Dynamische Softwarearchitektur für eingebettete Systeme

Frank Slomka und Christian Hausner
Institut für Eingebettete Systeme/Echtzeitsysteme
Universität Ulm

Mit der UML und den dazugehörigen Erweiterungen SysML und MARTE kann die statische Architektur einer Software gut beschrieben werden. Ein Problem im Rahmen eines ganzheitlichen Entwicklungsprozesses für eingebettete Software stellt die Dynamik des Systems dar. Insbesondere das dynamische Zusammenspiel zwischen der Hardware, dem Speichermodell, dem Betriebssystem und der Anwendungssoftware kann nur unzureichend strukturiert und beschrieben werden. Ausgehend von einem neuen Entwicklungsprozess wird eine zu UML kompatible Beschreibungsform vorgestellt, die die Besonderheiten des dynamischen Verhaltens eingebetteter Software berücksichtigt. Ein besonderes Augenmerk gilt dabei der Dynamik der Hardware/Software-Schnittstelle.

Einleitung

Eingebettete Systeme werden für die verschiedensten Einsatzgebiete und Aufgaben immer bedeutender und werden in stärkerem Maße miteinander vernetzt. Damit ist auch der Entwurf eingebetteter Systeme durch steigende Anforderungen und neue Herausforderungen gekennzeichnet. Dazu zählen:

- die steigende Komplexität der zu entwerfenden Systeme bedingt durch die komplexeren und vielfältigeren Aufgaben
- gesteigerte Vernetzung untereinander, mit der Umgebung und informationstechnischen Systemen
- Korrektheit, Robustheit, Verfügbarkeit und Sicherheit im zunehmend autonomen Einsatz in “unkontrollierten” Umgebungen.

Eine Lösung um derartige Herausforderungen zu meistern ist die Modellierung von Systemen. Aufgrund der verschiedensten Einsatzgebiete und deren unterschiedlicher Beschaffenheit muss die Modellierung die Spezialitäten, Methoden und Techniken dieser Einsatzgebiete berücksichtigen, spezialisierte Sichten darauf bereitstellen und miteinander vereinen/integrieren können (siehe dazu auch [4]).

Eine weitere etablierte Methodik um die Herausforderungen, insbesondere die Komplexität im Entwurf derartiger Systeme zu bewältigen ist die Plattform-Entwicklung (eng. Platform-based design, siehe [1] and [8]).

Diese Entwurfsmethoden werden durch den komponentenbasierten Ansatz ergänzt. Man versteht unter einer Komponente einen modularen Teil eines (komplexen) Systems der in seiner Umgebung durch eine äquivalente Komponente ersetzt werden kann. Eine Komponente realisiert eine in sich abgeschlossene Teilfunktion eines Systems.

Die im Folgenden eingeführte Entwurfsmethodik stützt sich konsequent auf Plattform-Entwicklung und Modellierung von eingebetteten Systemen und deren Umgebung. Damit soll den genannten Herausforderungen begegnet werden und die Wiederverwendung von Plattform- und Anwendungs-Modellen in verschiedenen Produkten und Produktlinien erreicht werden. Wir haben die Entwurfsmethodik in [9] eingeführt, das Plattform-Modell wurde durch [2] ergänzt und detailliert. Insbesondere die Modellierung von Plattformen derartiger komplexer Systeme ist eine komplexe Aufgabe die durch unsere Entwurfsmethodik verbessert werden soll. Dabei kommt vor allem das Konzept der hierarchischen Modellierung zum Einsatz.

Die Modellierung der Anwendung und der Anwendungsumgebung ist ein weiterer wesentlicher Aspekt unserer Entwurfsmethodik. Neben funktionalen Anforderungen existiert eine Vielzahl nicht-funktionaler Anforderungen die durch die Plattform erfüllt werden müssen. Im Kapitel ”Anforderungsdefinition” werden wir eine Methode vorstellen um derartige Anforderungen (im Allgemeinen Einschränkungen, daher auch Constraints) grafisch zu modellieren.

Verwandte Arbeiten

Mit der UML [7] und den dazugehörigen Erweiterungen SysML [5] und MARTE [6] kann die statische Architektur einer Software gut beschrieben werden. UML wird für die Modellierung von eingebetteten Systemen durch die Erweiterungen MARTE und SysML ergänzt. SysML ermöglicht die Modellierung aller Arten von Systemen, ist dabei aber weniger konkret in der Definition der Semantik der SysML-Elemente und -Diagramme. Der Plattform-basierte Ansatz wird von SysML nicht explizit unterstützt. Für diesen Zweck kann sie mit MARTE kombiniert werden.

MARTE wurde für den Plattform-basierten Ansatz entwickelt und unterstützt, genau wie unsere Methodik, die hierarchische Modellierung von Plattformen. Ebenfalls nutzt MARTE das Konzept der Ressourcen und Dienste (siehe Kapitel ”Adressräume und Dienste des Betriebssystems”).

Insbesondere das dynamische Zusammenspiel zwischen der Hardware, dem Speichermodell, dem Betriebssystem und der Anwendungssoftware kann mit UML und den Erweiterungen SysML und MARTE nur unzureichend strukturiert und beschrieben werden.

Der epizyklische Entwurfsprozess

Unsere Entwurfs-Methodik erweiterte objektorientierte Entwurfsmethoden wie in [3] auf einen Technologie-unabhängigen komponentenbasierten Ansatz. Er ist iterativ und enthält einen Hauptprozess der aus Teilprozessen besteht. Wir visualisieren unseren Ansatz mittels Kreisen um darzustellen, dass es sich um Iterationen handelt. Alle Teilprozesse sind als Epizyklen auf dem Hauptprozess dargestellt und wird daher als epizyklischer Entwurfsprozess bezeichnet. Abbildung 1 gibt einen Überblick über den epizyklischen Entwurfsprozess.

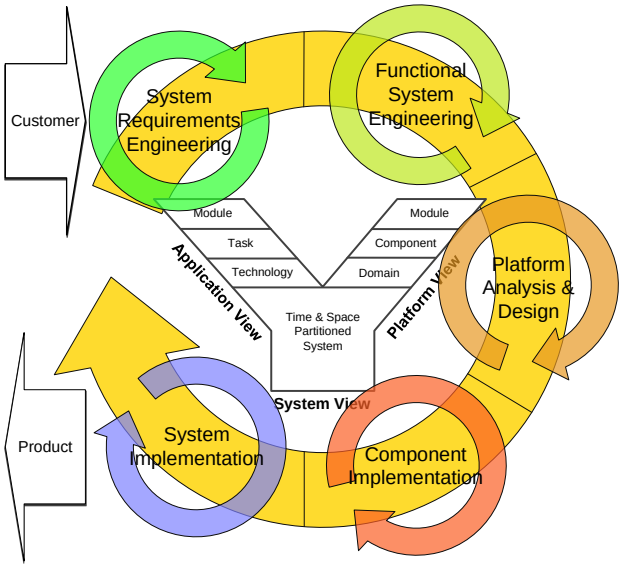

Abb. 1: Epizyklischer Entwurfsprozess - Übersicht

Eine Iteration des Hauptprozesses erzeugt eine Ausprägung (Version, Produkt) eines eingebetteten Systems. Eine Iteration besteht aus 5 Teilprozessen die aus Entwurfsschritten besteht. Teilprozesse erzeugen Arbeitsprodukte (Zwischenprodukte) und haben Abhängigkeiten zueinander. Die folgenden 5 Teilprozesse sind definiert:

1. System-Anforderungsanalyse
2. funktionaler Systementwurf
3. Plattformentwurf
4. Komponenten-Implementierung
5. System-Integration (Implementierung)

Der Hauptprozess kann iterativ kann mehrfach durchlaufen werden. Das zu entwerfende eingebettete System wird in mehreren Iterationen entwickelt (Ausprägungen, Versionen, Baumustern). Eine Iteration des Hauptprozesses ist vergleichbar mit dem einmaligen Durchlaufen des klassischen V-Modells.

Teilprozesse können ebenfalls iterativ mehrfach durchlaufen werden. Sie produzieren Arbeitsprodukte für andere Teilprozesse sowie bei Bedarf interne Arbeitsprodukte und benötigen Inputs anderer Teilprozesse oder aus der Umgebung.

Das Ziel des epizyklischen Entwurfsprozesses ist einerseits eine automatisierte Erstellung von Systemen anwendungszentriert zu ermöglichen. Andererseits soll die systematische und wiederholbare Erstellung von Systemen ermöglicht werden. Die konkrete Plattform bzw. das System entsteht aus Kundenanforderungen bzw. Anforderungen der Anwendung und verfügbaren Plattform-Modellelementen. Dadurch kann sowohl ein optimal auf die Anwendung angepasstes System entwickelt werden als auch der Plattformansatz effizient verfolgt werden.

System-Anforderungsanalyse und funktionaler Systementwurf

Die System-Anforderungsanalyse beinhaltet das klassische Requirements-Engineering mit textuellen Anforderungen die von Bildern/Diagrammen ergänzt werden. In dieser Phase werden Kundenanforderungen aufgenommen, bewertet und abgestimmt. Da die hier erfassten Anforderungen und ihr Einfluss auf das zu entwerfende System in allen nachfolgenden Phasen benötigt werden, werden insbesondere die nicht-funktionalen Anforderungen hier bereits modelliert (formalisiert, wie im Kapitel ”Anforderungsdefinition” beschrieben). Das Ergebnis dieser Phase ist eine Anforderungs-Spezifikation.

Der Funktionale Systementwurf beinhaltet die Modellierung eines funktionalen Systems (auch Anwendung) anhand der funktionalen Anforderungen. Die nicht-funktionalen Anforderungen werden im Modell annotiert um die Auswirkungen auf Anwendung und später Plattform berücksichtigen zu können.

Das Anwendungs-Modell

Das Metamodell für die Anwendung und das umgebende System besteht aus Modulen, Tasks, Ports und Links. Ports dienen als Verbindungspunkte zwischen Modulen und Tasks und werden mittels Links verbunden. Wir unterscheiden zwischen strukturellen Ports und transferorientieren Ports. Im Folgenden werden nur transferorientierte Ports berücksichtigt.

Wir unterscheiden eine Technologie-unabhängige Modellierung und in der nächsten Verfeinerung eine Technologie-abhängige Modellierung. Der Übergang zwischen beiden Phasen wird als Technologie-Bindung bezeichnet (siehe [9]). Anwendungs-Module sind Elemente die andere Elemente wie Module und Tasks enthalten können. Sie werden verwendet um einen hierarchischen Entwurf zu ermöglichen. Ein Modul ist ein strukturelles Element das der Kapselung von Funktionen dient (Geheimnisprinzip). Module dienen der Gruppierung von Tasks oder Modulen mit hoher funktionaler Kopplung.

Module sind keine Elemente physischer Kapselung und sie korrespondieren nicht zwangsläufig mit Elementen der Plattform. Module beschreiben entweder ein Umgebungsmodell oder ein technisches System (die Anwendung).

Das Umgebungsmodell ist ein Modell eines relevanten Ausschnitts der realen Welt das im direkten Zusammenhang mit dem zu entwerfenden technischen System steht. Das technische Systemmodell interagiert mit dem Umgebungsmodell um die geforderten Aktionen/Reaktionen zu erreichen, also die funktionalen Anforderungen zu erfüllen.

Zusammen mit seiner Plattform realisiert ein technisches Systemmodell ein eingebettetes System.

Dem Gedanken des hierarchischen Enthaltenseins von Modulen folgend ist die gesamte Anwendungs-Architektur ebenfalls ein Modul das sowohl das technische System als auch notwendige Umgebungsmodelle enthält.

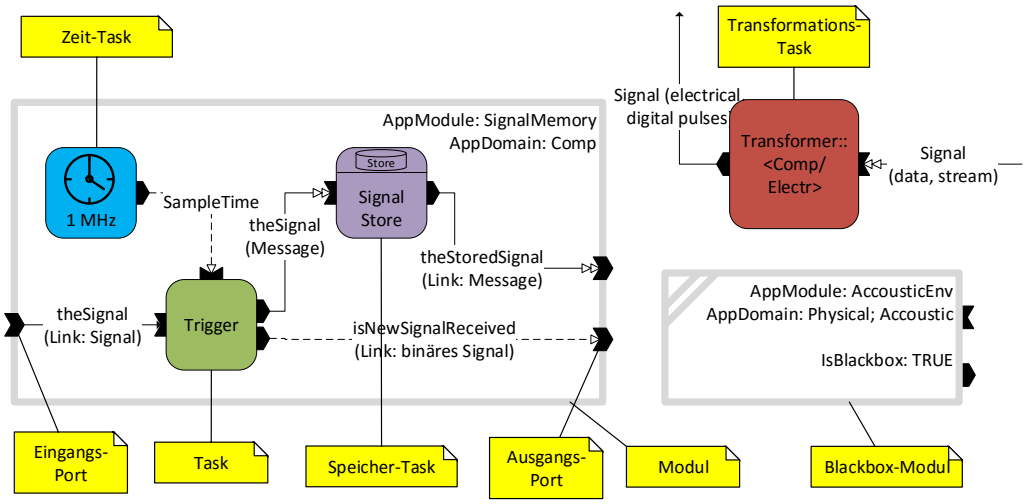

Abb. 2: Elemente des Anwendungs-Modells

Anwendungs-Tasks (Tasks) beschreiben und modellieren das Verhalten von Anwendungen. Tasks kapseln das Verhalten und können über Ports miteinander kommunizieren. Ein Task ist ein Entwurfselement das eine Basis-Funktionalität (Verhalten) mit Eingangsgrößen und Ausgangsgrößen bereitstellt. Tasks können beispielsweise Modelle, Gleichungen oder Funktionen sein. Jeder Task besitzt mindestens einen Port.

Neben Tasks mit Basis-Funktionalität gibt es spezielle Tasks die mit anderen Technologien interagieren. Sie werden als Transformations-Tasks bezeichnet und ermöglichen den Austausch zwischen unterschiedlichen Technologien. Andere Spezial-Tasks interagieren mit der Plattform eines technischen Systems. Wir unterscheiden hier Zeit-Tasks die eine Schnittstelle zu Zeit-Informationen der Plattform (Zeit-Ereignissen) bereitstellen und Speicher-Tasks die eine Schnittstelle zu Speicher-Ressourcen der Plattform bereitstellen.

Ports dienen als Kommunikations-Endpunkte für Tasks und Module. Ports werden untereinander durch Links verbunden. Sowohl Ports als auch Links sind durch ihre Schnittstellenbeschreibung typisiert. Nur kompatible Ports können durch passende Links miteinander verbunden werden. Ports modellieren die Ein- und Ausgänge von Tasks und Modulen und beschreiben deren Schnittstellen. Es werden Eingangs- und Ausgangs-Ports unterschieden.

Module können nur spezielle Ports besitzen die als Proxy-Ports bezeichnet werden. Sie dienen als externe Schnittstelle eines Moduls und verbinden die Umgebung des Moduls mit den internen Modulen oder Tasks. Es handelt sich um virtuelle Ports die der Sicherstellung von Kapselung und Geheimnisprinzip des Moduls und der enthaltenen Tasks und Module dient.

Ein Link verbindet zwei kompatible Ports. Ein Ausgangs-Port liefert Entitäten (zum Beispiel Informationen) abhängig von seiner Schnittstellenbeschreibung und ein Eingangs-Port empfängt diese Entitäten. Der verbindende Link erbt seine Schnittstellenbeschreibung von den Ports. Die Art der Schnittstelle kann mittels Symbolen am Link visualisiert werden.

Im ersten Entwurfsschritt werden die funktionalen Anforderungen in eine Technologie-unabhängige Anwendung überführt. Der nächste Schritt verfeinert diese Anwendung (technisches System) indem Entscheidungen über die realisierenden Technologien getroffen werden. Dieser Schritt wird Technologie-Bindung genannt. Hier werden im Rahmen des funktionalen Systementwurfs die ersten Plattform-Entscheidungen getroffen, denn jede Entscheidung über die Technologie bedingt Entscheidungen und Anforderungen an die Plattform. Hier wird z.B. die Entscheidung zwischen Digitaltechnik oder Analogtechnik getroffen. Wie weiter oben beschrieben wird die Verbindung zwischen unterschiedlichen Technologien durch sog. Transformations-Tasks modelliert.

Ein Modul kann unterschiedliche Technologien enthalten: ein mathematisches Modell, Digitaltechnik (Computersystem oder elektronisches System), ein analoges elektronisches System (Analogtechnik) oder ein physikalisches System (z.B. Mechanik, Optik, Akustik; häufig bei Umgebungsmodellen).

Anforderungsdefinition

Neben funktionalen Anforderungen unterscheiden wir nicht-funktionale (oder extra-funktionale) Anforderungen. Nicht-funktionale Anforderungen beschreiben Forderungen an z.B. die Leistungsfähigkeit, Reaktionsgeschwindigkeit oder Skalierbarkeit des Systems. Da sie häufig Einschränkungen beschreiben unter denen ein System seine Aufgaben erfüllen soll, werden sie im weiteren Verlauf als Constraints bezeichnet.

In der System-Anforderungsanalyse werden nicht-funktionale Anforderungen formalisiert und als Constraints modelliert. Es handelt sich sowohl um statische als auch dynamische Constraints. Dabei wird aktuell zwischen den folgenden Typen von Contraints unterschieden:

- Zeit
- Fläche oder Kosten
- Leistung und Energie
- Sicherheit und Zuverlässigkeit, Verfügbarkeit

Aufgrund der Erweiterbarkeit des Metamodells können einfach weitere Arten von Constraints hinzugefügt werden.

Constraints haben einen Gültigkeitsbereich, eine Menge von Modellelementen auf die sie wirken. Sie werden im Anwendungsmodell annotiert. Das erfolgt indem sie zwischen Ankerpunkten (Probes) spezifiziert werden. Diese Ankerpunkte können an jedes Modellelement der Anwendung gebunden werden. Um Constraints, deren Gültigkeitsbereich und Auswirkungen zu visualisieren, werden Symbole für Constraints definiert (Abbildung 3). Der Gültigkeitsbereich eines Constraints wird farblich hervorgehoben.

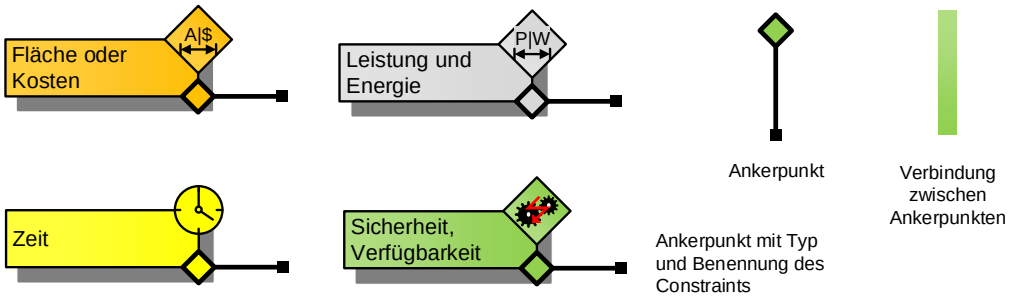

Abb. 3: Nicht-funktionale Anforderungen (Constraints) und Ankerpunkte (Probes)

Die eigentliche nicht-funktionale Anforderung wird durch einen Ausdruck modelliert der für den Gültigkeitsbereich des Constraints erfüllt sein muss. Hierbei unterscheiden sich verschiedene Arten der Erfüllung, z.B. kann eine Constraint erfüllt sein, wenn er auf allen Elementen des Gültigkeitsbereichs in Summe erfüllt ist ($\forall \sum$) oder wenn er für jedes Element des Gültigkeitsbereichs einzeln erfüllt ist ($\forall$).

Als Beispiel kann eine zeitliche Einschränkung die maximale Reaktionszeit eines Signals spezifizieren:

$$t_{\max} \leq 71\,ms \,|\quad \forall \sum (\text{Elemente des Gültigkeitsbereiches}) \tag{1}$$

Dieser Constraint wird mittels Ankerpunkten an die Kommunikations-Links der Anwendung annotiert die dieses Signal beinhalten (Gültigkeitsbereich). Damit wird im funktionalen Systementwurf auch die geforderte zeitliche Einschränkung des Signals visualisiert. Er wird im weiteren Verlauf auch an die Plattform des Systems annotiert und schränkt damit die verwendbaren Plattform-Elemente ein. Mittels dieser formalen Annotationen kann in frühen Entwurfsphasen die Einhaltung der nicht-funktionalen Anforderungen sichergestellt werden.

Plattformentwurf

Ausgehend vom funktionalen Systementwurf und den nicht-funktionalen Anforderungen (Constraints) kann eine passende Plattform abgeleitet werden. Die Modellierung der Plattform für digitale Systeme wird im Folgenden vorgestellt.

Schichtenmodell und Rollentrennung

Wesentliche Konzepte im Systementwurf sind Kapselung und Schichtenbildung. Eine SW-Plattform kann in verschiedene Schichten unterteilt werden. Unser Plattform-Modell ermöglicht die Bildung von Schichten über ein hierarchisches Modell. Damit entspricht die Modellierung einer Plattform-Schicht der Modellierung einer Anwendung die an die darunterliegende Plattform gebunden wird. Unser Platform Konzept

basiert auf einer Schichtenbildung wobei jede Schicht nur mit ihrer direkt benachbarten höheren oder niederen Schicht kommuniziert.

Dieses Konzept vereinfacht auch die Rollen- und Zuständigkeitstrennung. In unseren generischen Schichtenmodell wird die unterste Schicht durch die HW-Plattform (Schicht 0) repräsentiert. Diese Schicht wird mittels des Plattform-Komponenten-Modells modelliert (siehe Kapitel ”Hardwarearchitektur”). Darüber liegende Plattform-Schichten stellen Dienste bereit die von der jeweils darüber liegenden Schicht (Anwendung) genutzt werden. Diese Schichten werden mittels des Plattform-Domänen-Modells beschrieben (siehe Kapitel ”Adressräume und Dienste des Betriebssystems”). Die Abbildung 4 zeigt einen Überblick des Schichtenmodells.

Dadurch werden die Abhängigkeiten zwischen unterschiedlichen Rollen im Entwurfsprozess reduziert. Beispielsweise wird ein Hardware-Entwickler die Hardware-Plattform mittels des Plattform-Komponenten-Modells beschreiben und mit dem Plattform-Entwickler abstimmen. Direkte Abhängigkeiten zwischen Hardware-Entwickler und Anwendungs-Entwickler werden minimiert. Dieser hat im Allgemeinen nur mit dem Plattform-Entwickler Abstimmungsbedarf, denn er benötigt nur die oberste Schicht der Plattform um seine Anwendung zu modellieren.

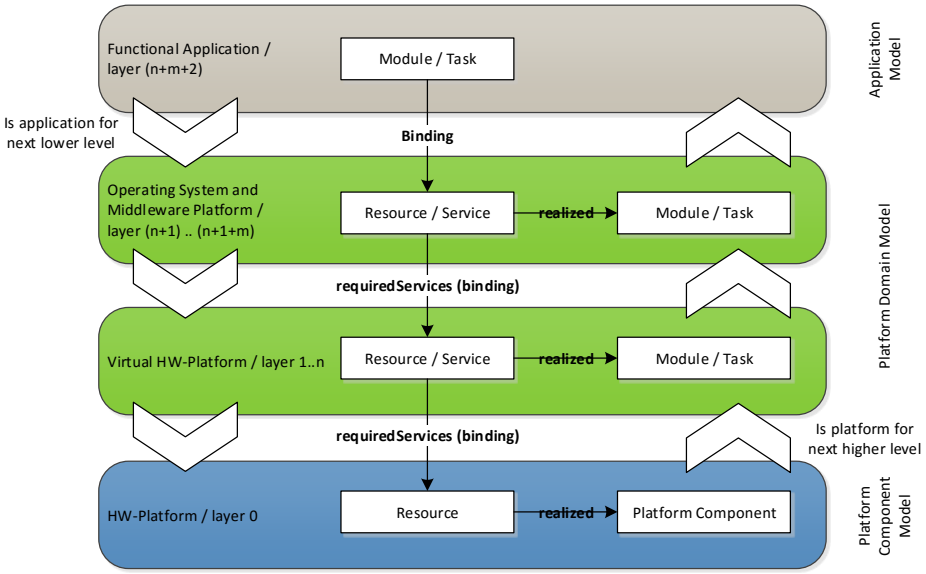

Abb. 4: Schichtenmodell

Hardwarearchitektur

Das Plattform-Komponenten-Modell (eingeführt in [2]) besteht aus Modulen und Komponenten und unterstützt ähnlich dem Anwendungs-Modell hierarchisches Design indem Module aus Modulen oder Komponenten bestehen.

Module sind beispielsweise physikalische Einheiten wie Geräte, Steckkarten oder Chips bzw. SoC (Systems on Chip). Komponenten sind die atomaren Hardware-Elemente, sie enthalten Bus Control Interfaces die als Schnittstellen dienen (siehe Abbildung 5). Ähnlich den Ports im Anwendungs-Modell können auch hier nur kompatible Bus Control Interfaces miteinander verbunden werden.

Weitere Elemente des Plattform-Komponenten-Modells sind Zeit-Elemente die Zeitgeber (Timer) und Uhren (Clocks). Sie können mit allen anderen Elementen verbunden werden und werden mittels eines Timing-Interface mit anderen Elementen verbunden (siehe auch Abbildung 5).

Ziel dieser Modellierung ist es, die Hardwarearchitektur von Plattformen strukturell bzw. in ihren Bausteinen zu beschreiben. Die Hardwarearchitektur kann durch das im Folgenden Kapitel beschriebene Plattform-Domänen-Modell in Hinblick auf Ressourcen und Dienste substituiert werden. Diese Dienste ermöglichen im weiteren Verlauf die Bindung der Applikation an die Plattform.

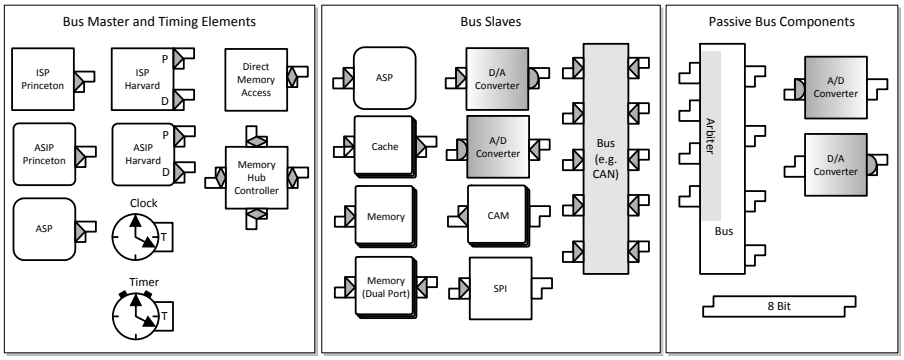

Abb. 5: Verfügbare Plattform-Komponenten

Adressräume und Dienste des Betriebssystems

In diesem Kapitel wird das Plattform-Domänen-Modell beschrieben. Die Idee hinter diesem Modell ist die Einführung einer Abstraktion die alle Aspekte einer Plattform aus Sicht der Anwendung berücksichtigt. Wir führen hier das Konzept der Domänen, Ressourcen und Dienste für alle Plattformen ein, unabhängig davon ob diese von Software oder Hardware bereitgestellt werden. Damit soll die Austauschbarkeit der Plattform und auch der Anwendung verbessert werden. Außerdem wird das Ziel verfolgt unabhängig von industriellen Einsatzgebieten zu sein.

Detaillierte Kenntnisse der Hardware-Plattform werden durch das Plattform-Domänen-Modell gekapselt. Die Anwendung muss nur die relevanten Informationen des Plattform-Domänen-Modells kennen. Mittels des Plattform-Domänen-Modells werden Plattformen logisch strukturiert und die physikalische Strukturierung aus dem Plattform-Komponenten-Modell (Geräte, Chips usw.) wird überwunden. Damit wird ein erhöhter Grad von Unabhängigkeit und Austauschbarkeit zwischen Plattform und Anwendung ermöglicht.

Plattform-Domänen

Eine Plattform-Domäne (im weiteren Domäne) ist ein Strukturelement des Plattform-Domänen-Modells. Es stellt einen Container dar, der logische Einheiten wie Ressourcen und Services kapselt. Das Domänen-Modell stellt die logische Sicht der Plattform

dar. Domänen können Sub-Domänen enthalten und ermöglichen damit wieder die hierarchische Modellierung von Plattformen.

Domänen besitzen eine Schnittstelle in Form von Diensten die von den enthaltenen Ressourcen bereitgestellt werden. Eine Domäne bindet Elemente des Anwendungs-Modells wie Tasks wenn sie eine Ausführungs-Ressource (Sequential Execution Resource) enthält. Besitzt sie keine derartige Ressource dient die Domäne als Container für weitere Sub-Domänen.

Domänen sind logische Einheiten die für die Realisierung des eingebetteten Systems in Hardware oder Software implementiert werden müssen.

In der untersten Plattform-Schicht sind Domänen die logische Sicht auf die mittels Komponenten modellierte Hardwarearchitektur. Hier können Domänen durch Module oder Gruppen von Komponenten (auch aus verschiedenen Modulen) realisiert werden. Dadurch kann eine Domäne Ressourcen besitzen die z.B. in verschiedenen Geräten oder Chips enthalten sind (siehe auch Kapitel ”Entwurfsstudie”).

Auf höheren Ebenen des Plattform-Schichtenmodells werden Domänen durch Software implementiert, beispielsweise durch Betriebssysteme und deren Zeit- und Speicher-Partitionen wie Prozesse und Threads. Eine mögliche Modellierung eines Betriebssystems im Domänen-Modell sieht so aus:

1. Betriebssystem selbst wird durch eine Domäne beschrieben
2. Jeder Prozess wird durch eine Sub-Domäne beschrieben
3. Jeder Thread wird durch eine Sub-Domäne der entsprechenden Prozess Sub-Domäne beschrieben

Falls die Anwendung keine Nebenläufigkeit oder Speichertrennung benötigt, wird im Allgemeinen auch kein Betriebssystem genutzt. Andere Dienste (z.B. Treiber für Plattform-Hardware) werden normalerweise trotzdem von der Anwendung benötigt und über eine Domäne beschrieben.

Ressourcen und Dienste

Ressourcen sind logische Elemente die Services für Anwendungs-Elemente oder Ressourcen höherer Schichten bereitstellen. Ressourcen andererseits benötigen Dienste niederer Schichten um ihre Aufgaben zu erfüllen.

Ressourcen-Typ	Beispiel
Prozessor-Ressource	CPU, µController, Scheduler (SW)
Speicher-Ressource	RAM, Cache, Stack
Zeit-Ressource	HW Uhr, SW Timer
Kommunikations-Ressource	CAN, Mailbox
Transformation-Ressource	A/D-Wandler
Ausführungs-Ressource	in jeder Domäne enthalten die Anwendungs-Elemente binden kann

Tab. 1: Typen von Ressourcen

Die von Ressourcen angebotenen Dienste bilden die Schnittstelle der Ressource. Man kann Dienste auch als die öffentlichen Methoden einer Ressource interpretieren. Dienste sind die einzige Möglichkeit Ressourcen aus einer höheren Schicht zu nutzen. Wir unterscheiden angebotene und benötigte Dienste sowie Eingangs- und Ausgangs-Dienste. Eingangs-Dienste sind z.B. schreibende Dienste oder Dienste die der Konfiguration einer Ressource dienen. Ausgangs-Dienste sind lesende Dienste oder Ereignis-Behandlungsroutinen. Die Einteilung in Eingangs- und Ausgangs-Dienste erfolgt immer aus dem Blickwinkel der den Dienst besitzenden Ressource.

Die Bindung von Plattform-Elementen benachbarter Schichten erfolgt durch die Verbindung von benötigten Diensten der Schicht X mit kompatiblen angebotenen Diensten der Schicht $X-1$. Auf der untersten Plattform-Schicht realisieren Plattform-Komponenten Ressourcen die Dienste bereitstellen. Diese Realisierungsbeziehung wird als Bindung in dieser Ebene betrachtet.

In einer Domäne werden normalerweise nicht alle möglichen Ressourcen und Dienste modelliert, sondern nur die die von höheren Schichten benötigt werden. Wir bezeichnen sie als relevante Ressourcen und Dienste.

Ressourcen und Dienste werden durch ihren Ressourcen-Typ, wie in Tabelle 1 dargestellt, klassifiziert. Weiterhin unterscheiden wir Ressourcen-Manager. Sie verwalten andere Ressourcen und deren Dienste. Ein typischer Ressourcen-Manager ist eine MMU (Memory Management Unit) die andere Speicher-Ressourcen verwaltet und als Proxy für sie und ihre Dienste dient.

Die zentralen Dienste die von Plattformen angeboten werden sind Rechenzeit bzw. Rechenkapazität, Speicher und Kommunikation. Wird eine Domäne nicht nur als Container für Sub-Domänen genutzt, stellt sie mindestens eine Rechenzeit-Partition (“Zeitscheibe”) dar die durch eine Ausführungs-Ressource (Sequential Execution Resource) modelliert ist. Das bedeutet, dass alle Tasks die durch die Anwendung an diese Domäne gebunden werden an genau eine Ausführungs-Ressource gebunden sind. Sie wird an eine Prozessor-Ressource gebunden. Das bedeutet, dass alle Aufgaben in einer Domäne nacheinander und nicht parallel (nebenläufig) abgearbeitet werden. Damit können mehrere Sub-Domänen einer Domäne nur durch eine verwaltete Prozessor-Ressource (Scheduler) mit Rechenzeit versorgt werden.

Im Gegensatz dazu können Domänen auch als Speicherpartition agieren und damit den gesamten verfügbaren Speicherbereich mittels Adressräumen separieren. Dabei werden alle Speicherzugriffe an virtuelle Speicher-Ressourcen (verwaltete Ressourcen) gebunden.

Bindung von Anwendung und Plattform

Dieser Entwurfsschritt kombiniert Anwendung und Plattform. Anwendungs-Elemente wie Tasks werden an angebotene Dienste der Plattform gebunden. Nachdem dieser Schritt erfolgt ist, kann das modellierte System analysiert und bei Bedarf verfeinert werden (erneute Iteration der vorgelagerten Prozesse).

Danach erfolgt die Implementierung oder Generierung des modellierten Systems bzw. seiner Komponenten (Teilprozess Komponenten-Implementierung) und Integration und Test der Komponenten zum eingebetteten System (Teilprozess System-Integration). Aufgrund der Verlagerung wesentlicher Analyse- und Testschritte wie Modellierung und Analyse der Einhaltung nicht-funktionaler Anforderungen (Constraints) in frühe Entwurfsphasen wird dieser Teilprozess deutlich vereinfacht.

Für diesen Entwurfsschritt wird eine spezielle Sicht zur Bindung von Anwendung und Plattform bereitgestellt. Sie wird von System- oder Anwendungs-Entwicklern genutzt um Anwendungs-Elemente an Plattformen zu binden. Hier werden die Anwendungs-Sicht und die Domänen-Sicht verschmolzen wobei die Anwendungs-Sicht über die Domänen-Sicht gelegt wird. Die grafischen Elemente der Domänen-Sicht werden reduziert weiter verwendet. Beispielsweise werden keine Ressourcen mehr angezeigt sondern nur noch die angebotenen Dienste denn die Darstellung der Plattform-Schnittstelle zur Anwendung und nicht die Darstellung der Plattform-Struktur ist die Hauptaufgabe dieser Sicht. Weitere Details zur grafischen Bindung sind Bestandteil unserer zukünftigen Forschungsarbeiten (siehe auch Kapitel ”Zusammenfassung”).

Entwurfsstudie

In diesem Kapitel wird die Modellierung eines Sonar-Systems für ein autonomes Unterwasserfahrzeug unter Anwendung unserer Methodik beschrieben. Die Hauptaufgabe des Sonar-Systems ist die Erkennung von Objekten im Wasser. Das Sonar-System sendet Schallwellen aus und empfängt die Echos die von Objekten im Wasser erzeugt werden. Ein solches System ist ein gutes Beispiel für die Nutzung unterschiedlicher Technologien. Es besteht aus elektromechanischen Teilen zur Erzeugung der Schallwellen, analogen elektronischen Teilen für die Ansteuerung der Schallerzeuger und für den Empfang der Echos sowie aus digitalen elektronischen Teilen mit Hard- und Software für die Signalverarbeitung und Objekterkennung.

Die Entwurfsstudie folgt unserer Methodik und teilt sich daher in die folgenden Abschnitte:

1. System-Anforderungsanalyse
2. Funktionaler Systementwurf
3. Plattformentwurf
4. Komponenten-Implementierung und System-Integration

Hierbei konzentrieren wir uns auf die ersten 3 Entwurfsprozesse, die Komponenten-Implementierung und System-Integration wird in dieser Entwurfsstudie nicht weiter betrachtet und sind Teil unserer weiteren Forschungsarbeiten (siehe Kapitel ”Zusammenfassung”).

System-Anforderungsanalyse

Es soll ein Sonar entwickelt werden, welches Unterwasserobjekte (Objekte) erkennen und einen Vektor errechnen kann, der Abstand und Ortung relativ zum System angibt. Die System-Anforderungsanalyse beginnt mit der Analyse und Aufstellung der wesentlichen Anforderungen die auszugsweise in der folgenden Abbildung 6 aufgeführt sind.

Anf.-ID	Anforderungs-Text
REQ-01	Das Sonar muss Unterwasserobjekte mittels akustischer Signale (Schallwellen) entdecken und deren Abstand ermitteln.
REQ-02	Das Sonar muss kontinuierlich akustische Signale von 200 kHz aussenden.
REQ-04	Um REQ-01 zu erfüllen muss das Sonar die Zeit t zwischen Aussenden eines akustischen Signals und Empfangen von Signalechos der Objekte messen.
REQ-05	Das Sonar soll in Objekte im Bereich von 2 m bis 200 m entdecken können.
REQ-06	Das Sonar muss nahe Objekte schneller entdecken können als weiter entfernte Objekte.
REQ-07	Das Sonar muss eine hohe Erkennungsauflösung bereitstellen.
REQ-08	Um REQ-06 und REQ-07 zu erfüllen muss zwischen den Entdeckungsteilbereichen (REQ-05) 2-50 m, 2-100 m, und 2-200 m ausgewählt werden können.
REQ-10	Das Sonar muss das Ergebnis (entdeckte Objekte und Abstand) über den CAN-Bus an die anderen Teilnehmer des Geamtsystems übermitteln.
REQ-11	Das Sonar muss mittels Kreuzkorrelation gesendete und empfangene Signale vergleichen können.
REQ-12	Das Sonar muss den Signalverlauf eines gesendeten Signals speichern.
REQ-14	Das Sonar muss Signale mit beliebiger Rechteckform aussenden können.
REQ-15	Das Sonar muss den Abstand d eines Objektes durch den Zeitunterschied t zwischen dem Senden und Empfangen des Signals ermitteln. Dazu wird folgende Formel angewendet: *d = 1/2 * t/C_Fl, C_Fl = Schallgeschwindigkeit Flüssigkeiten* .

Abb. 6: Anforderungen an das Sonar-System (Auszug)

funktionaler Systementwurf

Basierend auf den Anforderungen (Anforderungs-Spezifikation) beginnt der funktionale Systementwurf. Hierbei werden die funktionalen Hauptkomponenten identifiziert und die Technologie-Bindung vorgenommen.

Im ersten Schritt wird der funktionale Systemkontext festgelegt bzw. aus den Anforderungen abgeleitet. Es wird ein Sonar-System entworfen, dass Unterwasser-Objekte mittels Schallwellen erkennen kann. Das Sonar-System wird als Modul der Anwendung, bekannte Schnittstellen werden als Ports des Moduls modelliert.

Die nächsten Schritte sind nicht in strikter zeitlicher Reihenfolge durchzuführen. Im nächsten Schritt werden die Funktionen des Sonar-Systems aus den funktionalen Anforderungen extrahiert und als Anwendungs-Module im Modul Sonar modelliert. Jede Funktion die in einer funktionalen Anforderung beschrieben ist kann beispielsweise ein Modul sein.

Dann erfolgt die Gruppierung von Funktionen zu sinnvollen Einheiten. Dadurch entstehen Module die funktionale Gruppen bilden. Das Sonar-System wird dabei in die funktionalen Gruppen Signalerzeugung, Signalverarbeitung sowie Management aufgeteilt. Um erzeugte Signale in Schallwellen zu wandeln wird eine weitere Funktion benötigt. Um Schall-Echos von Objekten als Signale verarbeiten zu können, müssen

sie wieder empfangen werden können. Dafür wird die funktionale Gruppe Transceiver erzeugt. Hier wird bereits ein erster Teil der Technologie-Bindung durchgeführt da (elektrische) Signale in Schallwellen (und umgekehrt) umgewandelt werden müssen.

Technologie-Bindung

In der nächsten Verfeinerung wird die Technologie-Bindung durchgeführt. Elemente unterschiedlicher Technologie werden hierbei durch Transformations-Tasks verbunden und die Verbindungen (im Allgemeinen Signale oder Signalflüsse) je nach gewählter Technologie und Art der Signale verfeinert.

Der System-Architekt oder Anwendungs-Entwickler legt fest, welche Teile der Anwendung in welcher Technologie realisiert werden sollen. Die Module Signalerzeugung, Signalverarbeitung und Management werden als digitale Elektronik realisiert. Um digitale Signale in Schallwellen umzuwandeln und empfangene Schall-Echos digital verarbeiten zu können, ist eine Transformation von der digitalen Elektronik in analoge Elektronik und elektromechanischen Teile notwendig. Daher wird das Modul Transceiver bzw. Transducer als analoge Elektronik realisiert. Die Abbildung 7 zeigt das Anwendungs-Modell nach der Technologie-Bindung.

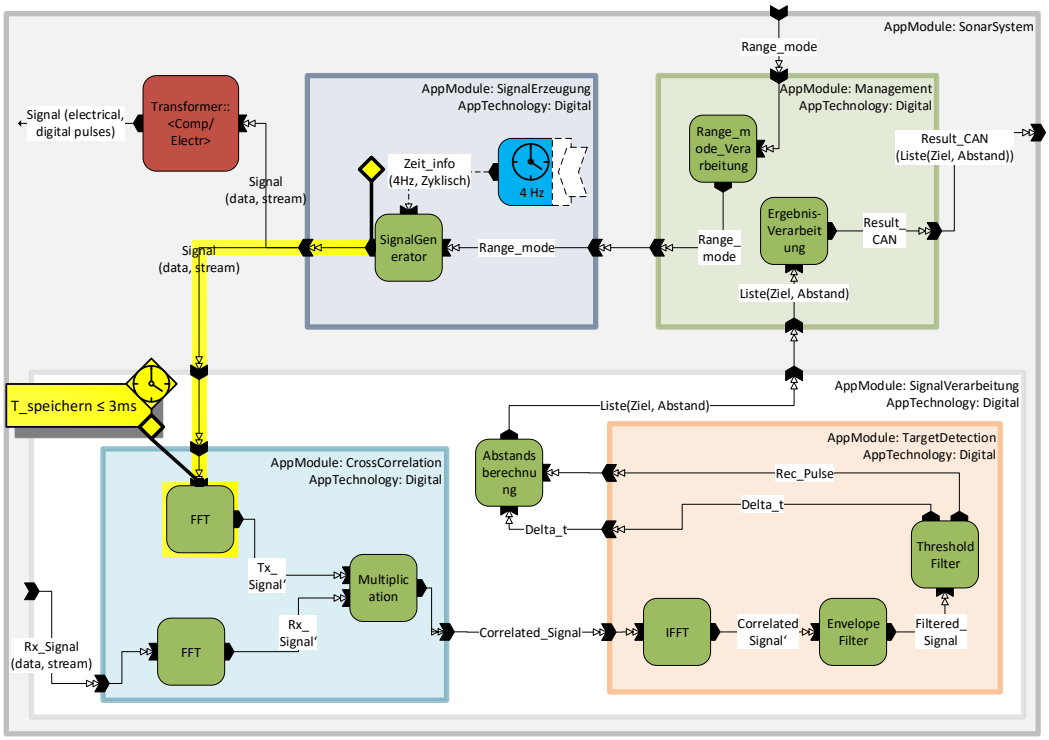

Abb. 7: Sonar-System nach Technologie-Bindung und mit Constraints

zeitliche Einschränkungen (Constraints)

Danach wird der funktionale Systementwurf durch die Annotation von Constraints an Anwendungs-Elemente (z.B. Signale) verfeinert und kann nun als ein Input in den Plattformentwurf dienen.

Um 200 kHz Signale sicher zu erfassen ist die doppelte Sampling-Rate notwendig. Im Sonar-System wird die Sampling-Rate der Analog-Digital-Wandlung auf 1 MHz festgelegt um Ungenauigkeiten im Low-Pass-Filter abzufangen und Signalanteile über 200 kHz Frequenz zu erfassen. Daher werden in diesem Schritt drei Zeit-Constraints annotiert, die sich aus den nicht-funktionalen Anforderungen ergeben:

- Die Sampling-Rate der Analog-Digital-Wandung,
- die maximale Zeit für die Objekterkennung und
- die gesamte Berechnungszeit des Systems .

Die maximale Zeit für die Objekterkennung ermittelt sich daraus, dass im Erkennungsbereich von bis zu 50 m ein Objekt-Echo innerhalb von 71 ms empfangen wird. Daher muss das Sonar-System diese Zeit warten bevor ein neues Signal ausgesendet werden kann. Innerhalb dieser Zeit muss das System ein Echo auswerten und die Objekterkennung durchführen können. Die Objekterkennung wird mittels Kreuzkorrelation durchgeführt. Dafür ist ein weiterer Zeit-Constraint notwendig, der sich aus dem bestehenden funktionalen Systementwurf und den Systemanforderungen ergibt. Das Fouriertransformierte ausgesendete Signal muss gespeichert werden bevor die ersten Echos empfangen werden. Die maximale Zeit bis zum Speichern ergibt sich daraus, dass der Erkennungsbereich bei 2 m beginnt. Innerhalb von 3 ms können die ersten Echos empfangen werden. Damit muss zusätzlich die maximale Zeit bis zum Speichern (T_Speichern) bei weniger als 3 ms liegen. Abbildung 7 zeigt wie dieser Constraint im funktionalen Systementwurf annotiert wird.

Plattformentwurf

In unserer Entwurfsstudie wird für den Plattformentwurf eine bestehende Hardware-Plattform wiederverwendet. Sie ist als Plattform-Komponenten-Modell (siehe Abbildung 8) verfügbar und besteht aus sechs Plattform-Modulen, dem FPGA-Motherboard, dem FPGA-Extension Board, dem A/D-Converter Board, dem Transducer Interface, dem Transducer selbst und einem Spannungsversorgungs-Modul.

Hier wird erkennbar, dass die Prozessor-Elemente (PWM und NIOS) physikalisch durch den Avalon-Bus miteinander verbunden sind. Der Avalon-Bus dient auch als Verbindung zum CAN-Controller und den SPI-Controllern “AD-Controller1” und “AD-Controller2” die sich auf dem FPGA-Extension Board befinden. Die Prozessor-Elemente erfüllen unterschiedliche Aufgaben. Um sie logisch zu trennen werden bei der Erstellung des Plattform-Domänen-Modells der Hardware-Plattform drei Domänen erzeugt die die Prozessor-Elemente kapseln (siehe Abbildung 9). Damit wird auch die physikalische Trennung der Plattform-Module überwunden und relevante Ressourcen den logischen Domänen bereitgestellt werden.

Alle Hardware-Komponenten werden durch passende Ressourcen repräsentiert, beispielsweise werden die Speicher-Elemente MEM2 und MEM3 als Speicher-Ressourcen modelliert. Alle Ressourcen die von mehreren Domänen genutzt werden, sind als geteilte Ressourcen (shared resource) modelliert und in der Domänen-Sicht im Zwischenraum zwischen den nutzenden Domänen visualisiert.

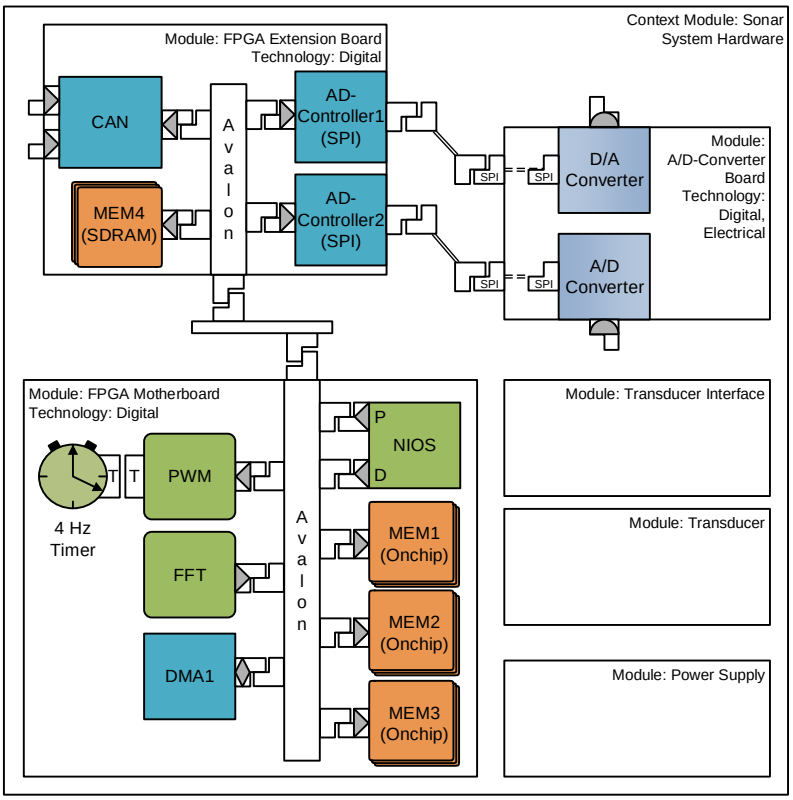

Abb. 8: Hardware-Komponenten der Sonar-System-Plattform

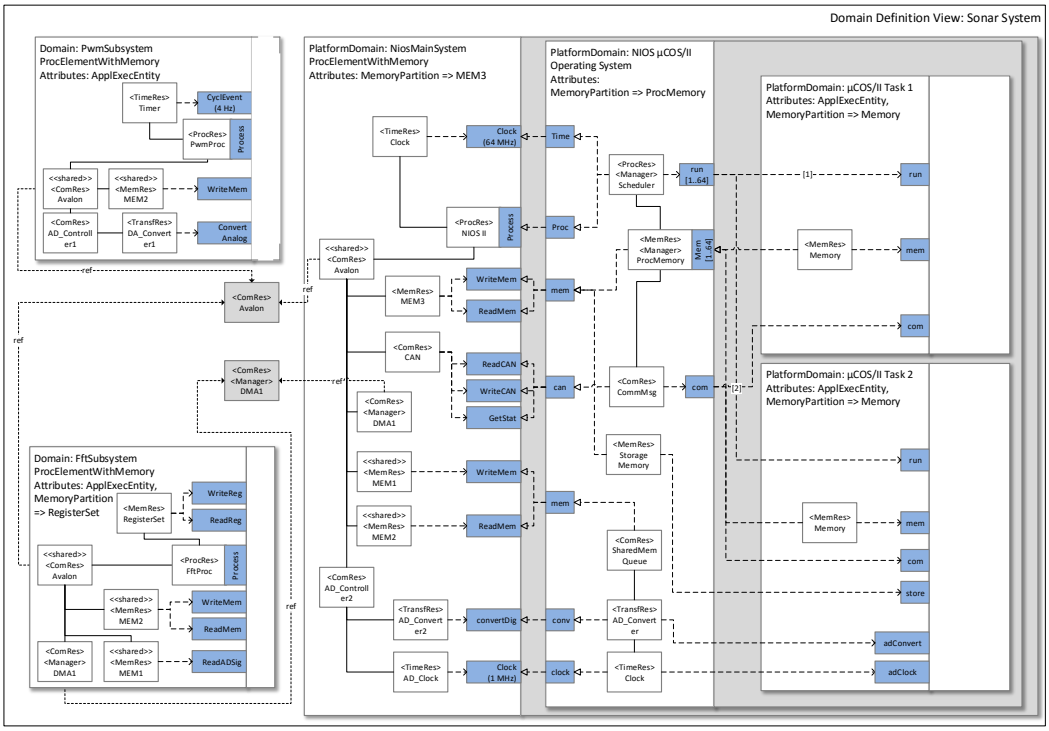

Abb. 9: Plattform-Domänen-Modell der Sonar-System-Plattform

Die Domäne NiosMainSystem die das Prozessor-Element NIOS kapselt, stellt eine Speicherpartition dar (mittels der Ressource MEM3). Im Gegensatz dazu ist PwmSubsystem nur eine Rechenzeitpartition.

Nachdem die Hardware-Domänen modelliert sind, werden höherwertige Domänen wie beim NiosMainSystem ein Betriebssystem modelliert. Hier werden 2 Tasks (Sub-Domänen) modelliert, die verschiedene Dienste anbieten. Ein Scheduler (Ressourcen-Manager) kann hier bis zu 64 Tasks bereitstellen (siehe Abbildung 9). Die Hardware-Domäne PwmSubsystem benötigt keine zusätzliche Software-Schicht, die Anwendungs-Elemente werden hier direkt an die Dienste der Hardware-Domäne gebunden.

Bindung von Sonar-Anwendung und Plattform

Nachdem sowohl die Anwendung als auch die Plattform modelliert wurden, kann die Bindung der Anwendungs-Elemente an die Plattform erfolgen. Abbildung 10 zeigt beispielhaft die Bindung von Tasks an Domänen und deren Ausführungs-Dienste. Die Bindung von Tasks an Domänen erfolgt, indem die Tasks im Ausführungs-Bereich der Domänen, der freien Fläche, platziert werden. Die Bindung von Kommunikation (Ports und Links) oder Spezial-Tasks wie Speicher und Zeit wird hier nicht dargestellt.

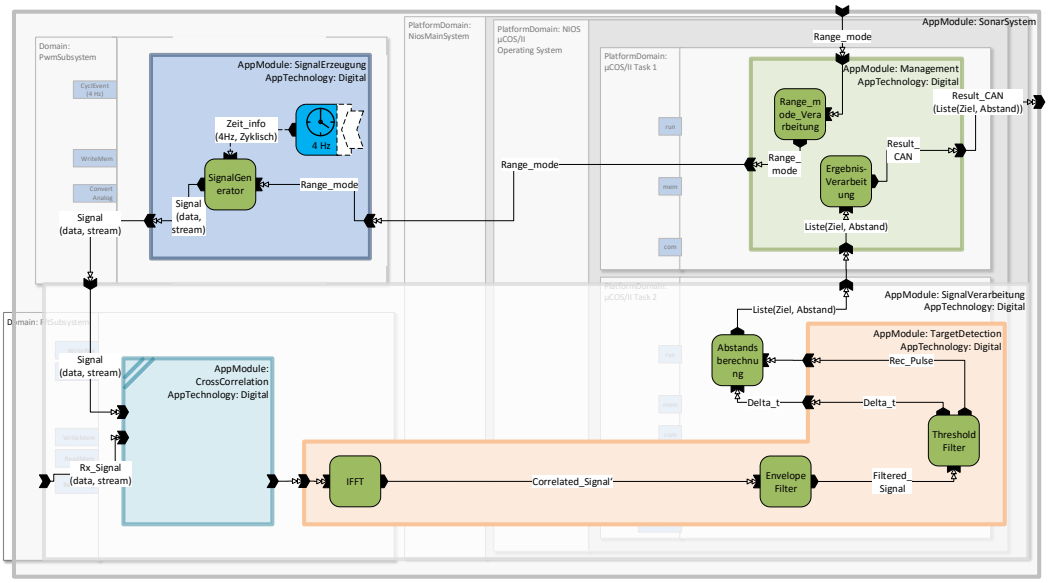

Abb. 10: Bindung von Tasks an die Plattform des Sonar-Systems

Die Plattform stellt eine Domäne FftSubsystem bereit, die speziell für die Berechnung von Fast-Fourier-Transformationen geeignet ist. Deshalb werden das Modul CrossCorrelation und der Task IFFT zur Ausführung an den Ausführungs-Dienst dieser Domäne gebunden. Der Anwendungs-Task SignalGenerator wird an die Domäne PwmSubsystem gebunden. Wie in Abbildung 10 dargestellt, werden die Tasks des Moduls TargetDetection auf 2 unterschiedliche Domänen aufgeteilt. Die Tasks EnvelopeFilter und ThresholdFilter werden durch die Domäne NiosMainsystem bzw. deren Sub-Domäne Task1 ausgeführt. Die Domäne Task2 bindet das Modul Management und ermöglicht damit eine nebenläufige Ausführung von Management und TargetDetection auf einem Prozessor.

Zusammenfassung

Wir zeigen mit unserer Entwurfsmethodik und dem zugehörigen Metamodell wie die Besonderheiten des dynamischen Verhaltens eingebetteter Software berücksichtigt werden können. Ein besonderes Augenmerk gilt dabei der Dynamik der Hardware/ Software-Schnittstelle.

In unseren weiteren Forschungsarbeiten wollen wir die Bindung von Anwendung und Plattform - insbesondere deren intuitive grafische Darstellung - analysieren und uns verstärkt um die Anbindung von Constraints an Anwendungs- und Plattform-Elemente kümmern. Als Basis dafür soll eine Implementierung der Entwurfsmethodik und des Metamodells erfolgen.

Literatur- und Quellenverzeichnis

[1] L. P. Carloni, F. De Bernardinis, C. Pinello, A. L. Sangiovanni-Vincentelli, and M. Sgroi. Platform-Based Design for Embedded Systems. In R. Zurawski, editor, *Embedded systems handbook*, Industrial information technology series, pages 1–26. Taylor & Francis, Boca Raton, 2006.

[2] C. Hausner and F. Slomka. Abstract Modeling of Embedded Systems Hardware. In *Proceedings of the 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2013)*, pages 251–258, 2013.

[3] I. Jacobson. *Object-oriented software engineering: A use case driven approach*. Addison-Wesley, Harlow, revised printing edition, 1998.

[4] G. Karsai. Modeling Cyber-Physical Systems: Challenges and Recent Advances, 05.09.2014.

[5] Object Management Group. *OMG Systems Modeling Language, version 1.2 (OMG SysML)*. Object Management Group, Needham, 1.2 edition, 2010.

[6] Object Management Group. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*. Object Management Group, Needham, 1.1 edition, 2011.

[7] Object Management Group. Unified Modeling Language (UML), 2013.

[8] A. L. Sangiovanni-Vincentelli. Defining Platform-based Design. 2002.

[9] F. Slomka, S. Kollmann, S. Moser, and K. Kempf. A Multidisciplinary Design Methodology for Cyber-physical Systems. In S. V. Baelen, S. Gérard, I. Ober, T. Weigert, H. Espinoza, and I. Ober, editors, *Proceedings of the 4th International Workshop on Model Based Architecting and Construction of Embedded Systems ACES-MB 2011*, CEUR Workshop Proceedings, 2011.

Autoren

Frank Slomka

Frank Slomka war von 2002–2007 Juniorprofessor an der Uni Oldenburg, seit 2007 ist Professor am Institut für eingebettete Systeme der Uni Ulm. Zu seinen Forschungsinteressen zählen der Entwurf eingebetteter Systeme insbesondere im Bereich der Nachrichten- und Energietechnik, die Entwurfsautomatisierung in der Mikroelektronik (EDA) und die Echtzeitanalyse und -modellierung. Er hat mehr als 50 Vorträge auf internationalen und nationalen Konferenzen gehalten und über 100 nationale und internationale Veröffentlichungen.

E-Mail: frank.slomka@uni-ulm.de

Internet: http://www.uni-ulm.de/in/es.html

Christian Hausner

Christian Hausner hat 2007 sein Diplom in Informatik an der HS Anhalt (FH) erworben und ist seit 2012 externer Doktorand am Institut für eingebettete Systeme der Uni Ulm. Seine Forschungsinteressen sind der Entwurf eingebetteter Systeme, die Entwurfsautomatisierung sowie die Modell-basierte und Modell-getriebene Entwicklung.

E-Mail: christian.hausner@gmx.de