestellt, um das jeweilige Schedulingproblem zu lösen. Es wird gezeigt, wie das Problem des Prozess Schedulings in verteilten Realtime Systemen gelöst werden kann, die über ein Time-Triggered-Protocol kommunizieren, aber auch darüber hinaus, wie ein statisches Real-time Scheduling für FPGAs erstellt werden kann. Die vorgestellten Lösungswege umfassen das Erstellen einer Heuristik, sowie das Lösen mittels Model-Checking. Diese Lösungswege können nur bedingt auf die Probleme angewendet werden, die in einem Real-Time Ethernet System auftreten, zeigen aber Wege auf, wie die Probleme generell angegangen werden können.

2.1 List Scheduling Heuristik

In dem im Jahre 2000 von Petru Eles, Alex Doboli, Paul Pop und Zebo Peng veröffentlichtem Paper (vgl. Eles u. a. (2000)), konzentrieren sich die Veröffentlicher auf ein verteiltes Real-Time System, dass aus einer Menge von programmierbaren Prozessoren und einer Menge von anwendungsspezifischen Hardware-Prozessoren besteht. Den programmierbaren Prozessoren kann jeder beliebige Task zugeordnet werden, von denen dann immer 1 zur Zeit ausgeführt wird. Welcher dieses ist, kann anhand des erstellten Schedulings von den Prozessoren entschieden werden. Die Hardware-Prozessoren haben fest definierte Tasks, die auf ihnen ausgeführt werden, welche aber auch parallel ausgeführt werden können. Als Kommunikationsmedium zwischen den verschiedenen Komponenten kommt ein System zum Einsatz, dass auf dem Time-Triggered Protocol (TTP) (vgl. Kopetz und G. (1997)) basiert. Die Eigenschaften dieses Protokolls werden beim Scheduling beachtet und darauf optimiert, zum Beispiel wird die Größe der Slots angepasst. Für das Scheduling werden die einzelnen auszuführenden Tasks und die zu versendenen Nachrichten betrachtet. Diese Nachrichten werden als Netzwerkprozesse angesehn, die äquivalent zu den anderen Prozessen zeitlich geplant werden. Die Planung erfolgt dabei so, dass eine möglichst geringe Ausführungszeit erreicht wird, um alle Prozesse im System zum Abschluss zu bringen.
Das Scheduling basiert auf einem Taskgraphen, der die verschiedenen Prozesse in Abhängigkeit zueinander bringt und einem List Scheduling Algorithmus, der am Ende eine Scheduling Tabelle erzeugt. Diese Komponenten werden nun näher vorgestellt, um zu demonstrieren, wie solche Heuristiken im generellen Funktionieren.

2.1.1 Taskgraphen

In der vorgestellten Arbeit wird ein Conditional Process Graph (CPG) verwendet, um die Abhängigkeiten und Interaktionen zwischen den Tasks darzustellen. Dieser CPG ist ein gerichteter azyklischer polarer Graph $G(V, E_S, E_C)$. Dabei ist V die Menge aller Prozesse P, die in dem Graphen durch einen Knoten repräsentiert werden. E_S ist die Menge aller einfacher Kanten, bzw. E_C die Menge aller bedingten Kanten. Zusätzlich sind in dem Graphen die Netzwerktasks durch ausgefüllte schwarze Knoten auf den Kanten dargestellt. Die folgende Abbildung 2.1 demonstriert einen solchen Conditional Process Graph.

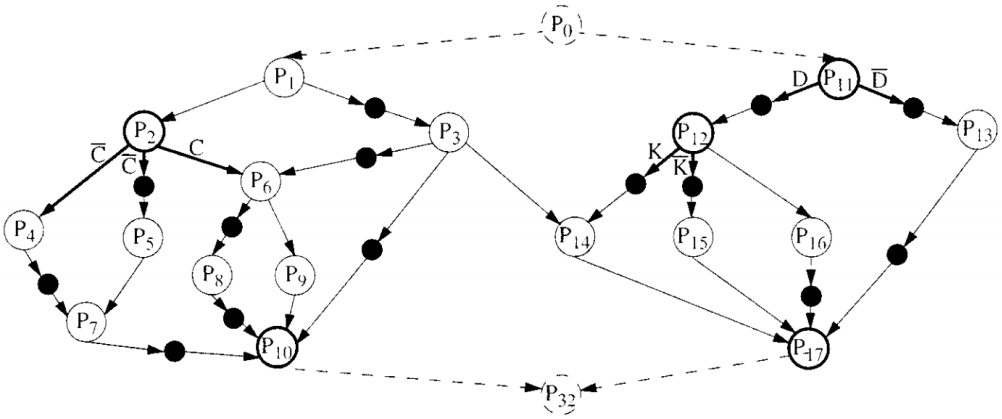

Abbildung 2.1: *Conditional Process Graph*

Der Graph besteht aus den Prozessen P_0 und P_32, die die jeweils die Quelle und Senke des Graphen repräsentieren. Dies sind Dummy Prozesse, um allen Prozessen einen Vorgänger, bzw. Nachfolger zuordnen zu können. Die Knoten P_1 bis P_17 sind die auszuführenden Prozesse, die durch einfache Kanten (E_S), bzw. bedingten Kanten (E_C) miteinander verbunden sind. Von P_1 nach P_2 zeigt zum Beispiel eine einfache Kante, während von P_2 mehrere bedingte Kanten ausgehen. Das heißt, Prozess P_6 wird nur aktiviert, wenn die Bedingung C erfüllt ist. Ansonsten werden P_4 und P_5 aktiviert, so dass auf der Kante von P_2 nach P_5 noch zusätzlich ein Netzwerkprozess ausgeführt werden muss, also eine Nachricht zwischen den beiden Prozessen versendet wird.
Mit solch einem Graphen können nun alle zeitlichen und logischen Abhängigkeiten zwischen den Prozessen abgebildet werden. Diese bilden die Grundlage für die meisten Heuristiken, wie auch für die im nächsten Abschnitt vorgestellte.

2.1.2 List Scheduling Algorithm

Hat man den Taskgraph aus dem vorhergehenden Abschnitt aufgebaut, kann man nun mit einem so genannten List Scheduling Algorithmus ein logisches und zeitlich deterministsches Scheduling erzeugen. Die folgende Abbildung 2.2 zeigt einen solchen Algorithmus in vereinfachter Form, anhand dessen kurz die Funktionsweise dieser Algorithmen erläutert werden soll. So wurde zum Beispiel auf die bedingten Kanten verzichtet, da dies den Rahmen dieses Abschnitts sprengen würde.

```
List_schedule ()
  for each processing element pe_i, i=1,2, ..., N_pe, do
    free_pei=0
  end for;
  schedule P_0 at t=0; initialize ready process list Ls_Ready
  with direct successors of P_0;        -- P_0 is the source node
  while Ls_Ready is not empty do
    p=Head(Ls_Ready);
    if M(p)∈HP then                     -- hardware supports several
       schedule p at t=p.t_ready        -- processes at a time;
    else
       p=Select(Ls_Ready, M(p));
       schedule p at t=max(p.t_ready, free_M(p));
       free_M(p)=t+t_p
    end if;
    delete p from Ls_Ready;
    Ls_Ready = Ls_Ready + processes which become
               ready after p is executed
  end while
end List_schedule;
```

Abbildung 2.2: *Basis List Scheduling Algorithmus*

Der Algorithmus besteht aus einer Menge von Prozessorelementen $pe_i, i = 1,2,\ldots,N_{pe}$, die die Programmierbaren-, Hardware- und Netzwerkprozessoren repräsentieren und die zum Beginn ($t = 0$) der Initialisierung alle frei sind ($free_{pei} = 0$), der Menge aller Prozesse P und einer Liste Ls_{Ready}, die zunächst mit allen Prozessen befüllt wird, die zum Systemstart direkt ausführbar sind, also Nachfolger der Quelle des Prozessgraphen sind.
Nun wird das erste Element aus der Ls_{Ready} Liste entnommen und einem entsprechenden freien Prozessor zugeordnet. Dessen Zeit wird nun aktualisiert ($free_{pei} = t + t_p$), indem die benötigte Ausführungszeit des Prozesses hinzu addiert wird. Nachdem der Prozess ausgeführt wurde, wird er von der Ls_{Ready} Liste entfernt und alle Prozesse, die dadurch aktiviert werden, der Liste hinzugefügt. Dies wird nun so lange wiederholt, bis diese Liste leer ist, also alle Prozesse einem Prozessor zugeordnet wurden und einen Start- und Endzeitpunkt haben. Mit diesen Werten wurde nun ein statisches Scheduling erzeugt, dass in einer Scheduling Tabelle dargestellt werden kann. Diese Tabelle wird nun im folgenden Abschnitt näher vorgestellt.

2.1.3 Schedule Table

Die Abbildung 2.3 zeigt einen Ausschnitt aus einer Scheduling Tabelle, dessen Inhalt mit dem aus Abschnitt 2.1.2 vorgestelltem Algorithmus erzeugt wurde.

	true	D	$D \wedge C$	$D \wedge C \wedge \overline{K}$	$D \wedge C \wedge K$	$D \wedge \overline{C}$	$D \wedge \overline{C} \wedge \overline{K}$	$D \wedge \overline{C} \wedge K$	$\overline{D}$	$\overline{D} \wedge C$	$\overline{D} \wedge \overline{C}$
P_1	0										
P_2	3										
P_3		6							6		
P_{18} (1→3)	3										
P_{19} (2→5)						9					8
P_{20} (3→10)				21	20		21	20		20	20

Abbildung 2.3: *Schedule Table*

Die Zeilen der Tabelle spiegeln die einzelnen Prozesse des Systems wieder und die Spalten die Bedingungen, die auftreten können. Die Werte in den einzelnen Zellen stehen für den Ausführungszeitpunkt. In Zeile 5 erkennt man zum Beispiel, dass der Netzwerkprozess $P_1 9$ unter der erfüllten Bedingung $D \wedge \overline{C}$ zum Zeitpunkt 9 ausgeführt wird und zum Zeitpunkt 8, wenn die Bedingung $\overline{D} \wedge \overline{C}$ erfüllt ist.
Anhand dieser zur Designzeit erstellten Tabelle können die Prozessoren nun entscheiden, welchen Prozess sie zu welcher Zeit ausführen müssen.

2.1.4 Ergebnis und Abgrenzung

Die vorgestellte Arbeit liefert einen guten Einblick, wie Schedulingprobleme mittels Heuristiken gelöst werden können. Die Methodik, dass Problem mittels Taskgraphen und List Schedule Algorithmen anzugehen ist eine generelle Lösungsmöglichkeit und kann auch bei komplexeren Problemen oder Optimierungen angewendet werden (vgl Pop u. a. (2004) und Pop u. a. (1999)).
Dennoch lässt sich diese spezifische Lösung nur bedingt auf das Problem anwenden, welches beim Scheduling in einem Time-Triggered Ethernet System entsteht. Das TTE Protokoll besitzt völlig andere Eigenschaften als das TT-Protokoll. Während TTP auf einen klassischen Feldbus basiert, setzt das TTE auf switched Ethernet, so dass sich zu dem vorgestellten Netzwerkprozessor keine Analogie ziehen lässt. Des weiteren besitzt das TT-Protokoll keinerlei Nachrichtenklassen, so dass diese und ihr Einfluss auf das Scheduling nicht weiter betrachtet wurden. Auch der starke Einfluss der bedingten Kanten, wie sie in dieser Arbeit vorkommen, sind beim TTE Scheduling so nicht zu finden. Das TTE System kann während der Laufzeit keinerlei Schedulingentscheidungen treffen, auch wenn diese vorher statisch berechnet wurden.

2.2 Model Checking - SPIN

Die nächste Arbeit, die von Zonghua Gu, Xiuqiang He und Mingxuan Yuan verfasst wurde (vgl. Gu u. a. (2007b)), versucht das Scheduling auf eine andere Art zu lösen. Statt eine Heuristik zu verwenden, wird auf den Model-Checker SPIN (vgl. Holzmann (1997)) gesetzt. Model-Checking steht für ein Verfahren, bei dem ein vollständig beschriebenes System, dass Modell, gegen eine Formel, die Spezifikation getestet wird (vgl. Palshikar (2004)). Ist das Problem komplett beschrieben, kann der aufgespannte Lösungsraum durchsucht werden, ob er die gestellten Anforderungen einhalten kann, also zum Beispiel ob das System bestimmte Schedulingeigenschaften erfüllt.
Das in diesem Paper betrachtete System basiert wieder auf verteilten programmierbaren Prozessoren, die über ein TTP Bussystem verbunden sind. Die Abhängigkeiten werden wieder über einen Taskgraphen abgebildet. Im Gegensatz zu der Arbeit aus dem vorherigen Abschnitt, wird aber auf Hardware-Prozessoren und bedingte Kanten im Taskgraphen verzichtet. Es handelt sich somit um ein einfacheres Problem.

2.2.1 SPIN

SPIN steht für 'Simple Promela Interpreter' und wurde speziell entwickelt um verteilte Software Modelle zu verifizieren. Das Programm wurde von Gerard J. Holzmann im Jahre 1980 in den Bell Labs geschrieben und wird auch noch heute regelmäßig weiterentwickelt. Beschrieben werden die Modelle für den Model-Checker Spin in der Promela (Process Meta Language) Spezifikationssprache, die synchrone und asynchrone verteilte Algorithmen und Protokolle mittels Automaten beschreibt. Zur Spezifikation der Eigenschaften des Systems wird wiederum die Linear Temporal Logic (LTL) verwendet.
Eine Besonderheit beim SPIN Model-Checker ist, dass dieser nicht der eigentlich Model-Checker ist. SPIN generiert aus dem beschriebenen Problem C-Code, der compiliert werden kann und dann den Model-Checker für das eine spezielle Problem arstellt. Das hat den Vorteil, dass der Model-Checker sehr effizient ist und auch eigener C-Code integriert werden kann.

2.2.2 Promela

In diesem Abschnitt wird ein Ausschnitt aus einem System, dass mit Promela spezifiziert wurde, näher beschrieben und zwar das Modell eines non-preemptive Tasks.

```
proctype Task(byte i) {
{Block until precedence relations are satisfied.}
/*Block until CPU is free.*/
atomic{CPU[Task[i].cpuID].status==FREE->
  Task[i].status=RUNNING;
  Task[i].finTime=time+Task[i].et;
  CPU[Task[i].cpuID].status=BUSY;}
/*Block until time advances to its finish time*/
atomic{time==Task[i].finTime->
  Task[i].status=DONE;
  CPU[Task[i].cpuID].status=FREE;
  {Send messages to remote receiver tasks.}
```

Abbildung 2.4: *Promela Modell eines non-preemptive Tasks*

Das Modell läuft in dem Model-Checker in einem eigenen Prozess ab, der so lange blockiert, bis die CPU, auf die der Task ausgeführt werden soll, frei ist. Wenn der Task ausgeführt werden kann, wird seine Endzeit mittels der Ausführungszeit des Tasks ermittelt und die CPU wird als beschäftigt markiert. Der Prozess blockiert nun wieder so lange, bis der Task ausgeführt wurde, so dass die CPU wieder für weitere Tasks verfügbar wird.
Mit solchen Modellierungen kann der Model-Checker den Zustandsraum eines Systems nach einer optimalen Lösung absuchen.

2.2.3 Ergebnis und Abgrenzung

Der Einsatz eines Model-Checkers ist eine interessante Alternative zum Einsatz einer Heuristik zur Problemlösung beim Scheduling. In der vorgestellten Arbeit wird zudem nachgewiesen, dass das eingesetzte Model-Checking bessere Ergebnisse liefert, als die Heuristik, die im ersten Abschnitt dieses Kapitels vorgestellt wurde. Als zusätzlicher Aspekt wird in dieser Arbeit noch das work- und non work-conserving Scheduling betrachtet, dass heißt Tasks können in ihrer Ausführung verzögert werden. Dabei wird nachgewiesen, dass das non work-conserving Scheduling das Ergebnis noch einmal verbessert.
Der Nachteil am Model-Checking ist, dass es sehr ressourcenintensiv ist, da das System je nach Anzahl der Tasks, Graphenform, Ausführungszeit und Nachrichtengröße beliebig komplex werden kann. Dies führt zu der so genannten Zustandsraumexplosion, d. h., dass der zu durchsuchende Lösungsraum eine Größe erreicht, der nicht mehr effizient durchsucht werden kann. Bei den angegebenen Beispielen wurden schon bei einer geringen Taskanzahl (< 20) mehrere Minuten benötigt, um eine Lösung zu finden. Dies stellt eine nicht zu unterschätzende

Begrenzung des Verfahrens dar.
Hinzu kommt, dass auch bei dieser Arbeit nur das TT-Protokoll betrachtet wurde und nicht das komplexere TTE Protokoll, welches die Komplexität des Systems weiter erhöht.

2.3 Model-Checking - UPPAAL

Die letzte hier vorgestellte Arbeit befasst sich wieder mit der Methodik des Model-Checkings und wurde auch von Zonghua Gu, Mingxuan Yuan und Xiuqiang He verfasst (vgl. Gu u. a. (2007a)), die schon den Model-Checker SPIN zur Lösung von Scheduling Problemen untersucht haben (vgl. 2.2). In diesem Fall wurde aber kein verteiltes System betrachtet, sondern *Partially Runtime-Reconfigurable* (PRTR) FPGAs, also FPGAs die zur Laufzeit neue Tasks ausführen können. Dazu werden Gatter, die nach der Ausführung eines Tasks nicht mehr benötigt werden, mit einem neuen Task beschrieben, der dann zur Ausführung kommt. Das auftretende Schedulingproblem bei diesem Verfahren ergibt sich durch die Tatsache, dass die Tasks alle eine unterschiedliche Menge an Gattern auf dem FPGA belegen. Das Problem ist also sehr ähnlich zu den bisher vorgestellten. Eine Anzahl von Tasks soll möglichst in kurzer Zeit mit beschränkten Ressourcen ausgeführt werden. Ein neuer Aspekt ist, dass auf auf einem FPGA, theoretisch gesehen, eine beliebige Menge an Tasks parallel ausgeführt werden können. Eine Parallelität, wie sie zum Beispiel auch in einem switched Bussystem auftritt. Gelöst wurde dieses Problem mit Hilfe des UPPAAL Model-Checkers.

2.3.1 UPPAAL

Entwickelt wurde der UPPAAL Model-Checker (vgl. uppaal-web) an den Universitäten von Uppsala und Aalborg. Der Unterschied zu anderen Model-Checkern, wie zum Beispiel dem SPIN Model-Checker, ist die Verwendung von *timed automata* an Stelle von normalen endlichen Automaten. Timed Automata fügen den Automaten eine Menge von Uhren hinzu, die im gleichen Takt laufen und jeweils die Zeit seit ihrem letzten Reset anzeigen. Durch diese Erweiterung eignen sich die Automaten für eine bessere Modellierung von Real-Time Systemen. Zusätzlich erhalten die Automaten noch Synchronisationskanäle, um zwischen den Automaten kommunizieren zu können und Integer Variablen, um weitere Zustände etablieren zu können.
Während das Modell des Systems mit den timed Automata beschrieben wird, werden die Eigenschaften, die erfüllt werden müssen, mittels der *timed computational tree logic* (TCTL) beschrieben. Der Pfad innerhalb dieses Baums, der zum final erreichten Zustand führt, entspricht dann der Reihenfolge, in der die Tasks ausgeführt werden müssen, um ein optimales Ergebnis zu erhalten.

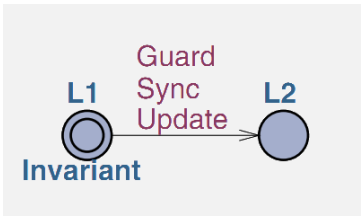

Abbildung 2.5: *Basis Syntax eines UPPAAL Timed Automata*

Die Abbildung 2.5 zeigt die Basis Syntax eines solchen Automaten. Jede Transition besitzt einen Guard, der die Ausführung der Transition schützt, einen Synchronisationskanal, über den Nachrichten versendet oder empfangen werden können und eine Update Funktion, mittels der die Uhr des Automaten zurück gesetzt werden kann oder eine Menge von Variablen modifiziert werden können.

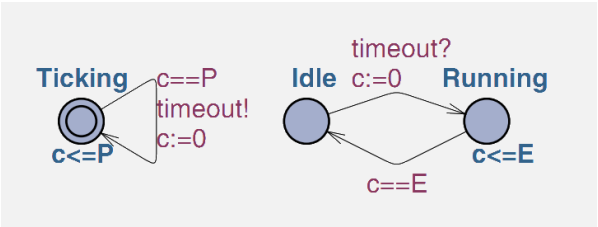

Abbildung 2.6: *Periodisches Task System*

In der Abbilding 2.6 wird ein periodisches Task System demonstriert. Das System besteht aus 2 Automaten, einem Timer auf der linken Seite und einem Task auf der rechten Seite, die jeweils eine lokale Uhr *c* besitzen. *timeout* ist der globale Synchronisationskanal, über den der Timer mehrere Tasks zur Ausführung bringen kann. *P* und *E* sind Variablen, die jeweils der Periode des Timers und der Ausführungszeit des Tasks entsprechen. Diese Modelle sind Templates, die durch den UPPAAL Model-Checker zu den eigentlichen Automaten instantiiert werden, ähnlich wie bei Klassen und Objekten einer Objektorientierten Programmiersprache.
Mit Hilfe dieser Modellierung lässt sich auch das statische Scheduling eines Task Graphen abbilden, auf dessen Lösung aber hier nicht weiter eingegangen wird, da dies den Rahmen der Ausarbeitung sprengen würde.

2.3.2 Ergebnis und Abgrenzung

Wie schon in dem Abschnitt zum SPIN Model-Checker, bietet dieses Verfahren eine interessante Lösungsmöglichkeit für Scheduling Probleme. Die *timed automata* bieten für Real-Time Tasks sehr gute Modellierungsmöglichkeiten, welches in der vorgestellten Arbeit auch durch das statische Scheduling von Task Graphen demonstriert wurde. Zudem wurde ein Blick

auf die Parallelität der Tasks geworfen, die sich auf das parallele Versenden von Nachrichten im TTE-System abbilden lässt.
Nicht betrachtet wurden die Aspekte, die durch ein verteiltes System auftreten. Allen vorran die Tatsache, dass keine Nachrichten über ein Kommunikationssystem ausgetauscht werden, dass zum Beispiel auf dem TT-Protokoll basiert. Es bleibt die Frage, wie ein solches Protokoll mit Hilfe der Automaten modelliert werden kann und vor allen Dingen, wie die Nachrichtentypen eines TTE-Systems mit den *timed automata* abgebildet werden können, da nicht alle Nachrichten zeitgesteuert sind.

3 Zusammenfassung und Fazit

Die vorgestellten Arbeiten bieten einen guten Einblick, in die Problemlösung beim Scheduling. Dabei wurden nicht nur unterschiedliche Systeme untersucht, sondern auch unterschiedliche Lösungsmöglichkeiten präsentiert. In der ersten Arbeit wurden der Taskgraph als eine Kernkomponente etabliert, um das Schedulingproblem anzugehen. Der danach angewandte List Scheduling Algorithmus erwies sich als relativ simple Heuristik, um eine Lösung zu finden und stellt eine Basis dar, um darauf komplexere Probleme zu bearbeiten. Die darauf folgenden Arbeiten über das Model-Checking stellen eine interessante und noch weiter zu verifizierende Alternative zu den Heuristiken dar. Dabei wurde sowohl ein klassischer Model-Checker, wie der SPIN Model-Checker vorgestellt, sondern auch das relativ neuere Konzept der *timed automata* umrissen, die eine andere Form der Modellierung ermöglichen. In dieser letzten Arbeit wurde auch eine Lösung für ein statisches Task Graph Scheduling entworfen, deren Basis eventuell weiter verwendet werden kann.
Auf der anderen Seite stehen die doch teilweise eklatanten Unterschiede zur eigenen Problemstellung, die besonders das verwendete Kommunikationsprotokoll betreffen. Wurde in den ersten beiden Arbeiten noch das TT-Protokoll verwendet, kam in der letzten Arbeit überhaupt kein Nachrichtenprotoll zum Einsatz. Auch das TT-Protokoll ist nur bedingt, mit dem TTE-Protokoll zu vergleichen, da bei diesem Protokoll immer noch die Nachrichtenklassen und das switched Bussystem fehlen.
Trotz dieser Unterschiede wurde ein guter Überblick über verschiedene Lösungsmöglichkeiten geschaffen, auf dessen Basis eigene Lösungen für das Schedulingproblem in einem TTE-System erarbeitet werden können. Um dies weiter zu unterstützen sollten zum Beispiel noch Arbeiten näher betrachtet werden, die sich weiterführend mit dem Task Scheduling beschäftigen, um das Gebiet der Heuristiken tiefer zu durchdringen und so Teile von bereits etablierten Algorithmen verwenden zu können.

Literaturverzeichnis

[uppaal-web] : *The UPPAAL Model-Checker*. – URL `http://www.uppaal.org`

[Eles u. a. 2000] ELES, Petru ; DOBOLI, Alex ; POP, Paul ; PENG, Zebo: Scheduling with bus access optimization for distributed embedded systems. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions* 8 (2000), Nr. 5, S. 472–491

[Gu u. a. 2007a] GU, Zonghua ; HE, Xiuqiang ; YUAN, Mingxuan: Optimal Static Task Scheduling on Reconfigurable Hardware Devices Using Model-Checking. In: *Real Time and Embedded Technology and Applications Symposium* (2007), S. 32 – 44

[Gu u. a. 2007b] GU, Zonghua ; HE, Xiuqiang ; YUAN, Mingxuan: Optimization of Static Task and Bus Access Schedules for Time-Triggered Distributed Embedded Systems with Model-Checking. In: *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE* (2007), S. 294–299

[Holzmann 1997] HOLZMANN, G.J.: The model checker SPIN. In: *Software Engineering, IEEE Transactions* 23 (1997), S. 279–295

[Kopetz u. a. 2005] KOPETZ, H. ; ADEMAJ, A. ; GRILLINGER, P. ; STEINHAMMER, K.: The time-triggered Ethernet (TTE) design. In: *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on*, May 2005, S. 22–33

[Kopetz und G. 1997] KOPETZ, H. ; G., Grünsteidlm: TTP – A protocol for fault-tolerant real-time systems. In: *Computer, IEEE* 27 (1997), Nr. 1, S. 14 – 23

[Leen und Heffernan 2002] LEEN, Gabriel ; HEFFERNAN, Donal: Expanding Automotive Electronc Systems. In: *Computer, IEEE* (2002), S. 88 – 93

[Nolte u. a. 2005] NOLTE, Thomas ; HANSSON, Hans ; LO BELLO, Lucia: Automotive communications-past, current and future. In: *Emerging Technologies and Factory Automation* (2005), S. 985 – 992

[Palshikar 2004] PALSHIKAR, Girish K.: *An introduction to model checking*. 2004. – URL `http://www.eetimes.com/design/embedded/4024929/An-introduction-to-model-checking`

[Pop u. a. 1999] POP, Paul ; ELES, Petru ; PENG, Zebo: An Improved Scheduling Technique for Time-Triggered Embedded Systems. In: *EUROMICRO Conference* 1 (1999), S. 301–310

[Pop u. a. 2004] POP, Paul ; ELES, Petru ; PENG, Zebo ; POP, Traian: Scheduling and Mapping in an Incremental Design Methodology for Dsitributed Real-Time Embedded Systems. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions* 12 (2004), Nr. 8, S. 793–811

[Steiner 2008] STEINER, Wilfried: *TTEthernet Specification*. TTTech Computertechnik AG. Nov 2008. – URL `http://www.tttech.com`