Laufzeitbudgetüberwachung in einer aspektorientierten Betriebssystemfamilie

Studienarbeit im Fach Informatik

vorgelegt von **Frederic Pollmann**

1. Juni 2008 bis 15. Dezember 2008

Lehrstuhl für Verteilte Systeme und Betriebssysteme
Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: Dipl.-Inf. Daniel Lohmann
Dipl.-Inf. Wanja Hofer

Betreuender Professor: Prof. Dr. Wolfgang Schröder-Preikschat

Frederic Pollmann
Matrikelnummer: 2104382
Nürnberger Str. 98
91052 Erlangen

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Zusammenfassung

In der Softwareentwicklung entsteht oft nicht nur ein einzelnes Produkt, sondern gleich eine ganze Produktfamilie mit einer gemeinsamen Basis. Die individuelle Ausprägung der Varianten dieser Produktfamilie erfolgt durch das Hinzufügen oder Weglassen von einzelnen Merkmalen.
Je breiter die Produktfamilie ausgeprägt ist, desto komplexer und fehleranfälliger wird das Verwalten und Hinzufügen einzelner Merkmale, was zusätzliche Kosten und Aufwand bedeutet.
Ein neuer Ansatz zur einfachen und sauberen Trennung einzelner Merkmale in der Entwicklung von Produktfamilien ist die *aspektorientierte Programmierung (AOP)*.
Um zu zeigen, dass dieser Ansatz auch in der Entwicklung eines Betriebssystem verfolgbar ist, wurde ein bestehendes Betriebssystem um ein zusätzliches Merkmal erweitert. Es zeigte sich, dass eine saubere Trennung von Belangen gut machbar ist und aspektorientierte Programmierung auch bei der Entwicklung von Betriebssystemen eingesetzt werden kann.

Inhaltsverzeichnis

Kapitel 1.

Einführung

Im Leben eines Software-Projektes wird oft der Zeitpunkt erreicht, an dem der Entwickler nicht mehr mit nur einer Version alle Bedürfnisse seiner Anwender bedienen kann oder will. Dies führt dazu, dass die Quelltext-Basis divergiert und aus einem einzelnen Produkt eine *Produktlinie* entsteht. Um möglichst einfach die einzelnen Varianten der Produktlinie pflegen zu können, bemüht man sich, die gemeinsame Basis der Varianten möglichst groß zu halten und nur an wirklich entscheidenden Stellen den Quelltext divergieren zu lassen. Dieses Problem ist in der Software-Entwicklung schon lange bekannt, dementsprechend existieren viele Lösungsansätze.

Bei der Entwicklung einer Betriebssystemfamilie für eingebettete Systeme, die typischerweise deutlich weniger Leistung und Speicherplatz bieten als klassische Desktop-Rechner, kommt es besonders auf Größe und Ausführungszeit der einzelnen Varianten an. Die Möglichkeit andere Varianten aus der gemeinsamen Basis zu erzeugen sollte eine einzelne Variante nicht negativ beeinflussen. Daher sollte die Trennung der Varianten nicht erst zur Laufzeit, sondern spätestens bei der Übersetzung des Quelltextes der Variante geschehen.

Ein klassischer Ansatz ist hier der Einsatz sog. *Präprozessor-Direktiven* (wie z.B. `#define`), was allerdings schnell unübersichtlich wird. Zudem gibt es keine zentrale Stelle, an der man Änderungen an einer Variante durchführen kann; vielmehr ist es notwendig, alle entsprechenden Stellen herauszusuchen und einzeln zu editieren. Neuere Ansätze basieren darauf, den Quelltext einer Variante aus einer gemeinsamen Datenbank zu generieren, oder mittels einer Meta-Sprache die Konfiguration aus dem normalen Quelltext herauszulösen.
Ein weiterer Ansatz ist die sog. *Aspektorientierung*. Dieses Konzept sieht vor, die Unterschiede der einzelnen Varianten in *Aspekten* zusammenzufassen, welche dann mittels einer speziellen Spracherweiterung mit dem restlichen Quelltext verschmolzen werden.

Die Anwendbarkeit dieses Konzeptes für die Entwicklung von Betriebssystemen wird in einem Forschungsprojekt des Lehrstuhls für *Verteilte Systeme und Betriebssysteme* des Deparments Informatik an der Universität Erlangen-Nürnberg erforscht. Die Betriebssystemfamilie *CiAO* ist eine hochkonfigurierbare Betriebssystem-Produktline

für eingebettete Systeme und wird unter Berücksichtigung der Prinzipien aspektorientierter Programmierung (AOP) entwickelt. Hauptziel von CiAO ist es, mit Hilfe von AOP die Konfigurierbarkeit grundlegender Architektureigenschaften wie z.B. der verwendeten Kernsynchronisations- oder Speicherschutz-Strategien zu erreichen. Dafür kommt die am Lehrstuhl entwickelte Sprache *AspectC++*, eine AOP-Erweiterung für C++, zum Einsatz.

Um den aspektorierentierten Ansatz zu evaluieren, wurde bereits in mehreren Arbeiten der Kern von CiAO-OS an die Kriterien der AUTOSAR-Spezifikation angepasst, z.B. in Bezug auf den Kern und den Speicherschutz. AUTOSAR [1] (AUTomotive Open System ARchitecture) ist ein Architekturkonzept, welches im Automobilbereich der standardisierten Entwicklung von Hard- und Softwarekomponenten dienen soll.
Aufgrund im Vergleich zur IT-Industrie langer Entwicklungs- und Produktzyklen in der Automobilindustrie sind Hard- und Software, die zu Beginn der Entwicklung eines Fahrzeugs noch aktuell waren, bei seinem Markteintritt oft schon veraltet. Die potentiell (jahre-)lange Lebensdauer führt dazu, dass für Reparaturen veraltete Bauteile auf Lager gehalten oder nachproduziert werden müssen. Dieses Problem will AUTOSAR durch die Definition von Standards lösen. Definierte Schnittstellen und Anforderungen erlauben es Herstellern und Entwicklern aus der KFZ-Industrie Baugruppen und Software eines Fahrzeugs während seines gesamten Lebenszyklus schnell und unkompliziert auszutauschen.

Ziel dieser Arbeit ist es nun zu evaluieren, wie ein weiteres AUTOSAR-Merkmal zur Überwachung von Laufzeitbudgets in die CiAO-Familie integriert werden kann, sowie eine Beispielimplementierung auf der TriCore-Plattform von Infineon zu erstellen. Dabei sollen zum einen in Anlehnung an die Anforderungen von AUTOSAR verschiedene Schutzkriterien implementiert werden; zum anderen soll gezeigt werden, dass eine feingranulare Konfiguration der daraus entstehenden Funktionalität mit Hilfe von AspectC++ möglich ist. Daher kommt neben der rein technischen Funktionalität auch der Konfigurierbarkeit der Schutzmechanismen eine große Bedeutung zu; der Anwender soll am Ende die Möglichkeit haben, verschiedene Schutzmechanismen auszuwählen und an seine spezifischen Anforderungen anpassen zu können.

1.1. Schutzkonzepte in AUTOSAR

AUTOSAR erweitert den bereits existierenden OSEK-Standard [2], und ist insofern abwärtskompatibel, als dass existierende OSEK-Anwendungen mit wenig oder keinen Anpassungen auch in einem AUTOSAR-Betriebssystem verwendet werden können. Eine der Erweiterungen des AUTOSAR-Standards gegenüber OSEK ist ein Schutzkonzept, welches aus den in folgender Tabelle gezeigten Bereichen besteht:

Art des Schutzes	Beschreibung
Dienstschutz (*service protection*)	Überwacht Aufrufe der Schnittstellen zwischen Anwendungen und Betriebssystem, um eine fehlerfreie Bearbeitung sicherzustellen.
Speicherschutz (*memory protection*)	Trennt und überwacht den von Anwendungen und Betriebssystem verwendeteten Speicher.
Zeitschutz (*timing protection*)	Überwacht zur Laufzeit die Einhaltung von in der Entwurfsphase definierten Zeitschranken und verhindert so die Ausbreitung von (Zeit-) Fehlern von einem Teil des Systems auf andere.

Das Thema dieser Arbeit ist die Implementierung des Zeitschutz-Systems in CiAO. Dabei definieren AUTOSAR bzw. AUTOSAR-OS, also der Teil der Spezifikation, der das Betriebssystem beschreibt, das erwartete Verhalten und die Schnittstellen, die Implementierungen anzubieten und zu verwenden haben.

1.2. Bisherige Arbeiten

CiAO-OS entstand im Rahmen eines Forschungsprojektes des Lehrstuhl für Verteilte Systeme und Betriebssystem des Departments Informatik an der Universität Erlangen-Nürnberg.
Die Entwicklung eines an AUTOSAR angepassten Kerns für CiAO-OS war Thema einer Diplomarbeit von Wanja Hofer. Darauf aufbauend entstand eine Studienarbeit von Jochen Streicher, welche die Entwicklung und Implementierung eines Modells zur aspektorientierten Unterbrechungsbehandlung behandelt. Eine weitere Diplomarbeit, ebenfalls von Jochen Streicher, untersucht die aspektorientierte Entwicklung konfigurierbarer Speicherschutzverfahren für CiAO-OS.

Die Umsetzung der *Timing Protection* in CiAO dient nicht nur zur Erweiterung der Funktionalität dieses Betriebssystems, sondern auch als Anwendungsfall für die Erweiterung eines bestehenden Systems mittels eines aspektorientierten Ansatzes. Dafür wird im Folgenden zunächst die zur Erweiterung gehörende Domäne *Timing Protection* näher untersucht, sowie darauf aufbauend die Erweiterung entworfen und schließlich implementiert.

Kapitel 2.

Domänenmodell *Timing Protection*

Als erster Schritt bei der Umsetzung des Merkmals *Timing Protection* für CiAO-OS wird ein Modell der Domäne *Timing Protection* erstellt. Hierfür wird untersucht, welche Anforderungen der AUTOSAR-Standard bezogen auf die Funktionsweise des Merkmals stellt, und welche Auswirkungen dies auf das Zusammenwirken mit dem bestehenden System hat. Im Anschluss an dieses Kapitel wird der Entwurf des neuen Merkmals in Bezug auf CiAO-OS beschrieben.

Zur Verwendung von Fachbegriffen Da die für diese Arbeit maßgebliche Spezifikation auf Englisch verfasst wurde, werden einige Schlüsselbegriffe in dieser Arbeit ebenfalls ohne Übersetzung verwendet werden. Sie werden jedoch für ein besseres Verständnis im Domänenlexikon (Abschnitt 2.3) übersetzt und erläutert.
Werden weitere Fachbegriffe ohne nähere textuelle Erklärung verwendet, so lassen sich diese ebenfalls im Domänenlexikon nachschlagen.

2.1. Domänendefinition

Timing Protection bezeichnet die Fähigkeit eines AUTOSAR-Systems, Anwendungen insofern voneinander zu trennen, dass eine Fehlfunktion einer Anwendung andere Anwendungen nicht negativ bei der Einhaltung ihrer Zeitschranken beeinflußt. Fehler bleiben somit auf ihren jeweiligen Verursacher beschränkt. Ziel ist es, statisch als erfüllbar bewiesene Zeitschranken zur Laufzeit einhalten zu können.

Der Schutz verschiedener Softwarekomponenten voreinander ist eine der Hauptanforderungen [3] von AUTOSAR-OS. Dabei ist *Timing Protection* im AUTOSAR-Standard unter der Kennung BSW11008 [4] aufgeführt. Die Implementierung soll das aspektorientierte Betriebssystem CiAO-OS modular erweitern und so eine feingranulare Konfiguration des Systems ermöglichen.

2.2. Domänenanalyse

Das im AUTOSAR-Standard beschriebene Merkmal *Timing Protection* beschreibt ein *Echtzeitsystem*. Die charakteristische Eigenschaft eines Echtzeitbetriebssystem besteht darin, dass Vorgänge und Ereignisse definitiv nach einer zu bestimmenden Zeitspanne bearbeitet worden sind. Diese Zeitschranken nennt man *Deadlines*.

Es gibt verschiedene Klassifizierungen des Echtzeitbetriebes [5], welche wie folgt beschrieben werden:

Klasse (dt./engl.)	**Beschreibung**
weich / soft	das Ergebnis einer zu einem vorgegebenen Termin nicht geleisteten Arbeit ist weiterhin von Nutzen.
fest / firm	das Ergebnis einer zu einem vorgegebenen Termin nicht geleisteten Arbeit ist wertlos und wird verworfen.
hart / hard	das Versäumnis eines fest vorgegebenen Termins kann eine „Katastrophe“ hervorrufen.

In der Definition werden verschiedene Kriterien genannt, die eine Verletzung von Deadlines nach sich ziehen können. Diese Kriterien werden im Anschluss einer Analyse unterzogen, um Anhaltspunkte für den Entwurf und die Umsetzung des Merkmals *Timing Protection* zu erhalten.

Timing Protection in AUTOSAR

Das Merkmal *Timing Protection* in AUTOSAR basiert auf der Überwachung von Kontrollflüssen und Ereignissen. Ein Kontrollfluss lässt sich auf Quelltext-Ebene abgrenzen als eine Funktion, die zur Laufzeit entweder nach einem bestimmten Schema oder auf ein bestimmtes Ereignis hin (einen sog. *Interrupt-Request (IRQ)*) ausgeführt wird. In AUTOSAR werden diese beiden Arten als *Task* bzw. *Interrupt Service Routine (ISR)* bezeichnet. Weiterhin werden *ISR* in zwei Kategorien aufgeteilt: Ein Kontrollfluss der ersten Kategorie (*ISR1*) darf keine Schnittstellen des Betriebssystemkerns aufrufen; einem Kontrollfluss der zweiten Kategorie (*ISR2*) hingegen ist dies erlaubt. Im Rahmen der *Timing Protection* werden in AUTOSAR die Laufzeiten von Tasks und ISR2 überwacht.

Gegenstand der Überwachung sind Zeitschranken, sog. *Deadlines*, die bei der Ausführung bestimmter Aufgaben nicht überschritten werden dürfen. Allerdings ist ein Kontrollfluss, der seine Deadline nicht einhalten konnte, nicht immer auch gleichzeitig der Kontrollfluss, der für die Deadlineverfehlung verantwortlich ist. Daher sieht AUTOSAR keine direkte Deadline-Überwachung vor, stattdessen wird mit bestimmten Einschränkungen, die in ihrer Gesamtheit den Zeitschutz nach AUTOSAR

ergeben, dafür gesorgt, dass das zur statischen Berechnung der Deadlines verwendete Modell zur Laufzeit nicht verletzt werden kann.

Bei der Definition der Faktoren, die die Einhaltung einer Deadline negativ beeinflußen können, werden in AUTOSAR ISR2 und Tasks beinahe gleich behandelt. Die Faktoren sind in AUTOSAR [6, 7.7.2.1] wie folgt definiert:

- *Inter Arrival Rate*: Die Häufigkeit, mit der Tasks oder ISR2 wiederholt ausgeführt werden dürfen.
- *Execution Time Budget*: Die Ausführungszeit eines Tasks oder ISR2.
- *Locking Time Budget*: Die Zeit, die ein Task oder ISR2 auf die Verwendung eines Betriebsmittels warten müssen, das gerade von einem anderen Task oder Interrupt-Handler exklusiv genutzt wird, sowie die Zeit, in der die Behandlung von Interrupts unterdrückt wird.

Zur Laufzeit werden Deadlines also dann nicht eingehalten, wenn ein Task oder ein Interrupt-Handler zu oft ausgeführt wird, dabei zuviel Rechenzeit verbraucht und/oder dabei zu lange gemeinsame Betriebsmittel oder andere Interrupts blockiert, so dass andere Kontrollflussfäden nicht rechtzeitig ausgeführt werden können.

Integration in CiAO-OS

Da CiAO eine variable Produktfamilie darstellen soll, muss *Timing Protection* als ein konfigurierbares Merkmal integriert werden. Der Konfigurator des Systems soll zudem die Möglichkeit haben, eine feingranulare Auswahl der Features zu treffen, ohne für nicht ausgewählte Merkmale eine „Strafe“ im Sinne von zusätzlicher Rechenzeit oder Speicherplatzverbrauch hinnehmen zu müssen.
Im AUTOSAR-Standard werden die Bereiche der *Timing Protection* als eine Einheit behandelt, es ist also prinzipiell nicht nötig, die einzelnen Überwachungsmöglichkeiten getrennt auswählen zu können. Da jedoch mit dieser Arbeit auch untersucht werden soll, wie gut sich feingranulare, querschneidende Erweiterungen eines bestehenden Systems mit den hier vorgegebenen Mitteln umsetzen lassen, wird neben der reinen Funktionalität bei Entwurf und Implementierung auch der Punkt der overhead-freien Konfiguration berücksichtigt. Dies bedeutet, dass die *Timing Protection* funktional und operational in seine einzelnen Belange auftrennbar sein muss, so dass sich diese einzeln und unabhängig voneinander in das System integrieren lassen.

Fehlerbehandlung

In der AUTOSAR-Spezifikation ist als Fehlerbehandlung nur vorgegeben, dass das System beim Auftreten von Zeitschutz-Fehlern eine Betriebssystemschnittstelle (den *ProtectionHook*) mit jeweils passenden Parametern aufrufen soll; die Funktionalität

dieser Schnittstelle muss jedoch der Konfigurator des Systems bereit stellen. Hierfür werden bei den einzelnen Merkmalen Vorschläge gemacht, wie man den Betrieb des Systems sinnvoll fortsetzen könnte.

2.2.1. Merkmalmodell

Ein Merkmalmodell beschreibt die Variabilität innerhalb einer bestimmten Domäne unter Berücksichtigung eventueller Abhängigkeiten zwischen den einzelnen Merkmalen der Domäne. Im Folgenden wird kurz das Modell der Domäne *Timing Protection* vorgestellt, welches dann als Basis für genauere Erläuterungen dient.

Notation Für die Beschreibung der Variabilität von Merkmalen wird eine von Krysztof Czarnecki und Ulrich W. Eisenecker [7] für Diagramme entwickelte Notation verwendet.
Ein Beispiel dieser Notation zeigt Abbildung 2.1. Dabei zeigt der Kreis über einem Merkmal an, ob es bei Auswahl des übergeordneten Merkmals ebenfalls selektiert werden muss. Wenn der Kreis gefüllt ist, so ist das Merkmal zwingend erforderlich, ansonsten ist es optional. Bögen über den Verbindungslinien mehrerer Kind-Merkmale definieren Merkmalgruppen; auch hier ist wichtig, ob der Bogen ausgefüllt ist oder nicht. Wenn ja, dann dürfen mehrere Merkmale der Gruppe ausgewählt werden, ansonsten nur ein einziges.

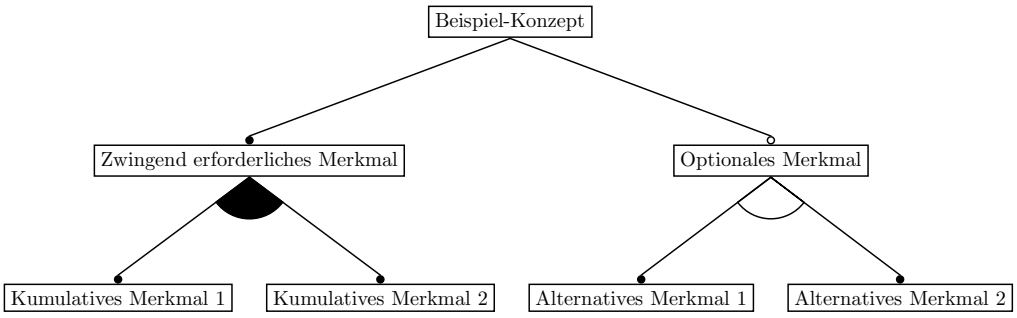

Abbildung 2.1.: Ein Beispiel für ein Merkmaldiagramm

Timing Protection Das Diagramm in Abbildung 2.2 beschreibt die Domäne *Timing Protection*. Diese gliedert sich funktional in drei Bereiche, die voneinander unabhängig sind. Dies sind die *Inter Arrival Rate*, das *Execution Time Budget* und das *Locking Time Budget*. Diese Bereiche überwachen jeweils zwei Kontrollfluss-Abstraktionen, nämlich auf *Tasks* und *ISR2*. Das *Locking Time Budget* überwacht wie lang der jeweilige Kontrollfluss Betriebsmittel für den Zugriff durch andere Kontrollflüsse sperrt. Das Merkmal lässt sich weiter aufteilen auf die Überwachung der Sperrung von *ISR2* und die Sperrung von *gemeinsamen Betriebsmitteln*. Zusätzlich

ist bei Auswahl mindestens eines *Timing Protection*-Merkmals die Selektion des *Deadline Handling* obligatorisch, da dieses Überwachung der jeweiligen Zeitspannen ermöglicht.

2.2.2. Ankunftshäufigkeit (*Inter Arrival Rate*)

Die Ankunftsrate eines Ereignisses ist die Häufigkeit pro Zeit, also die Frequenz, mit der es auftritt. Die *Inter Arrival Rate* kann hierfür als Maß für Interrupts und Tasks festgelegt werden. Bei Interrupts gibt sie an, wieviel Zeit mindestens zwischen einzelnen Interrupts vergehen muss, um zu verhindern, dass Interrupts in einem derart störenden Ausmaß auftreten können, dass nicht mehr genug CPU-Zeit für die Abarbeitung der eigentlichen Aufgaben des System verfügbar ist. Dies ist besonders dann wichtig, wenn die Signale für einen IRQ nicht mehr kontrolliert, sondern willkürlich und häufig auftreten. Die AUTOSAR-Spezifikation spricht hier von einem „babbling idiot“ [6, 7.7.2.1], also einem „brabbelnden Idioten“. Dies kann z.B. durch ein defektes oder falsch konfiguriertes externes Gerät ausgelöst werden. Bei Tasks gibt die *Inter Arrival Rate* die Zeit an, die vor dem wiederholten Ausführen eines Tasks vergehen muss, um eine Monopolisierung der CPU zu vermeiden und anderen Tasks die Möglichkeit zu geben ausgeführt zu werden.

Fehlerbehandlung Beim Überschreiten der *Inter Arrival Rate* einer ISR oder eines Tasks wird die system-interne Fehlerbehandlung gestartet. Außerdem wird der entsprechende Interrupt-Handler bzw. der Task nicht (weiter) ausgeführt, um negative Nebeneffekte zu vermeiden. Um weiteres Fehlverhalten zu vermeiden, könnte es sinnvoll sein, im Protection Hook die entsprechende Interrupt-Quelle (zeitweise) zu deaktivieren bzw. den entsprechenden Task von weiterer Ausführung auszunehmen.

2.2.3. Ausführungszeit (*Execution Time Budget*)

Zur statischen Berechenbarkeit von Zeitschranken ist es nicht ausreichend, nur die Ankunftshäufigkeit von Tasks oder Interrupts zu limitieren. Wichtig ist außerdem, wie groß die maximale Ausführungszeit eines Tasks oder Interrupt-Handlers ist. Um dies festlegen und überwachen zu können, wurde das Merkmal *Execution Time Budget* definiert, welches eine Überwachung der Ausführungszeit des jeweils aktuellen Fadens zulässt. Dabei kann für jeden überwachte Task und Interrupt-Handler das zur Verfügung stehende *Execution Time Budget* einzeln definiert werden.

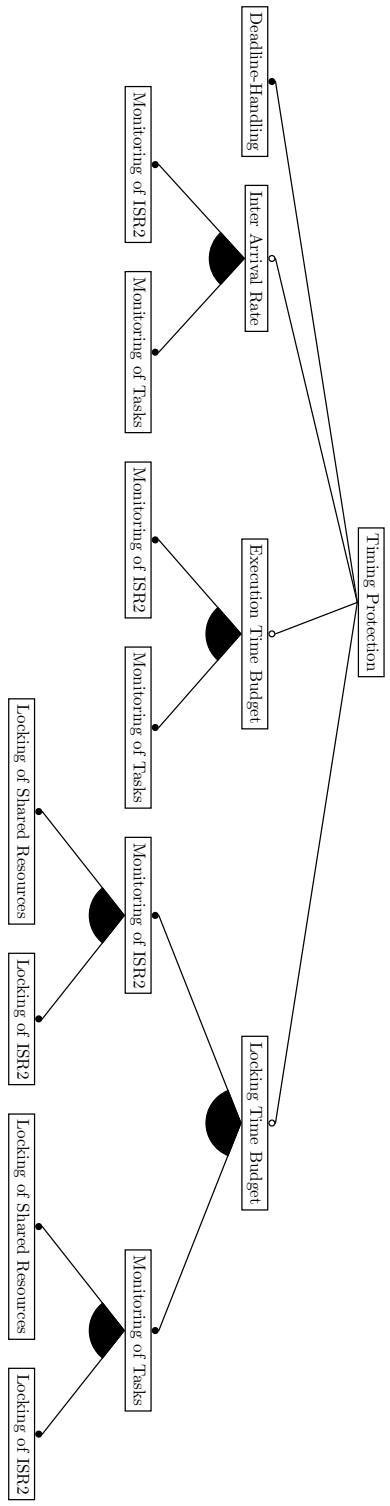

Abbildung 2.2.: Das Merkmalmodell der Domäne Timing Protection

Fehlerbehandlung Beim Überschreiten des *Execution Time Budget* wird die system-interne Fehlerbehandlung gestartet. Hier sollte - wenn möglich - die Ausführung des Tasks gestoppt und ggf. seine Ausführung neu gestartet werden.

2.2.4. Sperrzeiten (*Locking Time Budget*)

Die Kombination von *Execution Time Budget* und *Inter Arrival Rate* lässt nur dann eine statische Analyse der Laufzeiten der einzelnen Bestandteile eines Systems zu, wenn diese ihre einzelnen Beschränkungen auch einhalten können, ohne von anderen Komponenten des Systems benachteiligt zu werden. Dies wäre zum Beispiel der Fall, wenn ein benötigtes gemeinsames Betriebsmittel durch einen anderen Task mit niedrigerer Priorität blockiert wird (*Prioritätsumkehr*). Ein weiteres Beispiel wäre die Deaktivierung von Interrupts, so dass ein Interrupt-Handler gar nicht erst ausgeführt werden kann. Zur Vermeidung solcher Wechselwirkungen existiert das Merkmal *Locking Time Budget*, welches die Zeitspanne überwacht, in der ein Task oder Interrupt-Handler exklusiven Zugriff auf ein gemeinsame Betriebsmittel (*OSEK-Resourcen*) hat oder Unterbrechungsanforderungen sperrt.

Fehlerbehandlung Beim Überschreiten der maximalen Deaktivierungsdauer wird die system-interne Fehlerbehandlung gestartet. Eine Freigabe des gesperrten Betriebsmittels oder der Interrupt-Quelle würde anderen Task und ISR einen reibungslosen weiteren Betrieb gestatten, bringt aber auch ein Problem mit sich. Da kein Mechanismus existiert, den sperrenden Task oder ISR2-Handler von der Aufhebung der Sperre zu unterrichten, sollte zur Vermeidung von Seiteneffekten seine Ausführung gestoppt und er ggf. neu gestartet werden.

2.2.5. Zeitschranken-Verwaltung (*Deadline Handling*)

Um Zeitspannen, genauer gesagt das Ende von Zeitspannen (die *Deadlines*) überwachen zu können, muss die hierfür nötige Infrastruktur bei Verwendung der *Timing Protection* auf jeden Fall mit in das System übernommen werden.

2.3. Domänenlexikon

Im Domänenlexikon werden bisher verwendete Fachbegriffe erläutert und ggf. übersetzt.

Anwendung Eine Anwendung ist ein Software-Modul, welches spezifische Aufgaben eines Systems erledigt, und dabei nicht direkt auf die internen Datenstrukturen des Kernels zugreift, sondern definierte Schnittstellen (sog. System-Calls) des Kernels nutzt. In CiAO-OS besteht eine Anwendung aus einem oder mehreren Tasks und Interrupt-Handlern, welche sich in einer gemeinsamen Schutzdomäne befinden; d.h. ein Fehler in einer Komponente kann nur Auswirkungen auf Komponenten derselben Anwendung haben.

CiAO-OS CiAO: Rekursives Akronym für „CiAO is Aspect-Oriented“. CiAO-OS ist ein Forschungsbetriebssystem, das am Lehrstuhl für Verteilte Systeme und Betriebssysteme an der Friedrich-Alexander-Universität Erlangen-Nürnberg entwickelt wurde.

Deadline Eine Deadline ist ein definierter Zeitpunkt, an dem der Vorgang, auf den sich die Deadline bezieht, abgearbeitet sein muss.

Gemeinsames Betriebsmittel Ein Betriebsmittel, das von mehr als einem Task verwendet wird und dabei exklusiven Zugriff erfordert, d.h. zur Verwendung dieses Betriebsmittels muss dieses erst für den Zugriff durch andere gesperrt werden.

Interrupt-Handler Siehe Unterbrechungsbehandlung.

IRQ Interrupt Request. Siehe Unterbrechungsanforderung.

ISR Interrupt Service Routine. Siehe Unterbrechungsbehandlung.

Task Ein Kontrollfluss mit einer bestimmten Aufgabe in einem Betriebssystem. In CiAO-OS Teil einer Anwendung.

Timing Protection Ein Bestandteil des Schutzkonzeptes in AUTOSAR-OS, nach welchem Anwendungen vor fehlerhaftem zeitlichem Verhalten anderer Anwendungen geschützt werden. Thema dieser Arbeit.

Unterbrechungsanforderung Ein von Software oder externer Hardware generiertes Signal an den Hauptprozessor, den normalen Kontrollfluss zu unterbrechen und eine bestimmte Softwareroutine zur Behandlung des externen Ereignisses zu starten.

Unterbrechungsbehandlung (*Interrupt Service Routine*, *ISR*) Eine Softwareroutine, die zur Behandlung einer Unterbrechungsanforderung aufgerufen wird. Im OSEK-Standard [2] werden zwei verschiedene Kategorien von Unterbrechungsbehandlungen beschrieben, dabei bezeichnet Kategorie 1 (*ISR1*) eine Unterbrechungsbehandlung, die keine Betriebssystemschnittstellen aufruft; Unterbrechungsbehandlungen der Kategorie 2 (*ISR2*) hingegen beinhalten solche Aufrufe und sind mit daher mit dem Kern synchronisiert. Diese Einteilung übernimmt der AUTOSAR-Standard aus OSEK-VDX.

Zeitschutz in AUTOSAR bezieht sich auf nur Unterbrechungsanforderungen der Kategorie 2. In CiAO-OS Teil einer Anwendung.

Zeitschranke Siehe Deadline.

Kapitel 3.

Entwurf

Nach der Analyse im vorhergehenden Kapitel gilt es nun, mit Hilfe der gewonnenen Erkenntnisse eine Architektur für die *Timing Protection* zu entwerfen, die eine einfache und unkomplizierte Umsetzung der Funktionalität in CiAO erlaubt. Dabei soll aufgezeigt werden, wo und wie das Merkmal mit dem bereits bestehenden System interagiert, und wo dieses System erweitert werden muss, um die Umsetzung der *Timing Protection* zu erlauben.

3.1. Timer und Deadlines

Jedes der im Folgenden besprochenen Merkmale benötigt unabhängig von seiner genauen Konfiguration eine Möglichkeit, zur Überwachung der jeweiligen Deadlines zu definierten Zeitpunkten bestimmte Aktionen auszuführen. Diese Deadlines müssen gespeichert und verarbeitet werden; die Verwaltung wird im AUTOSAR-Standard nur implizit erwähnt, ist aber natürlich notwendig, um ein Funktionieren der *Timing Protection* zu ermöglichen und wird daher ebenso in das System integriert werden wie die einzelnen Merkmale der *Timing Protection*.

Ein Problem bei der Verwaltung der Deadline ist es, dass auch bei mehreren in Hardware vorhandenen Timern nicht garantiert werden kann, dass nicht zu irgendeinem Zeitpunkt zur Laufzeit des Systems mehr Deadlines überwacht werden müssen als Timer existieren. Daher muss eine Möglichkeit geschaffen werden, die vorhandene Timer-Hardware mehrfach zu verwenden. Dieses Multiplexen erfordert eine zentrale Verwaltung der oder des Timers. Die Verwaltung der Deadlines sollte daher in einer zentralen Klasse erfolgen, welche die Verwaltung der in Hardware vorhandenen Timer übernimmt und dabei die für die Zeitmessung verwendete Methodik kapselt und den Zugriff auf die Hardware verbirgt. Diese Klasse kann dann je nach eingesetztem System z.B. an die Anzahl der in Hardware vorhandenen Timer angepasst werden, ohne dass Timing-Protection-Merkmale dafür speziell angepasst werden müssten. Abbildung 3.1 zeigt den schematischen Entwurf der Deadline-Verwaltung und das interne Zusammenspiel ihrer einzelnen Bestandteile.

Für die Bearbeitung von Deadlines muss eine Schnittstelle geschaffen werden, die sowohl von der Hardware (also bei Ablaufen des Timers) als auch von der Software verwendet werden kann. Der Grund dafür ist, dass während des Hinzufügens oder Entfernens von Deadlines der Timer nicht gesetzt sein darf, um gleichzeitige Zugriffe und Manipulationen an der internen Deadline-Liste zu verhindern. Manipulationen an der Liste sind ein kritischer Ablauf, der, um die Integrität der Liste zu gewährleisten, nicht unterbrochen werden darf. Daher muss im Anschluss beim Aufziehen des Timers überprüft werden, ob eventuell währenddessen eine Deadline ablief. Die Bearbeitung solcher verpasster Deadlines wird durch Deadline-Verwaltung veranlasst.

Als Schnittstellen müssen mindestens Funktionen zum Hinzufügen einer bevorstehenden Deadline sowie zum Entfernen einer inzwischen ungültigen Deadline existieren. Wenn die Klasse auch die Bearbeitung abgelaufener Deadlines übernehmen soll, muss ihr die Information, wie eine abgelaufene Deadline zu behandeln ist, zusammen mit dem Fälligkeitstermin der Deadline übergeben werden. Um eine ungültige Deadline zu entfernen, muss eine Identifikation der einzelnen Deadlines von außen möglich sein. Am einfachsten ist dies zu erreichen, indem der Aufrufer der Hinzufügen-Methode einen Verweis auf die Deadline erhält und diesen für spätere Verwendung speichern kann.

3.2. Ankunftshäufigkeit

Die *Inter Arrival Rate* definiert eine Zeitspanne, in der ein Task oder ein ISR2 nur jeweils einmal ausgeführt werden darf. Erst wenn diese Zeitspanne verstrichen ist, darf der Task oder die ISR2 wieder ausgeführt werden.

ISR2 Das Ausführen des Interrupt-Handlers eines überwachten Interrupts markiert den Beginn der Zeitspanne, in welcher dieser Interrupt nicht noch einmal auftreten darf. Bei jedem Auftreten einer überwachten ISR2 muss überprüft werden, ob sie sich innerhalb einer solchen Zeitspanne ereignet. Der Interrupt-Handler darf nur dann ausgeführt werden, wenn dies nicht der Fall ist. Die Überwachung einer ISR2 findet in CiAO direkt vor dem eigentlichen Ausführen des Interrupt-Handlers statt. Dort muss überprüft werden, ob der Interrupt-Handler ausgeführt werden darf. Wenn nicht, so muss statt des Interrupt-Handlers die Fehlerbehandlung gestartet werden.

Tasks Der Mechanismus zur Überprüfung der *Inter Arrival Rate* wird für Tasks in den Scheduler integriert. Anhand Abbildung 3.2 wird das Zusammenspiel der Merkmale *Inter Arrival Rate* und *Execution Time Budget* mit den verschiedenen Zuständen und Zustandsübergängen von Tasks illustriert. Wenn ein Task aus dem

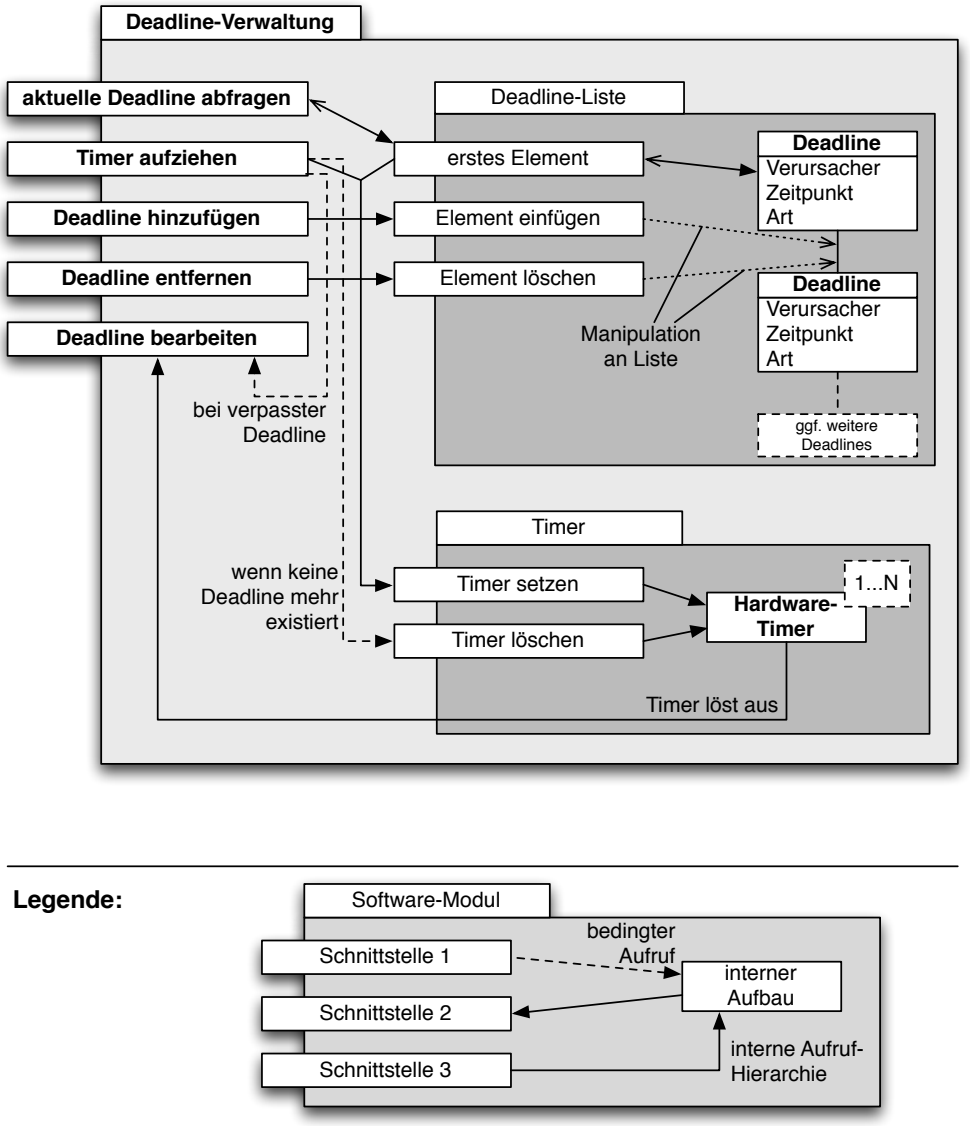

Abbildung 3.1.: Schematischer Entwurf des internen Aufbaus der Deadline-Verwaltung

Zustand *Suspended* oder *Waiting* in den Zustand *Ready* wechselt, so beginnt die Zeitspanne, in welcher er nicht noch einmal aktiviert werden darf. Beim Wechsel eines Tasks in den *Ready*-Zustand muss daher überprüft werden, ob der Wechsel zu diesem Zeitpunkt zulässig ist. Wenn nicht, so darf der Wechsel nicht ausgeführt werden. Sollte derselbe Task innerhalb eines bereits laufenden Timeframes noch einmal in diese Zustände wechseln, so wird das *Inter Arrival Rate* Limit verletzt und die Fehlerbehandlung ausgelöst.

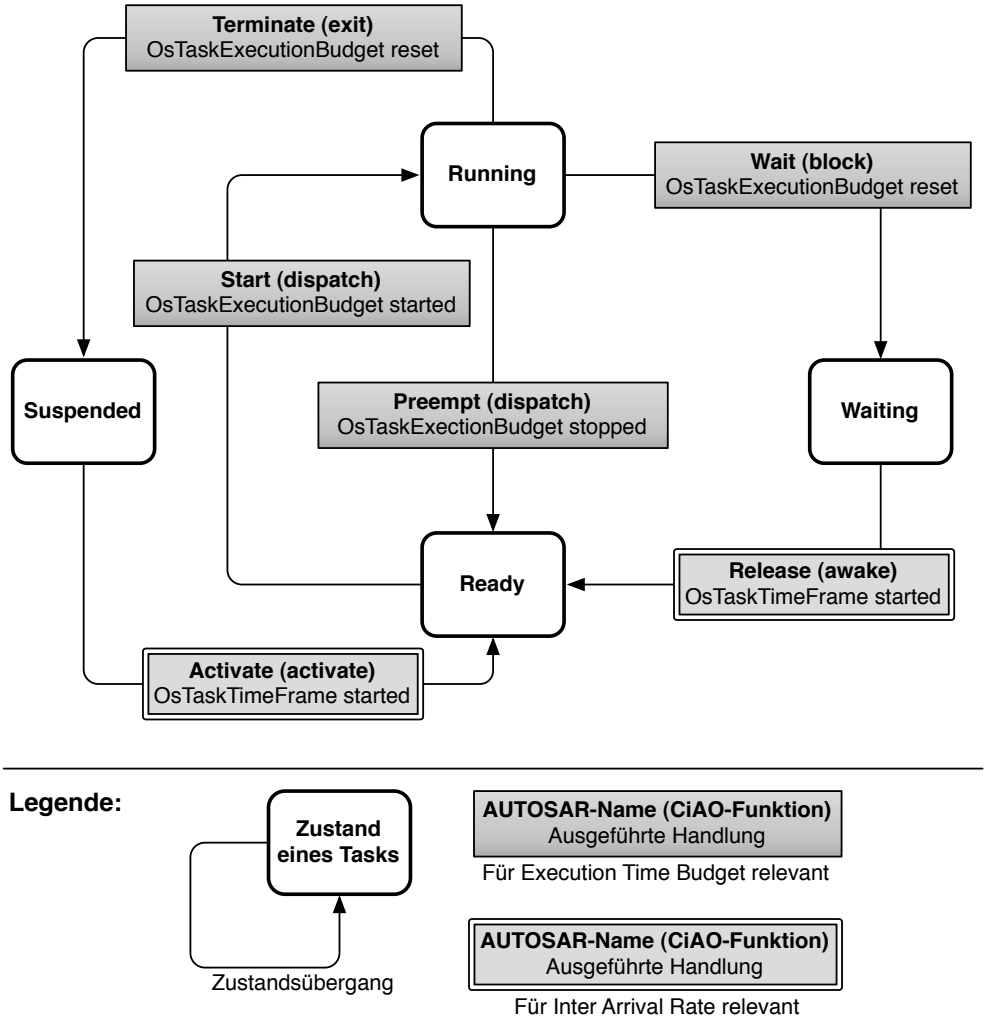

Abbildung 3.2.: Die Merkmale *Inter Arrival Rate* und *Execution Time Budget* im Zusammenspiel mit dem Scheduling von Tasks (nach [6, S. 58])

Die Überprüfung lässt sich jeweils realisieren, indem in der Datenstruktur, die den ISR oder Task identifiziert, eine Variable eingefügt wird, die angibt, ob die *Inter Arrival Rate* zu diesem Zeitpunkt ein erneutes Ausführen zulässt oder nicht. Beim Start des Systems erlaubt diese Variable ein Ausführen; sobald der Task oder Interrupt-

Handler dann tatsächlich ausgeführt wird, wird die Variable so gesetzt, dass eine erneute Ausführung nicht zulässig ist. Durch eine hinzugefügte Deadline wird diese Variable dann nach Verstreichen des entsprechenden Zeitraums wieder auf den ursprünglichen Wert zurückgesetzt.

3.3. Ausführungszeit

Die Laufzeit eines überwachten Interrupt-Handlers darf nicht länger sein als sein konfiguriertes *Execution Time Budget*. Das Budget wird zurückgesetzt, sobald der Interrupt-Handler abgearbeitet ist und der Kontrollfluss wieder vom Betriebssystem übernommen wurde. Sobald ein Interrupt-Handler ausgeführt wird, muss also berechnet werden, wie lange er bei Ausschöpfung seines Budgets maximal ausgeführt werden darf und dieser Zeitpunkt dann als Deadline gesetzt werden. Wenn die Ausführung des Interrupt-Handlers endet, bevor die Deadline erreicht wird, so muss diese wieder entfernt und das verbleibende Budget aktualisiert werden. Sollte während der Ausführung des Interrupt-Handlers jedoch die Deadline erreicht werden, so muss die Ausführung des Interrupt-Handlers unterbrochen und die Ausführung einer Fehlerbehandlungsroutine eingeleitet werden.

Die Ausführungszeit von Tasks wird ab dem Zeitpunkt überwacht, ab dem er gestartet wird. Außer den beiden Zuständen *Suspended* und *Waiting*, bei deren Erreichen das Budget zurückgesetzt wird, muss noch ein Sonderfall beachtet werden:
Wenn ein Task während seiner Ausführung verdrängt wird, so wird sein Budget eingefroren und nicht zurückgesetzt (siehe Abbildung 3.2); bei erneutem Starten des Tasks wird der „eingefrorene“ Wert als neues Budget verwendet.

Aufgrund der in CiAO bereits vorhandenen konzeptuellen Trennung von Tasks und ISR2 kann auch dieses Merkmal getrennt für Tasks und ISR2 implementiert werden.

3.4. Sperrzeiten

Damit statisch berechnete Zeitschranken eingehalten werden können, darf es nicht vorkommen, dass ein Interrupt-Handler oder Task den Zugriff auf gemeinsame Betriebsmittel durch andere Tasks oder Interrupt-Handler länger sperren, als es ihr konfiguriertes *Locking Time Budget* erlaubt. Ebenso gilt, dass ein Interrupt-Handler oder Task nur eine bestimmte Zeit lang die Bearbeitung von Interrupts durch das Betriebssystem deaktivieren dürfen. Um dies sicherzustellen, müssen die entsprechenden Zeiträume durch das Setzen von Deadlines überwacht werden. Zur Verdeutlichung siehe Abbildung 3.3.

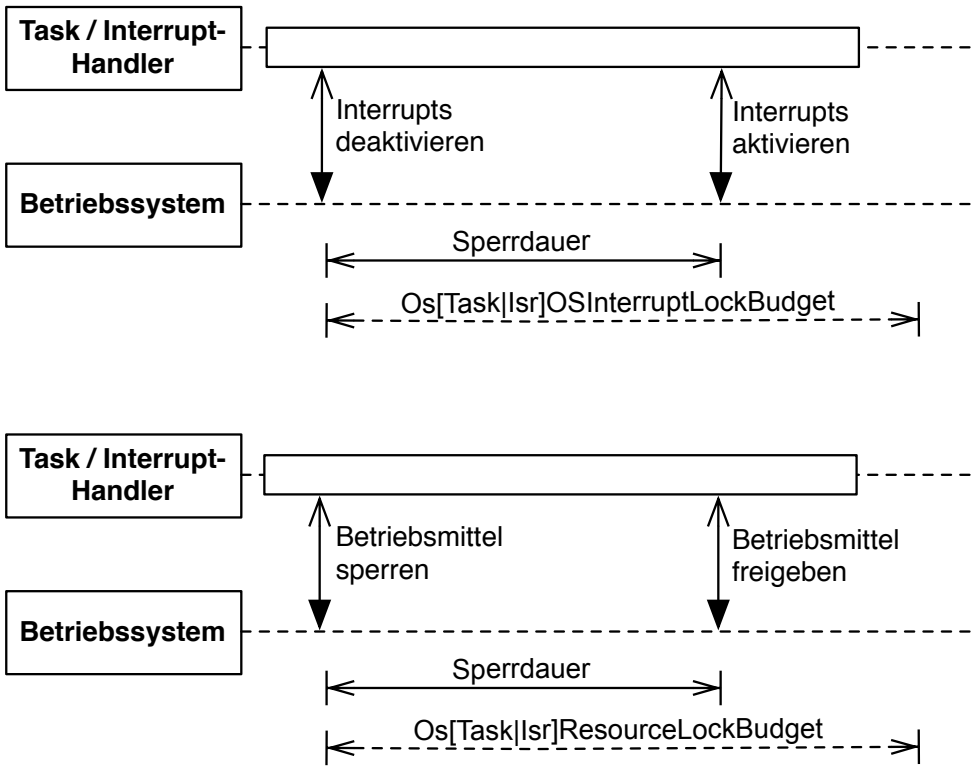

Abbildung 3.3.: Das Merkmal *Locking Time Budget*

Die interessanten Punkte für das Errechnen und Setzen einer Deadline bezogen auf das aktuell verbleibende Budget sind hier das Sperren bzw. wieder Freigeben eines gemeinsamen Betriebsmittels bzw. das De-/Reaktivieren der Interrupts.

Die Berechnungen erfolgen analog zum *Execution Time Budget* (3.3), die Folgen bei Überschreiten des *Locking Time Budgets* müssen jedoch andere sein. Da keine Möglichkeit besteht, dem betreffenden Task oder ISR2-Handler von einer zwischenzeitlich erfolgten zwangsweisen Freigabe der gesperrten Entität in Kenntnis zu setzen, muss dessen Ausführung beendet werden. Je nach Konfiguration könnte es hier möglich sein, nur den Task oder ISR2-Handler oder gleich das ganze System in Folge neu zu starten.

Obwohl die Überwachung der Sperrung gemeinsamer Betriebsmittel und der Deaktivierung von ISR2 in AUTOSAR zusammengefasst wird, können die beiden Bereiche unabhängig voneinander implementiert werden; was tatsächlich zur Anwendung gelangt, kann also dem Konfigurator überlassen bleiben.

3.5. Gemeinsamkeiten von Ausführungszeit- und Sperrzeit-Überwachung

Es fällt auf, dass sowohl für die Überwachung der Ausführungszeit als auch der Sperrzeiten jeweils zu einem bestimmten Punkt, nämlich wenn eine bestimmte Betriebssystem-Schnittstelle verwendet wird, eine Deadline eingerichtet und bei rechtzeitiger Beendigung des überwachten Zeitraums durch die überwachte Entität diese Deadline wieder entfernt muss. Möglicherweise lässt sich dies ausnutzen, um Teile der Implementierung wiederzuverwenden.

Kapitel 4.

Umsetzung

Da die verschiedenen Merkmale, die in ihrer Gesamtheit den Zeitschutz nach AUTOSAR ausmachen, auch einzeln in das fertige CiAO-System integriert werden können, werden ihre Umsetzung hier jeweils einzeln betrachtet.

Vorweg ein Hinweis zum Wertebereich des Timers in CiAO: Abweichend von den AUTOSAR-Anforderungen werden Zeitangaben aller Art in CiAO nur durch einen 32-Bit-Wert angegeben; die Unterstützung für 64-Bit-Werte ist auf der verwendeten TriCore-Zielplattform in den arithmetischen Funktionen der verwendeten Bibliotheken noch nicht produktiv einsetzbar. Bei Verwendung einer 32-Bit-Variable ereignet sich bei einer Zählerfrequenz von 50 MHz nach ca. 86 Sekunden ein Zählerüberlauf; für das Testen der Timing-Protection-Funktionalität war dies ausreichend. Für den produktiven Einsatz empfiehlt es sich, den kompletten Zahlenbereich der 56 Bit des Hardware-Timers zu verwenden — denn dann beträgt die Laufzeit vor einem Zählerüberlauf mehr als 30 Jahre.

4.1. Verwendete Programmiersprachen

Die Implementierung von CiAO-OS sowie die Beschreibung der Konfiguration werden mit verschiedenen Programmiersprachen realisiert, die auch für die Umsetzung der *Timing Protection* Anwendung fanden.

4.1.1. AspectC++

Die Programmiersprache AspectC++ erweitert C++ um einige Konzepte und syntaktische Elemente, mit deren Hilfe Schnittpunkte in C++ definiert werden können, an welchen dann neuer Code eingefügt oder ausgeführt wird.

Das wichtigste Konzept ist der sog. *Aspekt*, mit dessen Hilfe ein an verschiedenen Stellen mit einem System interagierender Belang eines Systems in einem gemeinsamen Rahmen zusammengefasst werden kann. Für diesen Zweck stehen verschiedene Sprachkonstrukte zur Verfügung:

Aspekte beschreiben einen oder mehrere Belange; diese lassen sich i.d.R. nicht mit den Sprachmitteln von C++ in einer Klasse zusammenfassen und werden daher durch einen Aspekt beschrieben. Ähnlich zu Klassen in C++ können Aspekte in AspectC++ auch vererbt werden.

Slices sind Quelltextausschnitte, die Member oder Methoden beinhalten, die in bereits bestehende Klassen und Strukturen eingefügt werden können.

Pointcuts sind mit Hilfe einer speziellen Syntax angegebene Schnittpunkte und -mengen in existerendem C++-Quelltext.

Advice ist die Bezeichnung für zusätzliche Handlungen, die an mittels Pointcuts definierten Stellen im System vorgenommen werden. Sie werden in die bereits bestehende Architektur einfügt und können mit vorhandenem Programmteilen interagieren, sowie die durch Slices eingefügten Erweiterungen nutzen.

4.1.2. XML und XSLT

Während die generellen Fähigkeiten, die CiAO-OS besitzen soll, mit Hilfe eines Programmes zum Variantenmanagement definiert werden, so müssen die jeweiligen Ausprägungen der einzelnen Instanzen dieser Fähigkeiten noch festgelegt werden. Dies geschieht mit Hilfe der auf die Merkmale eines Systems abgestimmten XML-Datei *config.xml*, in welcher Eigenschaften festgelegt werden, die zwischen einzelnen Ausprägungen ein- und desselben Merkmals variieren. Aus dieser XML-Datei werden als erster Teil des Übersetzungsvorgangs, mit welchem aus der Lösung ein Binärbild für die entsprechende Hardwareplattform erzeugt wird, mit Hilfe einer XSLT-Datei anwendungs-spezifische Quelltextdateien erzeugt.

Die für die Verwendung eines Timing Protection-Merkmals notwendigen Attribute werden jeweils in den folgenden Abschnitten erläutert.

Beispiel:
CiAO-OS bietet — neben vielen weiteren Merkmalen — prinzipiell Unterstützung für grundlegende Konzepte und Abstraktionen wie Interrupts, Tasks oder gemeinsame Betriebsmittel ebenso wie für von AUTOSAR vorgesehene Mechanismen wie Speicher- oder Zeitschutz. Ein Konfigurator könnte sich nun entscheiden, dem System zwar Unterstüzung für Interrupts und Tasks ebenso wie Zeitschutz hinzuzufügen, Merkmale wie Speicherschutz oder gemeinsame Betriebsmittel jedoch zu nicht zu integrieren.
Das konfigurierte und transformierte System unterstützt nun zwar Interrupts, jedoch

müssen erst noch Verknüpfungen zwischen den einzelnen Interrupt-Quellen, den zugehörigen Interrupt-Handlern und eventuell weiteren vorhandenen Eigenschaften einer Interruptquelle hergestellt werden. Dies geschieht durch ein XML-Element, welches mit seinen Attributen die Verknüpfung zwischen dem Namen des Gerätes, das den Interrupt auslöst, und der Funktion, die ihn dann bearbeitet, herstellt. Weiterhin werden hier noch die zusätzliche Attribute für den Zeitschutz definiert.

4.2. Deadline-Überwachung

Für die Verwaltung und Überwachung von Deadlines wurde eine eigene Klasse implementiert. Diese bietet einige, im nächsten Abschnitt genauer beschriebene Schnittstellen an. Solange diese Schnittstellen nicht geändert werden, kann die dahinter stehende Programmierlogik ausgetauscht werden, ohne Anwendungen extra an die Änderungen anpassen zu müssen. Dies könnte zum Beispiel hilfreich sein, wenn CiAO auf eine Hardwareplattform portiert werden sollte, die andere Timer anbietet als der Infineon TriCore.

4.2.1. Schnittstellen

Den Überlegungen im Entwurf folgend, ist die als Singleton implementierte zentrale Klasse *TP_Deadlines* mit der in Diagramm 4.1 beschriebenen Signatur für die Verwaltung von Deadlines verantwortlich.

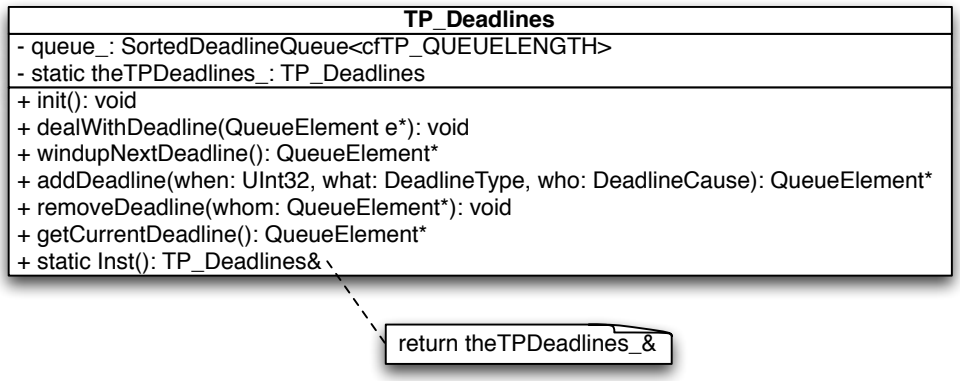

Abbildung 4.1.: UML-Diagramm der Klasse *TP_Deadlines* zur Verwaltung von Deadlines

Die Methode *windupNextDeadline()* erledigt die interne Berechnung, welche Deadline als nächstes ansteht, und stellt den Timer auf den entsprechenden Zeitpunkt.

Um sicherzustellen, dass keine Deadline verpasst wird, muss sie nach jeder Aktion, welche Deadlines einrichtet oder entfernt, ausgeführt werden. Die Methoden *addDeadline* und *removeDeadline* kapseln die Aufrufe an eine interne Klasse, welche die Deadlines verwaltet. Die Methode *getCurrentDeadline* liefert immer die nächste anstehende Deadline zurück; sollte keine Deadline vorhanden sein, so ist der Rückgabewert *null*.
Für die Behandlung einer abgelaufenen Deadline wird der Methodenrumpf *dealWithDeadline* bereit gestellt, welcher über einen Pointcut mittels AspectC++ als Schnittstelle zum Einfügen der konkreten Deadlinebehandlung durch ein Timing-Protection-Merkmal dient. Auf diese Art kann dem System die Behandlung einer für ein Merkmal spezifischen Deadline zusammen mit dem Merkmal hinzugefügt werden.

4.2.2. Interne Datenstrukturen und Algorithmen

Die tatsächliche Verwaltung der Deadlines wird mit Hilfe einer internen Klasse realisiert, welche im Folgenden kurz vorgestellt wird.

Speicherung der Deadlines Die Verwaltung der Deadlines und der damit verknüpften Daten erfolgt mittels zweier doppelt verketteter Listen, die jedoch auf demselben Array operieren und durch die Klasse *SortedDeadlineQueue* (Listing 4.1) implementiert werden. Wie in Listing 4.2 zu sehen, enthält die Klasse *QueueElement*, welche als Container für eine Deadline dient, neben den für die Verkettung der Liste notwendigen Feldern noch drei weitere Felder, in welchen gespeichert wird, wann die Deadline fällig ist (*UInt32 time*), was der Ursprung der Deadline ist (*DeadlineCause who*), und was der Typ der Deadline ist (*DeadlineType type*).
Diese letzte Information ist nötig, da z.B. ein ISR2 sowohl durch das Merkmal *Inter Arrival Rate* als auch durch das Merkmal *Execution Time Budget* Deadlines verursachen kann.

Der für die Listenelemente benötigte Speicher könnte zwar theoretisch auch zur Laufzeit zugewiesen werden, jedoch unterstützte das CiAO-System zum Zeitpunkt der Implementierung der *Timing Protection* keine dynamische Speicherverwaltung: Sämtlicher benötigter Speicher muss also schon zum Zeitpunkt der Übersetzung des System feststehen. Aus diesem Grund muss die Länge der verketteten Liste schon bei der Konfiguration des Systems bestimmt werden. Dieser Wert wird bei der Generierung des Systems als Template-Parameter N verwendet, um die Größe des Arrays der Listenelemente festzulegen.

Initialisierung der Liste Nach dem Systemstart wird mittels eines Aspektes die Funktion *SortedDeadlineQueue<N>::Init()* ausgeführt. Diese verknüpft die in

Listing 4.1: Die Klasse *SortedDeadlineQueue*

```
1  template <unsigned int N>
2  class SortedDeadlineQueue {
3  public:
4    SortedDeadlineQueue() : first(NULL){};
5    void Init();
6    inline QueueElement* getFirst();
7    QueueElement* insert(UInt32 when);
8    void remove(QueueElement* e);
9    void printQueue();
10 private:
11   QueueElement array[N];
12   QueueElement* activeDeadlines;
13   QueueElement* unusedDeadlines;
14   enum {CAPACITY = N};
15 };
```

Listing 4.2: Ein Listenelement zur Deadlineverwaltung

```
1  class QueueElement {
2    friend class SortedDeadlineQueue<cfTP_QUEUELENGTH>;
3  public:
4    QueueElement() : next(0), prev(0) {};
5    DeadlineType type;
6    UInt32 time;
7    DeadlineCause who;
8  private:
9    QueueElement* next;
10   QueueElement* prev;
11 };
```

dem internen Array enthaltenen Listenelemente zu einer doppelt verketteten Liste, welche dann alle unbenutzten Listenelemente enthält. Außerdem wird der Zeiger *QueueElement* unusedDeadlines* auf das erste Element der Liste gesetzt.

Hinzufügen einer Deadline Wenn nun durch den Aufruf der Methode *insert* eine Deadline in die Liste der aktiven Deadlines eingetragen werden soll, wird das erste Element aus der Liste *unused* ausgehängt und an der korrekten Position einer aufsteigend nach dem Wert *UInt32 time* sortierten Liste eingehängt, welche über den Zeiger *QueueElement* activeDeadlines* referenziert wird.
Der Aufwand zum Entnehmen eines nicht benötigten Elements beträgt *O(1)*; durch das iterative Suchen der korrekten Einhängeposition in der zweiten Liste beträgt der Aufwand zum Einfügen der Deadline *O(n)*.

Entfernen einer Deadline Für das bewusste Entfernen einer aktiven Deadline aus der Liste *activeDeadlines* steht die Schnittstelle *removeDeadline* zur Verfügung, welche als Parameter einen Zeiger auf ein Listenelement übernimmt. Das Element wird aus der Liste *activeDeadlines* ausgehängt und als erstes Element in die Liste *unusedDeadlines* eingefügt.
Wenn eine Deadline abläuft, so wird im Zuge der Deadline-Behandlung das entsprechende Listenelement automatisch entfernt. Dies ist die einzige Möglichkeit einer selbsttätigen Entfernung einer Deadline durch das System selbst.

Setzen des Timers Nach jeder Änderung der Deadlinequeue durch Hinzufügen oder Entfernen einer Deadline muss für die jeweils aktuelle Deadline der Timer neu aufgesetzt werden. Dies erledigt die Funktion *windupNextDeadline*, welche durch die einzelnen Timing-Protection-Merkmale aufgerufen wird. Das Aufrufen nach Ablauf einer Deadline erledigt der in Listing 4.3 gezeigte Aspekt *os_tp_DeadlineHandling*. Der Interrupt-Handler, welcher über den Pointcut `TimerIRQ()` (Zeile 2) referenziert wird, ist zunächst nur ein leerer Funktionsrumpf, welcher als Anknüpfungspunkt für Aspekte dient. Mit einem Order-Advice (Zeile 8) wird sichergestellt, dass der Execution-Advice (Zeile 4) tatsächlich immer als letzter Bestandteil der Timer-Interrupt-Behandlung ausgeführt wird, auch wenn andere Aspekte, wie z.B. die des Merkmals *Inter Arrival Rate*, ebenfalls an diese Funktion binden.

4.3. Merkmal *Inter Arrival Rate*

Im Folgenden werden die für die Anwendung des Merkmals *Inter Arrival Rate* notwendige Konfiguration in CiAO und seine Implementierung vorgestellt.

Listing 4.3: Aspekt für den Timer-Interrupt mit Order-Advice

```
1  aspect os_tp_DeadlineHandling {
2  pointcut TimerIRQ() = "% hw::irq::STM_SRC1::Handler(...)";
3
4  advice execution(TimerIRQ()) : after {
5    os::tp::TimingProtectionData::Inst().windupNextDeadline();
6  }
7
8  advice execution(TimerIRQ()) : order \
9    (!"os_tp_DeadlineHandling", "os_tp_DeadlineHandling");
10 };
```

4.3.1. Konfiguration in CiAO

Für die *Inter Arrival Rate*-Überwachung wurde für die Elemente *ISR* und *Task* ein neues Attribut *timeframe* eingeführt, welches die Länge des überwachten Zeitraums zwischen dem einzelnen Auftreten des überwachten Merkmals angibt. Durch Angeben oder Weglassen dieses neuen Attributes ist es möglich, die Überwachung der Ankunftsrate nur auf ausgewählte Interrupts zu beziehen. Die Einheit für hier angegebene Zeitspannen sind Mikrosekunden.

Aus der Gesamtheit der überwachten ISR wird durch das XSL-Skript automatisch ein *Pointcut* generiert, welcher die Handler aller überwachten ISR umfasst. Ein Beispiel ist in Listing 4.4 zu sehen. Wenn in der XML-Konfiguration des Systems für eine Interrupt-Quelle kein Zeitraum für die *Inter Arrival Rate* angegeben wird, so wird der entsprechende Interrupt-Handler nicht in diesem Pointcut erscheinen und somit auch nicht überwacht werden. Dies gibt dem Konfigurator die Möglichkeit, nur bestimmte Interrupt-Quellen überwachen zu lassen.

Listing 4.4: Der automatische erzeugte Pointcut aller überwachten ISR2

```
1 pointcut pcISR2FunctionsWithArrivalRate() = ""
2 || "% hw::irq::DMA_SYSSRC2::Handler(...)"
3 || "% hw::irq::DMA_SYSSRC3::Handler(...)";
```

4.3.2. Implementierung

Der in der XML-Datei angegebene Wert wird, umgerechnet auf die Zeiteinheiten der Zielplattform, in einer durch ein Slice eingefügten Variable gespeichert. Zusätzlich wird noch eine Bool'sche Variable eingefügt, welche anzeigt, ob ein erneutes Ausführen des ISR-Handlers oder Tasks erlaubt ist oder nicht. Dieses Slice kann sowohl für ISR als auch Tasks verwendet werden.

Wenn die Bool'sche Variable beim Überprüfen den Wert *true* liefert, so wird sie auf *false* gesetzt, sowie eine Deadline eingerichtet, die beim Ablaufen die Variable wieder auf *true* setzt. Wenn die Variable den Wert *false* besitzt, wurde innerhalb der Task/Interrupt-Handler bereits innerhalb seines Zeitrahmendes ausgeführt; erneutes Starten würde gegen die *Inter Arrival Rate* verstoßen, so dass stattdessen der Protection-Hook aufgerufen wird.

Für das Überprüfen und Aktualisieren dieser Variablen existieren zwei Advice-Blöcke. Zwar wird die Überprüfung für Tasks und ISR2 prinzipiell gleich durchgeführt, allerdings muss dabei auf die in die internen Datenstrukturen des Interrupt-Handlers bzw. Tasks eingefügten Variablen zugegriffen werden. Eine Vereinheitlichung dieses Zugriffs hätte mehr Overhead verursacht als beseitigt, so dass keine Zusammenführung der entsprechenden Advice-Blöcke vorgenommen wurde.

Pointcuts für ISR Für die Überprüfung kann hier der aus der XML-Konfiguration generierte Pointcut *pcISR2FunctionsWithArrivalRate* (siehe Listing 4.4 verwendet werden. Durch Order-Advice wird sichergestellt, dass diese Überprüfung immer vor Ausführen des eigentlichen ISR-Handlers, der ebenfalls durch Advice eingebunden wird, stattfindet.

Pointcuts für Tasks Die Überwachung der *Inter Arrival Rate* für Tasks muss immer dann erfolgen, wenn ein Task zur Ausführung gelangt. Die einfachste Methode dies zu erreichen, besteht darin, den bereits von CiAO zur Verfügung gestellten Pointcut *asInternalPreTaskHook* zu verwenden.

4.4. Merkmal *Execution Time Budget*

Im Folgenden werden die für die Anwendung des Merkmals *Execution Time Budget* notwendige Konfiguration in CiAO und seine Implementierung vorgestellt.

4.4.1. Konfiguration in CiAO

Sowohl für Task- als auch für ISR-Elemente wurde ein neues Attribut *executionbudget* eingeführt, welches die maximale Zeitspanne in Mikrosekunden beschreibt, die ein Task oder ISR2-Handler am Stück ausgeführt werden darf. Aus der Gesamtheit der überwachten ISR wird durch das XSL-Skript automatisch ein *Pointcut* generiert (vgl. den Abschnitt zum Merkmal *Inter Arrival Rate*, 4.3.1).

4.4.2. Implementierung

Für die Überwachung des *Execution Time Budgets* werden bei ISR und Tasks zwei neue Attribute eingeführt: eines, in dem der in der Konfiguration angegebene Wert für das Budget gespeichert wird, und eines, das auf eine ggf. eingerichtete Deadline für den jeweiligen Interrupt-Handler oder Task verweist.
Bedingt durch die Möglichkeit, dass ein Task verdrängt wird, er aber noch nicht sein gesamtes Budget verbraucht hat (siehe Abbildung 3.2), muss eine Möglichkeit geschaffen werden, ein Restbudget zu speichern. Zu diesem Zweck wird eine zusätzliche Variable in die Task-Verwaltungsstruktur eingeführt.

Behandlung von Tasks Tasks steht möglicherweise nicht bei jedem Ausführen durch den Scheduler ihr volles Budget zur Verfügung, da beim Verdrängen eines Tasks dessen Budget gespeichert und beim nächsten Start wieder für die Berechnung der Deadline herangezogen wird. Um dies zu erreichen, wird mittels des von CiAO bereit gestellten Pointcuts *asInternalPreTaskHook* vor dem Ausführen eines Tasks das aktuell verfügbare Budget als Deadline gesetzt. Wenn durch aufgrund Scheduling oder Erreichen des Ende der Task-Funktion ein Tasks nicht weiter ausgeführt wird, wird mittels des Pointcuts *asInternalPostTaskHook* das verbleibende Budget des Tasks gespeichert sowie die entsprechede Deadline entfernt. Sollte die zu einem Task gehörige *Execution Time Budget*-Deadline ablaufen, so wird durch die Deadline-Behandlung der *ProtectionHook* aufgerufen.

Behandlung von ISR Die Implementierung bei ISR ist einfacher, denn da ein ISR2-Handler regulär nicht unterbrochen werden kann, muss auch kein eventuelles Restbudget gespeichert werden[1]. Beim Start eines Interrupt-Handlers wird eine Deadline eingerichtet, die nach Ablauf des Budgets ausgelöst wird. Sollte der Interrupt-Handler seine Arbeit vorher erledigt haben, so wird diese Deadline wieder entfernt; andernfalls löst der Timer-Interrupt aus, der Interrupt-Handler wird unterbrochen und der *ProtectionHook* wird ausgeführt. Sowohl für Aktivierung als auch für Deaktivierung wird an den generierten Pointcut `pcISR2FunctionsWithETB()` gebunden, allerdings für die Aktivierung mit dem Schlüsselwort *before* und für die Deaktivierung mit dem Schlüsselwort *after*.

[1] Dies ist nicht ganz korrekt: Sowohl Tasks als auch ISR2-Handler können von ISR1 unterbrochen werden. Da diese sich jedoch außerhalb des Timing-Protection-Konzeptes von AUTOSAR befinden und *per definitonem* keine Betriebssystem-Schnittstellen aufrufen dürfen, können ihre Laufzeiten nicht überwacht werden, so dass es nicht möglich ist, die Laufzeit eines ISR1-Handlers von der Laufzeit des unterbrochenen Tasks oder ISR2-Handlers zu trennen.
Für das Verpassen einer Deadline kann daher auch ein zu lange laufender ISR1-Handler verantwortlich sein. Dieses Thema wird von der AUTOSAR-Spezifikation jedoch nicht abgedeckt.

4.5. Merkmal *Locking Time Budget*

Wie schon im Entwurfs-Kapitel diskutiert, kann die Überwachung der Sperrzeit von ISR und Ressourcen getrennt voneinander implementiert werden. Ebenso kann die Überwachung getrennt für Interrupt-Handler und Tasks aktiviert werden, so dass im Endeffekt vier getrennte, aber kombinierbare Möglichkeiten existieren, das *Locking Time Budget* zu überwachen.

Die Möglichkeiten sind:

- Interrupt-Handler deaktivieren Interrupts
- Interrupt-Handler sperren Ressourcen
- Tasks deaktivieren Interrupts
- Tasks sperren Ressourcen

4.5.1. Konfiguration in CiAO

Diese verschiedenen Möglichkeiten können durch die Einführung zweier Attribute *isrlockbudget* und *resourcelockbudget* für die Elemente *task* und *isr* konfiguriert werden.

4.5.2. Implementierung

Um die einzelnen Ausprägungen der *Locking Time Budget*-Überwachung getrennt konfigurierbar zu halten, wurden sie mit getrennten Aspekten implementiert. Trotz großer Ähnlichkeit im Vorgehen konnten die Aspekte nicht zusammengefaßt werden, da sich die jeweils aufzurufenden Betriebssystemschnittstellen unterscheiden und eine Fallunterscheidung in einem hypothetischen gemeinsamen Aspekt die Vorteile des Zusammenfassens wieder konterkariert hätte.

Ansonsten funktionieren die Algorithmen ähnlich wie die Überwachung des *Execution Time Budgets* für ISR, so dass hier auf eine eingehendere Betrachtung verzichtet wird.

Kapitel 5.

Evaluation

Zur Überprüfung, ob und wie gut die Designziele dieser Arbeit erreicht werden konnten, werden Messungen an einem CiAO-System in einer Basis-Variante und mit den verschiedenen Timing-Protection-Merkmalen vorgenommen. Um aufzuzeigen, welchen Einfluß eine feingranulare Konfiguration auf die Größe des übersetzten Betriebssystems hat, werden mehrere Konfigurationen übersetzt und die Größenveränderungen in den Segmenten der Binardatei vergleichen.

Abschließend erfolgt eine kritische Diskussion der eingesetzen Werkzeuge sowie der eigenen Vorgehensweise.

Beschreibung des Mess-Systems Als Basis für alle Messungen wurde eine Applikation entworfen, welche alle potentiell von Timing-Protection-Merkmalen betroffenen Merkmale von CiAO verwendet. Die Applikation besteht aus zwei Tasks, welche abwechselnd aktiviert werden und periodisch gemeinsame Betriebsmittel sperren und wieder freigeben. ISR2 werden mittels eines Software-Mechanismus ausgelöst, um einen Einfluß externer Faktoren auf die Messungen zu eliminieren. Das zugrundeliegende CiAO unterstützt nur die für diese Applikation notwendigen Merkmale.

Die genaue Konfiguration des CiAO-Kerns wird im Anhang in Anhang A gezeigt, ein Listing der Applikation befindet sich in Anhang B.

5.1. Laufzeit-Messungen

Mittels der Laufzeit-Messungen kann untersucht werden, welchen Einfluß die zusätzlichen Merkmale einzeln und kombiniert auf die Ausführungszeiten bestimmter Vorgänge im Betriebssystem haben. Untersucht wurden hier die Dauer vom Auslösen eines ISR2 bis zum Sprung in seine entsprechende Behandlung, die Dauer eines freiwilligen Wechsels von einem Task zum anderen, sowie die Dauer des Sperrvorgangs eines gemeinsamen Betriebsmittels. Die ersten beiden Vorgänge sind von den

Merkmalen *Inter Arrival Rate* (*IAR*) und *Execution Time Budget* (*ETB*) betroffen, das Sperren von Betriebsmitteln wird durch das Merkmal *Locking Time Budget* (*LTB*) beeinflußt. Zusätzlich wurden noch alle Merkmale zusammen aktiviert und gemessen, um zu untersuchen, wie stark die Ausführungszeit bei Verwendung verschiedener Merkmale anwächst.

Methodik Die Laufzeit-Messungen wurden mit dem Programm *Trace32* unter Verwendung eines Lauterbach-Debuggers mit Tracingmodul erstellt. Dabei wurden jeweils 4096 Samples genommen. Ein Sample besteht dabei aus der Ausführungszeit des jeweils beobachteten Vorgangs, welche in Mikrosekunden angegeben wird. Die folgenden Diagramme geben den Mittelwert der Ausführungszeiten sowie die Standard-Abweichung an.

Ausführungszeit eines ISR2-Handlers Die Ergebnisse dieser Messung sind in Abbildung 5.1 dargestellt. Erwärtungsgemäß ist die Ausführungszeit eines ISR2-Handlers am geringsten, wenn keines der zusätzlichen Merkmale aktiv ist. Auch bei Verwendung des Merkmals *LTB*, welches keine Einfluß auf ISR2 hat, ist keine signifikante Verlängerung der Ausführungszeiten erkennbar.
Bei Aktivierung der Merkmale *IAR* oder *ETB* hingegen steigt die Ausführungszeit des Interrupt-Handlers auf mehr als das Vierfache an. Dies ist damit zu erklären, dass nun erst geprüft werden muss, ob die Ausführung des ISR2-Handlers erlaubt ist bzw. eine Deadline zur Überwachung der Laufzeit eingerichtet werden muss. Werden alle Merkmale ausgewählt, dann ergänzen sich beide Faktoren, so dass die Latenz vom Auslösen des Interrupts bis zur Behandlung auf durchschnittlich 6µs anwächst.

Ausführungszeit eines Task-Wechsels Die Ergebnisse dieser Messung sind in Abbildung 5.2 dargestellt. Wie zu erwarten, wird die Dauer eines Taskwechsels nur beim Hinzufügen der Merkmale *Inter Arrival Rate* und *Execution Time Budget* durch das dort nötige Einrichten der Deadlines beeinflusst. Die leicht erhöhte Dauer des Task-Wechsel bei Aktivierung des *Execution Time Budgets* wird durch die beim Beenden eines Task erfolgende Berechnung eines Restbudgets verursacht.

Ausführungszeit des Sperrens eines gemeinsamen Betriebsmittels Die Ergebnisse dieser Messung sind in Abbildung 5.3 dargestellt. Das Einrichten einer Deadline für die maximale Sperrdauer einer Ressource verzögert den Sperrvorgang um ungefähr 2,5µs, während die anderen Merkmale erwartungsgemäß keinen Einfluß haben.

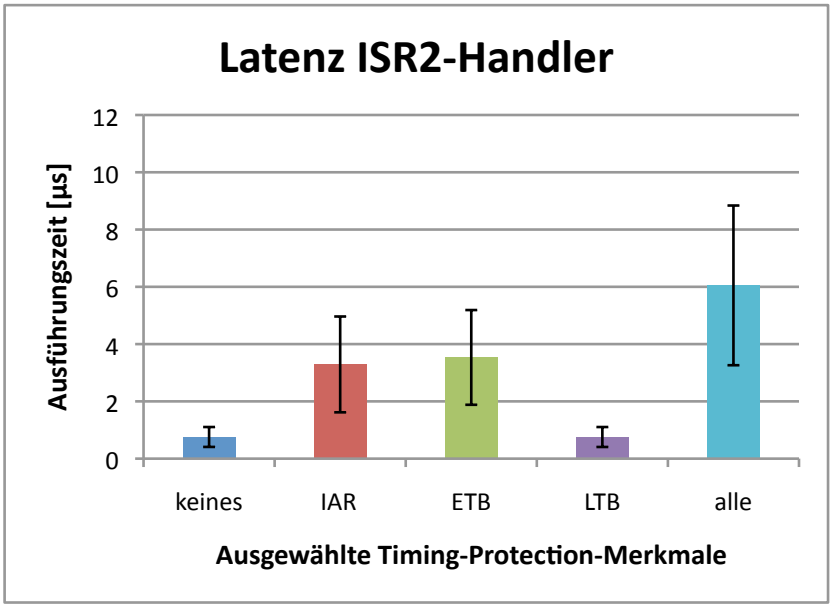

Abbildung 5.1.: Zeitspanne vom Auslösen eines ISR2 bis zu seiner Behandlung mit verschiedenen Timing-Protection-Merkmalen

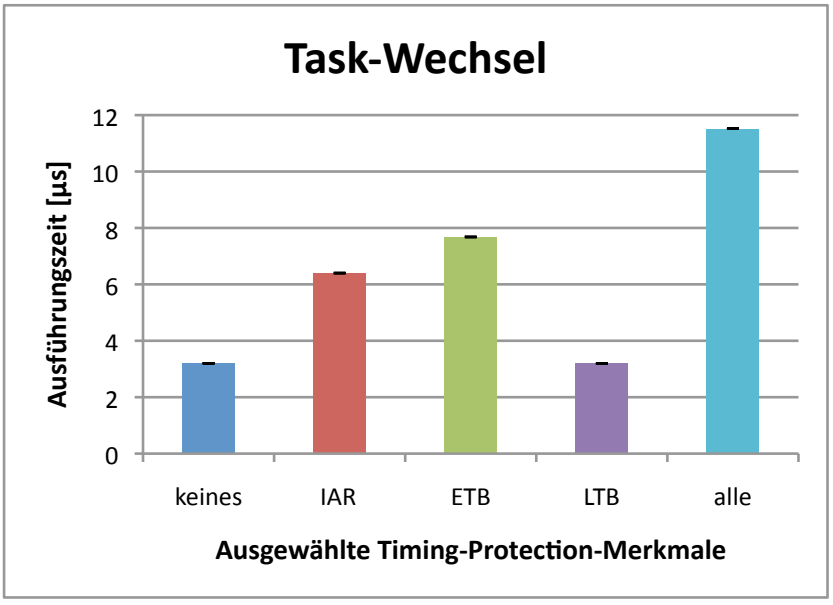

Abbildung 5.2.: Messungen der Ausführungszeit eines Task-Wechsels mit verschiedenen Timing-Protection-Merkmalen

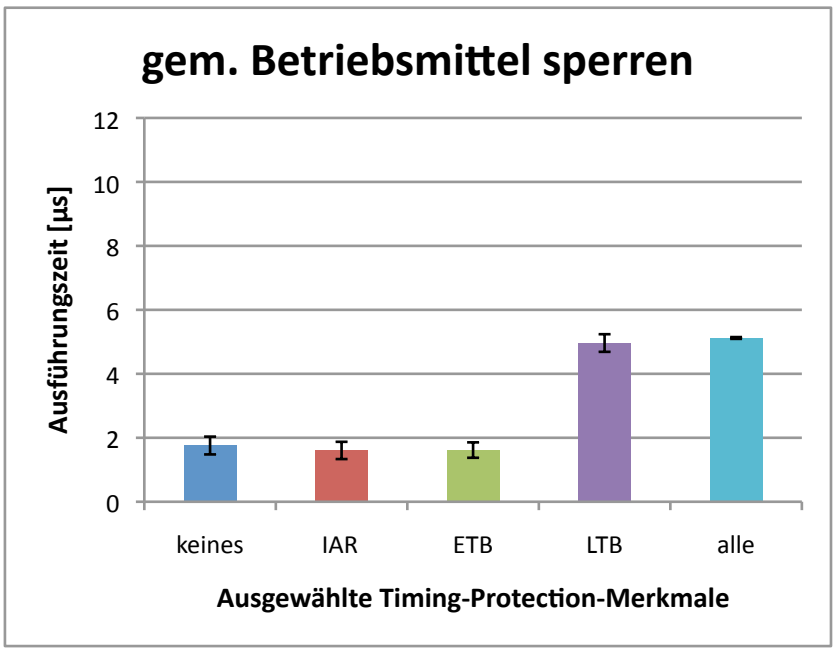

Abbildung 5.3.: Messungen der Ausführungszeit einer Betriebsmittel-Sperrung mit verschiedenen Timing-Protection-Merkmalen

5.2. Größen-Messungen

Die Größenmessungen erfolgten anhand der Binärdateien im ELF-Format, welche auch für die Laufzeitmessungen auf die verwendete TriCore-Plattform geladen wurden. Die Ergebnisse sind in Abbildung 5.4 dargestellt. Als Grundlage für den Vergleich dient ein Basissystems, welches keine Timing-Protection-Merkmale enthält. zum Vergleich wurde das System jeweils mit den Merkmalen *Inter Arrival Rate*, *Execution Time Budget* und *Locking Time Budget* übersetzt, wobei jeweils immer alle verfügbaren Optionen der einzelnen Merkmale aktiviert wurden. Zuletzt wurde noch ein System erstellt, welches alle Timing-Protection-Merkmale enthielt.

BSS-Segment Der Zuwachs im BSS-Segment hängt von den zusätzlichen statischen Klassen ab, die für die Timing-Protection benötigt werden. Für die statischen Variablen im Timer `STMTimer1` sowie in der Klasse `TPDeadlines` werden 109 Byte belegt; den Großteil davon belegt das für die Deadline-Queue benötigte statische Array von `QueueElements`. Der Zuwachs hier ist also konfigurationsabhängig, denn die Anzahl der für die Queue verfügbaren Elemente kann vom Konfigurator festgelegt werden.

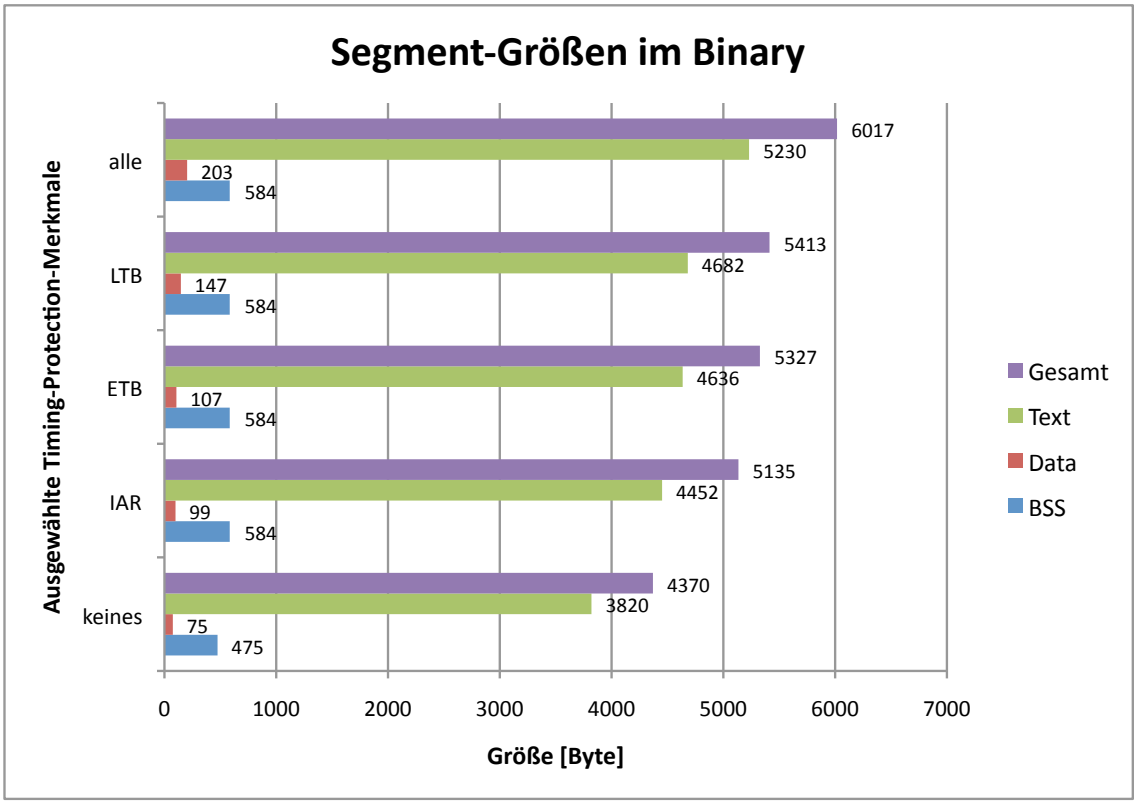

Abbildung 5.4.: Die Größe des übersetzten Betriebssystems in Abhängigkeit von den gewählten Timing-Protection-Merkmalen

Daten-Segment Im Daten-Segment werden die globalen Variablen abgelegt; in der gemessenen Beispielkonfiguration finden sich hier die Arrays für die ISR- und Task-Structs. Da die Timing-Protection-Merkmale durch Slices zusätzliche Variablen in diesen Structs anlegen, entsteht hier ein entsprechender Größenzuwachs.

Text-Segment Die Größenveränderungen im Text-Segments spiegeln wieder, dass zum einen neue Funktionen, z.B. für die Deadline-Verwaltung, hinzugefügt wurden. Zum anderen schlägt sich hier auch das Hinzufügen von Funktionsteilen bei *before*- oder *around*-Pointcuts nieder.

5.3. Diskussion

Abschließend wird diskutiert, wie gut die Ziele der Arbeit erreicht wurden, und wie Vorgehensweisen und Werkzeuge eingesetzt wurden.

Implementierung der Funktionalität

Mit Hilfe der Aspektorientierung ließ sich die Realisierung der *Timing Protection* sauber vom Rest des Systems trennen. Da die Erweiterung des Systems darauf basiert, *Pointcuts* zu definieren, an denen neuer Code zum Einsatz kommen soll, ist jedoch ein grundlegendes Verständnis des bestehenden Systems unabdingbar; die Abstraktion durch die Aspektorientierung steht und fällt mit einer Bereitstellung geeigneter Pointcuts, die verwendet werden können, um das Grundsystem ohne intime Kenntnisse der internen Abläufen zu erweitern. Davon abgesehen, ermöglicht die Aspektorientierung eine saubere Zusammenfassung und Kapselung des neu hinzuzufügenden Belanges. Dabei ist es allerdings nicht ausreichend, Funktionen und Datenstrukturen einfach zu erweitern; man muss sich ebenfalls Gedanken machen über die Reihenfolge von Funktionsaufrufen in den Erweiterungen, um unerwünschte Nebeneffekte zu vermeiden.

AUTOSAR-Konformität

Für diese Arbeit wurde kein 100%-iges Erfüllen des AUTOSAR-Standards angestrebt, allerdings diente die Definition der Timing Protection nach AUTOSAR als Leitlinie für Planung und Umsetzung. Wo nötig, wurde von den Forderungen des AUTOSAR-Standards Abstand genommen, so zum Beispiel bei der Speicherung des Timerstandes in 32-Bit-Variablen statt der eigentlich vorgesehenen 64-Bit-Variablen.

Zudem war es Teil der Aufgabenstellung, das Merkmal *Timing Protection* feinkonfigurierbar zu gestalten und so mit einer möglichst großen Variationsbreite auszustatten; der Standard sieht diese Variabilität nicht vor, sonndern erlaubt nur ein Aktivieren oder Deaktivieren mit allen Untermerkmalen.

Eingesetzte Werkzeuge

Die aspektorientierte Sprache AspectC++ erwies sich als mächtiges Werkzeug, um querschneidende Belange in ein bestehendes System einzufügen. Allerdings erschwert die teilweise noch dürftige Dokumentation den Einstieg bisweilen. Da die Sprache jedoch noch relativ neu und in Entwicklung begriffen ist, ist damit zu rechnen, dass dieser Missstand im Laufe der Zeit und durch eine größere Verbreitung nach und nach behoben werden wird.

Die Variabilität des Systems war mit dem Werkzeug *pure::variants* gut abbildbar. Für die Verarbeitung konfigurationsspezifischer Details und die Erzeugung von Quelltext aus diesen wurde XSLT eingesetzt. Dies stellte zwar kein Problem dar, allerdings wäre hier auch der Einsatz einer Skriptsprache wie z.B. *perl* denkbar, da Kenntnisse solcher Sprachen verbreiteter sein dürften als die der XML-Technologien XSLT und XPath.

Eigene Vorgehensweise

Der Plan, zunächst die Domäne *Timing Protection* komplett zu analysieren und darauf aufbauend die Erweiterung des Systems zu entwerfen und schließlich zu programmieren, funktionierte nicht immer gut. Ein großes Problem stellte dar, dass das zu erweiternde System zunächst unbekannt war und nicht alle benötigten Komponenten aufwies bzw. diese nicht funktionierten. Daher erwies sich oft der Ansatz als praktikabler, nach Verschaffung eines Überblicks über die Anforderungen direkt mit der Implementierung zu beginnen, so dass unvorhergesegene Probleme direkt gelöst werden konnten und man nicht durch einen möglicherweise nicht passenden Entwurf eingeschränkt wurde.

Wenn allerdings der Programmierer einer Erweiterung eines Systems mittels aspektorientierter Programmierung das System detailliert kennt, spricht nichts gegen die ursprüngliche Herangehensweise. Sollte der Ersteller dieser Arbeit noch einmal einen querschneidenden Belang für CiAO umsetzen, so wäre ein solches Vorgehen aufgrund der nun vorhandenen Detailkenntnisse von CiAO und der verwendeten Werkzeuge vermutlich ohne größere Probleme möglich.

Fazit

Die hier erreichte feingranulare Konfigurierbarkeit beweist, dass im Zusammenspiel aller Werkzeuge auch kleine Implementierungsdetails so gestaltet werden können, dass der Konfigurator des Systems sie sinnvoll verwenden kann. Aspektorientierte Programmierung, bzw. die Umsetzung durch AspectC++ ist ein mächtiges Werkzeug, mit dem querschneidende Belange besser und leichter verständlich als mit anderen Methoden (Präprozessor, Template Meta Programming)gekapselt werden können.

Abbildungsverzeichnis

Literaturverzeichnis

[1] "AUTOSAR homepage." http://www.autosar.org/.

[2] "OSEK/VDX standard." http://www.osek-vdx.org/.

[3] AUTOSAR, "Main requirements (version 2.0.5)," tech. rep., Automotive Open System Architecture GbR, Aug. 2008.

[4] AUTOSAR, "Requirements on operating system (version 2.0.5)," tech. rep., Automotive Open System Architecture GbR, Aug. 2008.

[5] W. Schröder-Preikschat, "Echtzeitsysteme." http://www4.informatik.uni-erlangen.de/Lehre/Veranst/V_EZS, Wintersemester 2005/06.

[6] AUTOSAR, "Specification of operating system (version 3.1.1)," tech. rep., Automotive Open System Architecture GbR, Feb. 2009.

[7] K. Czarnecki and U. W. Eisenecker, *Generative Programming. Methods, Tools and Applications.* Addison-Wesley, May 2000.

Anhang A.

Konfiguration des CiAO-Kerns

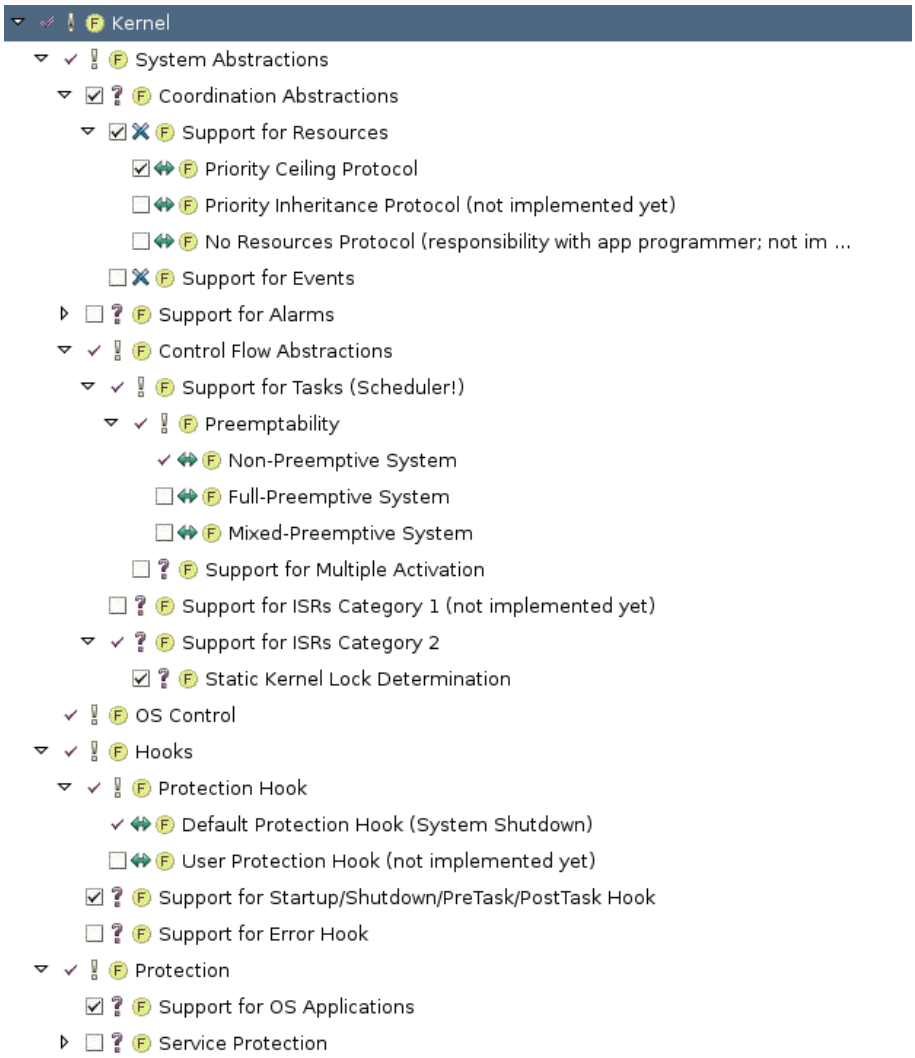

Abbildung A.1.: Die Konfiguration des CiAO-Kerns für die Beispiel-Applikation

Anhang B.

Die Evaluations-Applikation

Listing B.1: appcode.h

```
class App {
  private:
    static App inst_;
    static int foo;
  public:
    static App &Inst() { return inst_; }
    static void functionTaskTask0 (void);
    static void functionTaskTask1 (void);
    static void functionISRButton1ISR (void);
};
```

Listing B.2: appcode.cpp

```
int App::foo = 0;

void App::functionTaskTask0 (void) {
  int i = 0;
  hw::irq::DMA_SYSSRC3::enable();
  hw::irq::STM_SRC1::enable();

  while (1 > 0){
    i++;
    if ((i % (1024 * 1)) == (256 * 0)){
      asm volatile ("FROMSCHEDULE:");
      AS::ChainTask (Task1);
    }
    if ((i % (1024 * 1)) == (256 * 1)){
      asm volatile ("FROMLOCKRES:");
      AS::GetResource (0);
      asm volatile ("TOLOCKRES:");
    }
    if ((i % (1024 * 1)) == (256 * 2)){
      asm volatile("FROMISR:");
```

```
21        hw::irq::DMA_SYSSRC3::set();
22      }
23      if ((i % (1024 * 1)) == (256 * 3)){
24        asm volatile ("FROMUNLOCKRES:");
25        AS::ReleaseResource (0);
26        asm volatile ("TOUNLOCKRES:");
27      }
28    }
29  }
30
31  void App::functionTaskTask1 (void) {
32    asm volatile ("TOSCHEDULE:");
33    int i = 0;
34
35    while (1 > 0){
36      i++;
37      if ((i % (1024 * 1)) == (256 * 0)){
38        AS::ChainTask (Task0);
39      }
40      if ((i % (1024 * 1)) == (256 * 1)){
41        asm volatile ("FROMLOCKISR:");
42        AS::DisableInterruptSource (0);
43        asm volatile ("TOLOCKISR:");
44      }
45      if ((i % (1024 * 1)) == (256 * 2)){
46        asm volatile ("FROMUNLOCKISR:");
47        AS::EnableInterruptSource (0);
48        asm volatile ("TOUNLOCKISR:");
49      }
50      if (foo == 1024 * 4){
51        //block program with endless loop
52        for (volatile int stop = 1; stop > 0; stop = 1)
53          ;
54        }
55    }
56  }
57
58
59  void App::functionISRButton1ISR (void) {
60    asm volatile ("TOISR:");
61
62    for (volatile int i = 0; i < 1024*8; i++)
63      foo++;
64  }
```

Listing B.3: config.xml

```
<?xml version="1.0"?>

<ciaoApp name="EvaluationTimingProtection"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="../../../../../../../../tools/xml/ciaoAp

  <applicationmode name="OSDEFAULTAPPMODE">
    <autostarttasks>
      <taskref name="Task0"/>
    </autostarttasks>
  </applicationmode>

  <task name="Task0"
    function="functionTaskTask0"
    priority="3"
    activation="1"
    schedule="non"
    stacksize="128"
    timeframe="100"
    executionbudget="40000"
    resourcelockbudget="10000"
    isrlockbudget="20000"
  />

  <task name="Task1"
    function="functionTaskTask1"
    priority="5"
    activation="1"
    schedule="non"
    stacksize="128"
    timeframe="100"
    executionbudget="40000"
    resourcelockbudget="10000"
    isrlockbudget="20000"
  />

  <isr name="Button1ISR"
    function="functionISRButton1ISR"
    device="DMA_SYSSRC3"
    timeframe="100"
    executionbudget="10000000"
    resourcelockbudget="10000"
    isrlockbudget="20000"
  />

```

```
46    <resource name="CommonResource"
47      ceiling="3"/>
48
49    <osapplication name="theOneApp" trusted="false">
50      <componentref name="App"/>
51    </osapplication>
52
53    <component name="App">
54      <taskref name="Task0"/>
55      <taskref name="Task1"/>
56      <isrref name="Button1ISR"/>
57    </component>
58
59    <applicationheaders>
60    <applicationheader name="appcode.h"/>
61    </applicationheaders>
62  </ciaoApp>
```

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum Unterschrift