Xpert.press

Die Reihe **Xpert.press** vermittelt Professionals in den Bereichen Softwareentwicklung, Internettechnologie und IT-Management aktuell und kompetent relevantes Fachwissen über Technologien und Produkte zur Entwicklung und Anwendung moderner Informationstechnologien.

Joachim Wietzke · Manh Tien Tran

Automotive Embedded Systeme

Effizientes Framework – Vom Design zur Implementierung

Mit 84 Abbildungen

Springer

Joachim Wietzke
Fachhochschule Darmstadt, Fachbereich Technische Informatik und Grundlagen der Informatik
Haardtring 100
64295 Darmstadt
wietzke@fbi.fh-darmstadt.de

Manh Tien Tran
Fachhochschule Kaiserslautern, Fachbereich IMST
Amerikastr. 1
66482 Zweibrücken
tran@informatik.fh-kl.de

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

ISSN 1439-5428
ISBN-10 3-540-24339-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-24339-7 Springer Berlin Heidelberg New York

Springer ist ein Unternehmen von Springer Science+Business Media
springer.de

Printed in The Netherlands

Satz: Druckfertige Daten der Autoren
Herstellung: LE-TeX Jelonek, Schmidt & Vöckler GbR, Leipzig
Umschlaggestaltung: KünkelLopka Werbeagentur, Heidelberg
Gedruckt auf säurefreiem Papier 33/3142/YL - 5 4 3 2 1 0

Vorwort

Perfektion ist nicht erreicht, wenn nichts mehr hinzugefügt werden kann, sondern wenn nichts mehr übrig ist, was man weglassen kann.

Antoine de Saint-Exupéry

Wer hat, als Entwickler in einem Team oder als Manager eines Teams, noch nicht erlebt, dass mit zunehmendem Wachstum der Gruppe die Effizienz immer schlechter wurde?

Eine Verdopplung der Gruppe etwa führte zwar annähernd zu einer Verdopplung der Fehler, nicht aber zur Verdopplung des produzierten Codes. Potenziert wurde das Problem, als dann nicht mehr alle Teammitglieder in einen Raum passten. Einige Meter Abstand waren dann schon so schlecht wie ein ganz anderer Standort.

Die übliche Abhilfemaßnahme war in der folgenden Zeit der Versuch, mehr Objektorientierung in die Implementierung zu bringen, mit dem Ergebnis, dass die Fehler komplizierter waren und langwieriger zu finden, und dass die Implementierungen wie ein Flickenteppich aus verschiedenen Programmiersprachen und Paradigmen erschienen.

Außerdem wurde der Speicher schnell knapp und die Performance schlecht. Geräte, die vorher keine Schwierigkeiten machten, zeigten nach der OOP-Umstellung bei gleichem Feature-Umfang erhöhte und inakzeptable Startzeiten im zweistelligen Sekundenbereich und vielfache Speichergrößen.

Als Nächstes kamen dann Vorschläge für einheitliche Design-Regeln, Testumgebungen und vorgegebene Software-Lösungen, die von den wenigen Kollegen, die Objektorientierung studiert hatten, geschrieben werden sollten.

Schließlich setzte sich, oft zu spät, die Erkenntnis durch, dass es wohl einige Architekturvorgaben geben müsse, die so implementiert waren, dass sie nicht leicht umgangen werden könnten.

Banal war dann noch die Erkenntnis, dass die Kollegen, die das am besten konnten, dafür freigestellt werden müssten und nicht die, die dafür Zeit hätten und es gerne mal probieren wollten.

Gerade die Mitarbeiter wurden aber gebraucht, die dafür nie Zeit hatten, weil es ja die besten waren und deshalb überall alte Brände löschen mussten. Seltsamerweise waren es auch die mit dem besten Domänenwissen, nicht die Gurus der Objektorientierung.

Bis sich all diese Erkenntnisse durchgesetzt hatten, war dann das Projekt um Monate überzogen, einige Task-Forces gegründet und versandet oder das Projekt war beim Kunden bereits verloren.

Im Nachhinein wuchs dann auch die Erkenntnis, dass es hier eigentlich um die Entwicklung eines Frameworks ging, nicht nur um die Implementierung eines Projekts. Beim nächsten Mal würde man deshalb zunächst ein Framework entwickeln und später ein neues Projekt daraus ableiten. Die Effizienz des Frameworks würde messbar sein und gemessen werden. Die Projektmitarbeiter würden nur noch Kundenvarianten programmieren, alle wesentlichen Fundamente wären schon im Framework festgelegt.

Für dieses Framework wurde dann eine große Mannschaft über mehrere Standorte hinweg berufen, damit auch alles Expertenwissen einfließen konnte. Schließlich wurde ein Steering-Committee darüber gestellt, dem strategische Entscheidungen vorgelegt werden sollten.

Das Schicksal dieses Frameworks kennen Sie schon, es war zu groß, zu langsam und wurde nicht fertig. Es konnte auch nicht angewendet werden oder vielleicht nur für ein einziges unvermeidliches Projekt.

Wenn sich dann schließlich beim nächsten Versuch hoffentlich vier bis fünf Kollegen finden, ausgestattet mit Motivation und Expertenwissen ihrer Domäne, Gummibärchen, Cola, besten Arbeitsmitteln und ohne politisches Gremium darüber, dann gibt es Hoffnung auf eine gute Lösung. Wenn diese Pioniere dann ein kleines Nachschlagewerk brauchen, aus dem sie manche Erfahrungen oder neue Ideen schöpfen können, dann nutzt Ihnen, meine Damen und Herren, hoffentlich dieses Buch.

Viel Spaß bei der Arbeit!

Manh Tien Tran, Joachim Wietzke

Inhaltsverzeichnis

Die Listings

Abbildungsverzeichnis

1

Einleitung

Es gibt in der derzeitigen Fachliteratur einige Ausführungen über Embedded-Systeme. Ihre Schwerpunkte liegen oft auf den dafür verwendeten Betriebssystemen oder auf Applikationen in kompakter, nicht vernetzter Umgebung. Beide Autoren haben in den letzten Jahren die Einführung von modernen SW-Strukturen ins Auto miterlebt und mitgestaltet.

Die Ausführungen beziehen sich deshalb im Folgenden beispielhaft auf vernetzte Systeme im Auto. Im Moment ist der europäische Standard dafür ein optischer Bus, MOST-Bus genannt (media oriented system transport), [MOST].

Die Ausführungen sind aber allgemein genug, um auch für andere Vernetzungen verwendbar zu sein. Das Buch ist in zwei Teile gegliedert.

Teil 1:

Welche Bausteine zu einem Framework gehören, welche Dinge festgelegt werden müssen und welche Anforderungen an ein Framework gestellt werden, wird in diesem ersten Teil des Buches besprochen.

Insbesondere der erste Teil lässt sich auf viele Domänen der Embedded Software anwenden.

Da es vielen, auch erfahrenen SW-Ingenieuren nicht bekannt ist, wie sich ihre Programme im Speicher auswirken, beginnt der Teil 1 mit einigen Grundlagen dazu. Anschließend werden Grundkonzepte und Grundbausteine von Embedded Systemen erörtert.

Ein Schwerpunkt der Diskussion ist dabei das Speicherverhalten und die Startperformance. Grundmechanismen der Betriebssysteme werden erläutert und ausgemessen, um Speicherverwendung, Kommunikation und Synchronisationen effizient zu gestalten.

Dabei wird darauf geachtet, dass Implementierungen unter Linux, QNX und Windows lauffähig sind. Dieser Teil ist sowohl an Studierende als auch an Embedded Experten gerichtet.

Teil 2:

Im zweiten Teil des Buches wird eine prototypische objektorientierte Framework-Implementierung besprochen. Diese diente zur Erprobung besonders effizienter Synchronisations- und Kommunikationsansätze. Außerdem sollte das Framework besonders klein sein und gute Startzeiten haben. Die Vorsätze sind in dem kleinen Rahmen recht gut geglückt, ein Framework entsprechend der vorgestellten Erkenntnisse mit mittlerer Applikationsausstattung bewegte sich im Rahmen von 2,5 s Startzeit und einer Größe von einigen 100 kByte. Noch viel wichtiger betrug die Umsetzungszeit nur wenige Mannmonate.

Neue Ideen

In dieses Buch sind einige Konzepte eingeflossen, die den Autoren aus der Literatur noch nicht bekannt waren. Es sind dies:

- Flexible, aber einheitliche Konfiguration für Threads und Prozesse,
- dafür intensive Nutzung des Shared-Memories in Verbindung mit dem `fork`-Konstrukt und von Thread-specific-Data;
- Queue-Konstrukte mit Vorrichtungen gegen Prioritäts-Inversion;
- Komponenten-Klassen mit festem Set von Queues und Services;
- Hintergrundsysteme: Watchdog, Timer;
- spezielle FSM-Architekturen für Embedded Systeme;
- Lauffähigkeit auf QNX, Linux, Windows;
- const-Objekte im Code-Segment zum Dekodieren der MOST-Nachrichten zur Performance-Optimierung.

Eine Schwierigkeit bei der Erstellung war das Ziel, im Text verallgemeinert genug zu bleiben, um noch lesbar zu sein und doch gleichzeitig auch Quellcodes zu zeigen, damit sich der Leser ein substanzielles Bild machen kann, wie die Erklärungen gemeint sind und umgesetzt werden könnten.

Ein Versuch, dieses Dilemma zu lösen, war, die Quelltexte auf das Nötigste zu kürzen. Sämtliche Includes wurden weggelassen. Nach dem Vorbild von [Stev] wurden alle wichtigen Includes in einen Abschnitt im Anhang verlagert. Weiterhin wurden viele Programmzeilen durch „. . . “ ersetzt.

Alle Sourcen laufen prinzipiell auf QNX, Linux und Windows. Allerdings wurden auch hier einige ifdefs herausgekürzt, die nötig waren, um auf die Eigenheiten der Betriebssysteme einzugehen. Im Anhang finden sich dafür Hinweise.

Ohne dass ein ganzes Framework in allen Einzelheiten vorgestellt wird, kann der Leser sicher manche Erkenntnisse aus den Beispielen gewinnen, sie umsetzen oder eigene Implementierungen an den vorgestellten Implementierungen messen. Unter dem Link

www.fbi.fh-darmstadt.de/∼jwietzke/Das_Buch

können viele Quellen angesehen und heruntergeladen werden. Auch ein aktueller Stand eines studentischen Framework-Projektes wird dort ab Juli 2005 zu finden sein.

Anmerkungen zur Domäne und zur Sprache:

Ein Framework ist immer domänenspezifisch, auch viele SW-Muster (Pattern) sind domänenspezifisch, in unseren Beispielen spezifisch für Embedded Automotive Systeme. In jeder Domäne gibt es Fachbegriffe, die sich nur schwer übersetzen lassen.

Wir werden auf die Eindeutschung von englischen Begriffen verzichten, ein Thread wird nicht als Laufzeitfaden übersetzt, sondern bleibt, nach einer inhaltlichen Erläuterung, ein Thread. Auch die Frage der Groß- oder Kleinschreibung englischer Begriffe, die eventuell im Deutschen schon bekannt sind, wird systematisch ignoriert. Wir wählen, wann immer wir darauf achten, die Großschreibung oder die Kleinschreibung.

Auch die Frage, welche Sprache in den Kommentaren der Implementierung gewählt wird, soll ohne Bedeutung sein, wesentlich ist nur das Vorhandensein hoffentlich aufschlussreicher Kommentare.

Teil I

Bausteine eines Embedded Frameworks

2

Ein Infotainment-System

Ein typisches so genanntes Infotainment-System im KFZ besteht aus einigen miteinander vernetzten Komponenten. In unseren Beispielen und Bildern gehen wir von einer MOST-Vernetzung aus. (Der MOST-Bus ist ein optischer Bus, der für automobile Anwendungen definiert wurde. Die Details, die wir für unsere Software-Betrachtungen brauchen, werden wir an passender Stelle erörtern und verwenden bis dahin den MOST-Bus als Platzhalter für irgendeine Vernetzung).

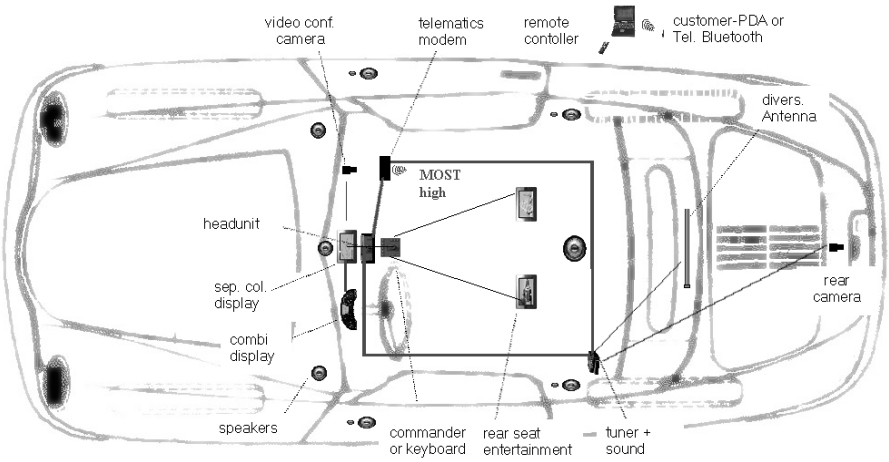

Abbildung 2.1. Beispiel eines Infotainment-Systems

Minimalumfang ist die Headunit, die als Bedieneinheit (HMI= human machine interface) für das System und ggf. auch für weitere Funktionalitäten des KFZ dient. Beispielsweise steuert sie auch Klimaanlage, Sitzverstellung und Diagnosefunktionen. Die Headunit ist die Kommandozentrale (HMI) des Multimedia-Systems. An ihr sind das Display und die meisten Bedienelemen-

te angeschlossen. In einem vernetzten System übernimmt sie normalerweise auch die Masterfunktionen im Netz bezüglich On/Off, Audiokanalvergabe, Display-Vergabe und die Prioritätsvergabe von Ressourcen. Sie befindet sich sichtbar im Armaturenbrett mit einem Display und einigen Bedienschaltern und beinhaltet den Hauptrechner, welcher die Masterfunktion im Netzwerk hat.

Zum System selbst gehören je nach Ausstattungsgrad:

Die Headunit	Display, Bedienelemente, Netzwerkmaster
Gateway	Brücke zu Kfz-Bussystemen wie CAN
Audiokomponenten	Tuner, Amplifier, Soundsystem, Kopfhörerverstärker, CD-Laufwerk, CD-Wechsler, DVD-Laufwerk, Kassettenlaufwerk
Videokomponenten	Video-/DVD-Laufwerk, TV-Tuner
andere Komponenten	Navigation mit Sensorik wie GPS, Radimpulse, Rückwärtsgang
Kommunikationskomponenten	Telefon/Telematik-Box
Rear Seat Entertainment	Unterhaltung für Fonds-Passagiere, zweite Slave-Headunit

2.1 Die Headunit

In der Headunit werden alle Funktionen des Netzwerkes gesteuert. Sie ist gleichzeitig der Bedienknoten für den Benutzer. Fähigkeiten, die einzelne angeschlossene Geräte haben, können in der Headunit miteinander zu neuen Funktionen kombiniert werden. Die Hinterleuchtung des Displays kann vom Fahrlicht abhängen, Fenster können geschlossen werden, wenn das Telefon klingelt, abhängig vom Zündschlüssel können Nutzerprofile mit unterschiedlichen Bediensprachen, Sitzeinstellungen und Sender-Favoritenlisten angeboten werden. Ersichtlich ist die Headunit der Netzwerkknoten mit der höchsten Funktionalität und Komplexität im Netzwerk, eine gewählte Software-Architektur muss sich mit vielen Eigenheiten der angeschlossenen Geräte auseinandersetzen. Hinzu kommt, dass die Hardware im Wesentlichen einem Embedded System entspricht, das heißt vor allem, dass Speicher knapp und Rechenleistung nicht allzu hoch sind. Da immer kritische Ruhe- und/oder Betriebsstromforderungen gestellt werden, sind die Hardware-Kenngrößen eine bis zwei Generationen hinter den allgemeinen PC-Trends. Zudem sind die Systeme nicht immer in ihrer Software nachladbar und müssen ca. 20 Jahre schadlos überdauern und wartbar bleiben.

Ein wenn auch kleiner Teil der Embedded Anforderungen an die Software beinhaltet Echtzeit-Anforderungen. Protokoll-Zeiten im MOST-System

müssen eingehalten werden, Positionsgenauigkeiten der Navigation, Reaktionszeiten in der Bedienung und schnelle Startzeiten der Systeme (2 sec).

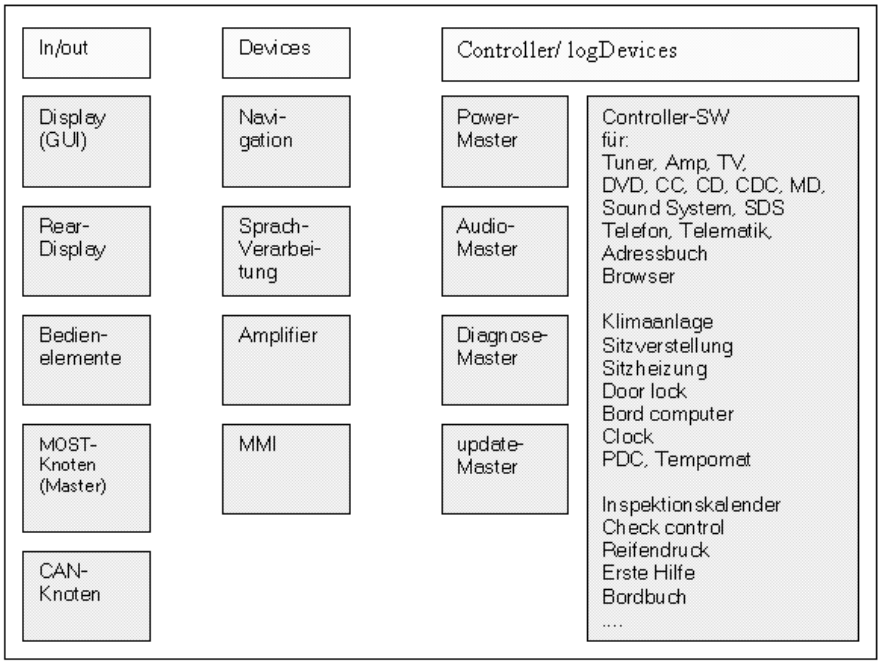

Abbildung 2.2. Beispiel eines Funktions-Blockbilds einer Headunit

2.2 Geräte/Devices im System

Devices sind Geräte mit abgegrenzter Funktion. Wir werden später sehen, dass jedes Device einen eigenen Namen im Netzwerk hat, beispielsweise sind Tuner und Amplifier einzelne Devices, auch wenn sie sich tatsächlich im gleichen Gehäuse auf der gleichen Hardware befinden können.

Sie sind normalerweise über eine Vernetzung an der Headunit angeschlossen, durch die heutige Hochintegration gibt es aber auch Devices, die im selben Gehäuse wie die eigentliche Headunit verbaut sind.

Es gibt die Unterscheidung in interne Devices und externe, d.h. vernetzte Devices.

Wir unterscheiden weiterhin in MOST- und non-MOST-Devices, also Devices, die dem Vernetzungsprotokoll folgen, und solche, die andere, normalerweise low-level APIs (Application Programming Interface) haben. Ein über das MOST-System angeschlossener DVD-Player wäre demnach ein externes

MOST-Device; die in der Headunit integrierte Navigation wäre ein internes MOST-Device, der DVD-Mediaplayer, der über ein ATAPI-Interface angesprochen wird, wäre ein internes non-MOST-Device.

2.3 Begriffsklärung: Was ist ein Framework

Ein Framework ist normalerweise nicht explizit nötig bei kleinen Arbeitsgruppen, die ein einmaliges Projekt ohne geplante Wiederverwendung realisieren. Alle Regeln, Schnittstellen und Abstraktionen geschehen auf Zuruf, Dokumentation ist eher schwach ausgeprägt.

Sobald größere Gruppen, eventuell sogar verteilt über Standorte, an einem oder mehreren Projekten arbeiten, müssen Regeln und Lösungen festgelegt werden, die im Framework niedergelegt werden.

Das Framework legt fest:

- Design Rules, Namenskonventionen, do's und don'ts;
- die Definition der Komponenten als Threads oder Prozesse;
- die Kommunikation zwischen Komponenten;
- Speicherdefinitionen, viele effiziente Mechanismen für Embedded Systeme;
- Kapselung des Expertenwissens bezüglich Vernetzung, OS-Spezialitäten, Prozessverwaltung, Prozesskommunikation, On/Off ... und damit Trennung von Schnittstellen und Implementierungen;
- Kapselung und Trennung von Information/Daten/Zuständen und algorithmischer Implementierung;
- Hardware-Abstraktionen;
- OS-Abstraktionen;
- Portabilitätsdefinition;
- Basisklassen und Patterns für Strukturen, Controller, Proxies, Treiber, Anzeigen;
- Header, Interfaces, Beispiele;
- Verwendbarkeit: Wartbarkeit, Erweiterbarkeit, Restrukturierbarkeit, Portabilität;
- Interoperabilität;
- Effizienz;
- Testbarkeit;
- Wiederverwendbarkeit;
- das Framework legt Performance und Ressourcen grundsätzlich fest, Fehler, die hier gemacht werden, können durch spätere Korrekturen im Handwerk nicht mehr verbessert werden. Normalerweise werden solche Fehler erst kurz vor Abschluss eines Projektes entdeckt und können nicht mehr zufriedenstellend beseitigt werden. Üblicherweise haben Kunden in Folgeprojekten kein Verständnis für eine „Alles-neu-Entwicklung", weshalb sich die Fehler durch mehrere Generationen fortpflanzen.