

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 National-
bibliografie; 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

Dieses Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, des Vortrags, der Entnahme von Abbildungen und Tabellen, der Funksendung, der Mikroverfilmung oder der Vervielfältigung auf anderen Wegen und der Speicherung in Datenverarbeitungsanlagen bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Eine Vervielfältigung dieses Werkes oder von Teilen dieses Werkes ist auch im Einzelfall nur in den Grenzen der gesetzlichen Bestimmungen des Urheberrechtsgesetzes der Bundesrepublik Deutschland vom 9. September 1965 in der jeweils geltenden Fassung zulässig. Sie ist grundsätzlich vergütungspflichtig. Zuwiderhandlungen unterliegen den Strafbestimmungen des Urheberrechtsgesetzes.

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

© Springer-Verlag Berlin Heidelberg 2005
Printed in The Netherlands

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutzgesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften. Text und Abbildungen wurden mit größter Sorgfalt erarbeitet. Verlag und Autor können jedoch für eventuell verbliebene fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

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

1	Einleitung	1
----------	-------------------------	----------

Teil I Bausteine eines Embedded Frameworks

2	Ein Infotainment-System	7
2.1	Die Headunit	8
2.2	Geräte/Devices im System	9
2.3	Begriffsklärung: Was ist ein Framework	10
2.4	Fokus unserer Framework-Betrachtungen	11
3	Anforderungen an ein Embedded Automotive Framework	13
3.1	Verteilte Entwicklung	13
3.2	MVC, Organisationen	13
3.3	Speicheranforderungen	15
3.4	Startzeiten	15
3.5	Menüwechselzeiten	16
3.6	Konfiguration versus Dynamik	16
3.7	Datenfluss/Kontrollfluss/Zustandssteuerung	16
4	Betriebssysteme	17
4.1	Prozesse, Tasks, Threads	18
4.2	Der Scheduler	19
4.2.1	Präemptive Prioritätssteuerung	19
4.2.2	Round-Robin	19
4.2.3	Prioritäts-Inversion, Prioritäts-Vererbung	19
4.2.4	Task-Switch, Kontextwechsel	20
4.3	Zusammenspiel mit der MMU	20

5	Design-Rules und Konventionen	21
5.1	Namenskonventionen	22
5.2	Shared-Verzeichnisse, Libraries	22
5.3	Eigene Definitionen im Shared	22
5.3.1	Standard-Typen	22
5.3.2	Alignment	23
5.3.3	Little-Endian, Big-Endian	24
5.3.4	Zahlgrenzen	25
5.3.5	Const Definitionen	25
5.3.6	Eigene Assertions	26
5.3.7	Exceptions und Signale	27
5.3.8	Eigene Ausgaben	27
6	Speicherverwaltung	29
6.1	Statische Speicherverwaltung	31
6.2	Dynamische Speicherverwaltung	31
6.3	Start-Up-Strategien	33
6.4	Speicherbedarf und kleine Pittfalls	33
6.4.1	Welche Variable wo?	33
6.4.2	Alternative Strategie 1: Späte Konstruktion	36
6.4.3	Alternative Strategie 2: Späte Initialisierung	38
6.4.4	Alternative Strategie 3: Lokale statische Variablen	39
6.4.5	Speicherort von Konstanten	40
6.4.6	Speicherbedarf eines einfachen Objekts	43
6.4.7	Speicherbedarf eines einfachen Objekts mit virtueller Methode	44
6.4.8	Speicherbedarf eines Objekts einer abgeleiteten einfachen Klasse	46
6.4.9	Speicherbedarf eines Objekts einer abgeleiteten Klasse mit virtueller Methode	48
6.4.10	Speicherbedarf eines Objekts einer abgeleiteten Klasse mit virtueller Basis-Methode	50
6.4.11	Mehrfachvererbung und Pittfall	52
6.4.12	Speicherbedarf dynamischer Variablen	55
6.4.13	Schlechte Parameterübergabe	56
6.4.14	Probleme mit dem Kopierkonstruktor	57
6.4.15	Probleme mit dem Zuweisungsoperator	57
6.4.16	Schlechte Rückgaben	57
6.4.17	Padding-Probleme	59
6.4.18	Alignment-Probleme, Endian-Probleme	60
6.4.19	Speicherfragmentierung	60
6.4.20	Vektoren, STL	61

7	Embedded Speicherkonzepte, spezielle Muster	63
7.1	Statische Variablen	63
7.1.1	Reihenfolge der Konstruktoren	63
7.2	Quasi-statische Allokation aus festem Pufferspeicher	66
7.2.1	Anlegen eines statischen Puffers	67
7.2.2	Placement-new	67
7.2.3	Makros zur Allokation	68
7.3	Allokator	70
7.3.1	Statische Allokation	73
7.3.2	Dynamische Allokation mit temporärem Heap	75
7.3.3	Object-Pool-Allokation	78
7.4	Datencontainer	82
7.5	Gemeinsam benutzte Objekte (Shared-Objects)	82
7.6	Referenzzähler (reference counting)	83
7.7	Garbage-Collection	85
7.8	Die Captain-Oates Strategie	85
7.9	Speicherlimit	86
7.10	Partieller Fehler (Partial Failure)	87
7.11	Fazit	87
8	Prozesse und POSIX-Threads	89
8.1	Prozesse	90
8.1.1	Der Lebenszyklus eines Prozesses	90
8.1.2	Prozesszustände	91
8.1.3	Zombies	92
8.1.4	Bestandteile	92
8.1.5	Prozess mit fork erzeugen	93
8.1.6	Prozess mit exec erzeugen	94
8.1.7	Prozess mit spawn erzeugen	96
8.1.8	Prozessvererbung	97
8.1.9	Prozessattribute	97
8.1.10	Über Prozessgrenzen hinweg	98
8.2	Threads	99
8.2.1	Erzeugung von POSIX-Threads	99
8.2.2	Allgemeine Gestaltung eines Threads in eingebetteten Anwendungen	100
8.2.3	Thread-Attribute	101
8.2.4	Über Threadgrenzen hinweg	104
8.2.5	Thread-spezifische Daten	106
8.2.6	Thread-Träger und Thread-Objekte	108
8.3	Designentscheidung	118

9	Inter-Prozess-Kommunikationskanäle, IPC	121
9.1	Pipe	121
9.2	FIFO, Named-Pipe	125
9.3	Socket, local	125
9.4	Socket-Verbindung zwischen Prozessoren	127
9.5	Message-Queues	134
9.5.1	System V Message-Queues	134
9.5.2	POSIX Message-Queues	134
9.6	QNX-Message	137
9.7	Shared-Memory	137
9.8	Shared-Memory gegen Überschreiben schützen	144
9.9	Zeitverhalten	149
9.10	Designentscheidung und prinzipielle Implementierung	150
10	Gemeinsame Nutzung von Code	151
10.1	DLLs	151
10.2	Shared-Code	151
10.3	Designentscheidung und prinzipielle Implementierung	152
11	Synchronisierungsmechanismen für Prozesse und Threads	153
11.1	Semaphore	153
11.2	Unbenannter Semaphor	157
11.3	Benannter Semaphor	160
11.4	Mutex, rekursiver Mutex	161
11.5	Mutex-Guard	167
11.6	Signale	168
11.7	Bedingungsvariablen (Condition Variables)	168
11.8	Zeitverhalten	177
11.9	Deadlock, Livelock und Prioritätsinversion	177
11.10	Zusammenfassung	182
12	Kommunikation per Events	183
12.1	Wichtige Event-Typen	184
12.1.1	Signal-Events	184
12.1.2	Events mit Cookies	185
12.1.3	Call-Events	185
12.1.4	Timer-Events	185
12.1.5	Change-Events (Notifikation - Das Hollywood-Prinzip)	186
12.2	Realisierungen von Events	186
12.2.1	Einfache Events	186
12.2.2	C-Strukturen	187
12.2.3	Funktionszeiger	189
12.2.4	Event-Klassen und Objekte	190

	12.2.5	Zeiger auf Event-Objekte verschicken	192
	12.2.6	Events am Beispiel von MOST	195
	12.3	Designentscheidung und prinzipielle Implementierung . . .	196
13	Event-Verarbeitung	197
	13.1	Queues	197
	13.1.1	Externe Queue	199
	13.1.2	Interne Queue	204
	13.1.3	System-Queue	205
	13.1.4	Queue mit verbesserter Inversions-Eigenschaft . . .	205
	13.2	Dispatcher	211
	13.2.1	Der System-Dispatcher	211
	13.2.2	Der lokale Dispatcher	212
	13.2.3	Signalisation im Dispatcher	213
	13.2.4	Registrierung im Dispatcher	213
	13.3	Synchrone Events	214
14	Zustandsautomaten	215
	14.1	Motivation	215
	14.2	Kommunikation	216
	14.3	Automaten	217
	14.4	Statechart-Elemente	217
	14.5	Probleme mit Statecharts und Ersatzlösungen	221
	14.6	Implementierung dynamischen Verhaltens	223
	14.6.1	Definition der Events	224
	14.6.2	Implementierung der Aktionen	225
	14.7	Implementierung der Statecharts	226
	14.7.1	Implementierung durch Aufzählung für Zustände und Unterzustände	226
	14.7.2	Zustandsmuster (State-Pattern)	231
	14.7.3	Verbesserte Aufzählungsmethode	240
	14.8	Implementierung nach Samek	246
15	Externer Kommunikationskanal: MOST-Bus	253
	15.1	Der synchrone Kanal	254
	15.2	Der asynchrone Kanal	255
	15.3	Der Kontrollkanal	255
	15.4	Geräte/Devices im Framework	255
	15.4.1	Logische Geräte, Modelle	257
	15.4.2	Methoden	258
	15.4.3	Properties	258
	15.4.4	MOST-Adressierung	259
	15.4.5	Adressierungsbeispiele	265
	15.4.6	MOST-Events, prinzipielle Dekodierung des Headers	266

15.4.7 MOST-Events, prinzipielle Dekodierung der Daten 268
 15.4.8 MOST-Konstanten 269
 15.4.9 MOST-Probleme 271

Teil II Das Framework

16 Das Framework 275

17 OS-Grundmechanismen 277

18 Komponentenarchitektur 279

18.1 Event-Handler und Event-Formate 281

18.2 Implementierung von Schnittstellen, Proxy und Handler . 285

18.3 Komponentenarchitektur 287

18.3.1 Komponentenkontext und Daten im
 Shared-Memory 288

18.3.2 Erzeugung der Kontexte 295

18.3.3 Main-/Admin-Task 300

18.3.4 Signale, Mechanismen über den Kontext 302

18.3.5 Komponenten-Dispatcher 308

18.3.6 Softtimer und Timer-Manager 310

18.3.7 Registrierung 318

18.3.8 Die Basisklasse `AComponentBase` und das
 Zusammenspiel mit der Umgebung 319

18.4 Zusammenfassung 327

19 Main-Dispatcher-Komponente 331

19.1 Dispatcher-Receiver 331

19.2 Dispatcher-Main-Thread 336

20 Modell-Komponenten, logische Geräte 341

20.1 Aufgaben des logischen Geräts 342

20.2 Was erwartet die HMI von einem logischen Gerät? 343

20.3 Zustände des Geräts 345

20.4 Gültigkeit der Daten 346

20.5 Kommunikationsanforderung 347

20.6 Struktur eines logischen Geräts 347

20.7 Datencontainer versus Event-Prinzip 347

20.8 Architektur eines Datencontainers 349

20.8.1 Struktur des Datencontainers 350

20.8.2 Lesezugriffe der HMI 356

20.8.3 Versenden eines HMI-Befehls 357

20.8.4 Menge der erlaubten High-Level-Aktionen 358

20.8.5 Abstrakte Klasse `ADataContainerAccessor` 359

20.9 MOST-Codec 363

20.9.1	Vorbereitung	363
20.9.2	Einfache Implementierung eines MOST-Decoders .	367
20.9.3	Objektorientierter Ansatz	370
20.9.4	Objekte zur Kompilierzeit	372
20.9.5	Konstante Objekte zur Dekodierung der MOST-Nachrichten	374
20.10	Zusammenfassung	387
21	HMI-Komponente	389
22	Persistenz-Controller und On/Off-Konzepte	391
23	Codegenerierung	395
23.1	Erstellen der XML-Eingabe-Dateien	395
23.2	Erzeugen des Quellcodes	397
23.3	Übersicht der erzeugten Dateien	398
23.4	Übersetzen des Quellcodes	404
24	Sonstige Aspekte	405
24.1	Internes non-MOST-Device, Beispiel Mediaplayer	405
24.2	Internes MOST-Device, Beispiel Tastatur	405
24.3	Treiber im Framework	406
24.4	Einfaches Debugging-Konzept	406
24.5	Überlastverhalten des Systems	408
24.6	Anmerkungen zur Komplexität und zum Testing	409
Anhang	411
25.1	Timer und Zeitmessung	412
25.1.1	Sleep	413
25.1.2	Timeout	414
25.2	Socket	414
25.2.1	InetAddr	414
25.2.2	CSocketAcceptor	415
25.2.3	CSocketConnector	418
25.2.4	CSocketStream	420
25.3	Keyboard	422
25.4	Shared-Memory	424
25.5	CBinarySemaphore	432
25.6	Extern const	439
Literatur	441
Index	443

Die Listings

5-1	Standard-Typen und Konstanten, <code>Global.h</code>	23
5-2	Alignment-Makro	23
5-3	Makros zur Endian-Format-Anpassung	25
5-4	Max/Min-Typedefs für eigene Datentypen	25
5-5	Eigene Assertions	26
6-1	Globales Objekt gemäß Singleton-Pattern	35
6-2	Späte Konstruktion des globalen Singleton-Objekts	36
6-3	Späte Konstruktion des globalen Singleton-Objekts mit Mutex-Schutz	37
6-4	Singleton mit Doppel-Checking	38
6-5	Späte Initialisierung eines Singletons	39
6-6	Lokale statische Variablen	40
6-7	Struktur mit plain old Data	41
6-8	Struktur mit Konstruktor-Aufruf	42
6-9	Struktur mit Konstruktor im Code-Segment	42
6-10	Instanz einer einfachen Klasse A	43
6-11	Speicherbedarf eines Objekts einer einfachen Klasse mit virtueller Methode	45
6-12	Speicherbedarf eines Objekts einer einfachen abgeleiteten Klasse	47
6-13	Beispiel für einen impliziten „Up-Cast“	49
6-14	Abgeleitete Klasse einer abstrakten Klasse	51
6-15	Mehrfachvererbung	52
6-16	Mehrfachvererbung, Up-Cast und Down-Cast, Testausgaben .	53
6-17	Beispiel-Interface in C++	55
6-18	Speicherbedarf für <code>int</code> - und <code>char</code> -Variablen unter VC++ . . .	56
6-19	Schlechte Rückgabe, Beispiel 1	57
6-20	Schlechte Rückgabe, Beispiel 2	58
6-21	Schlechte Rückgabe, Beispiel 3	58
6-22	Schlechte Rückgabe, Beispiel 4	58
6-23	Bessere Rückgabe, Beispiel	59

6-24	Achtlos platzierte Member-Variablen	59
6-25	Zeitmessungen zu Arrays und Vektoren, [Klein]	62
7-1	Systemobjekt	65
7-2	Systemobjekt mit reserviertem Speicherbereich	66
7-3	Anlegen eines statischen Puffers	67
7-4	Nutzung des Puffers durch eine C-Funktion	67
7-5	Einsatz des Placement-new-Operators	68
7-6	Syntax des Placement-new-Operators	68
7-7	Eigenes Überladen des Placement-new-Operators, [Nob]	68
7-8	Alle Makros in <code>FWMemory.h</code>	70
7-9	Interface-Definition <code>IAAllocator.h</code>	72
7-10	Makros zur Allokation von Objekten	73
7-11	Der Basisklassen-Prototyp <code>CStaticAllocator.h</code>	74
7-12	Auszug aus <code>CStaticAllocator.cpp</code>	74
7-13	Vereinbarung eines Allocators	75
7-14	Aufruf der <code>Allocator</code> -Methode	76
7-15	Klasse <code>CTemporaryHeapAllocator</code> , Prototyp	77
7-16	<code>CTemporaryHeapAllocator</code> , Implementierung	78
7-17	Prototyp der Klasse <code>CObjectPool</code>	79
7-18	Implementierung der Klasse <code>CObjectPool</code>	81
7-19	Eigene Smart-Pointer-Klasse	85
8-1	Syntax von <code>fork</code>	93
8-2	Beispiel für Prozess-Erzeugung mit <code>fork</code>	94
8-3	Aufruf von <code>execlp</code>	96
8-4	Start eines POSIX-Threads	100
8-5	Thread-Vereinbarung unter QNX, mit Lesen der Attribute	104
8-6	Erzeugung eines Threads mit eigenen Attributen	104
8-7	Implementierung der TSD-Initialisierung	108
8-8	Zugriff auf TSD über Makro	108
8-9	Basis-Interface <code>IRunnable.h</code>	109
8-10	Prototyp der Thread-Klasse <code>CThread.h</code>	112
8-11	Instantiierung eines <code>CThread</code> -Objekts	112
8-12	Instantiierung als Main-Thread	112
8-13	<code>CThread</code> -Implementierung	117
8-14	Kontextwechsel-Messungen	118
9-1	Pipe erzeugen, Implementierung der Sendeseite	122
9-2	Pipe schreiben, lesen, schließen, Prototypen	122
9-3	Erzeugen eines Socket-Paares, Prototyp	126
9-4	Socket-Applikation, mit <code>select</code> -Implementierung	127
9-5	Prototyp für Socket-Wrapper <code>CSockStream.h</code>	128
9-6	Implementierung des Socket-Wrapper <code>CSockStream.cpp</code>	129
9-7	Header-Datei des Connector-Wrappers <code>CSockConnector.h</code>	129
9-8	Implementierung des Connector-Wrappers <code>CSockConnector.cpp</code>	130
9-9	Header-Datei des Acceptor-Wrappers <code>CSockAcceptor.h</code>	130

9-10	Internet-Adress-Struktur <code>sockaddr_in</code> in <code><netinet/in.h></code> ..	131
9-11	Implementierung des Acceptor-Wrappers <code>CSockAcceptor.cpp</code>	132
9-12	Wrapper zum Setzen der Socket-Adresse	133
9-13	Verwendete <code>#includes</code>	133
9-14	Klasse <code>CMessageQ</code>	136
9-15	Erzeugen Shared-Memory	138
9-16	Einblenden des Shared-Memory in den Speicherbereich	139
9-17	Linux Kommandos für Shared-Memory	140
9-18	QNX Shared-Memory anlegen	140
9-19	Schreiben ins QNX Shared-Memory	142
9-20	Lesen aus dem Shared-Memory	143
9-21	<code>CMemoryProtector.h</code>	145
9-22	<code>CMemoryProtector.cpp</code>	145
9-23	Beispiel-Klasse <code>CProtectedSharedMemory</code>	148
9-24	Demonstration des Shared-Memory-Schutzes	148
11-1	Semaphor und die Zugriffsfunktionen <code>V(s)</code> und <code>P(s)</code>	155
11-2	C-Funktionen für unbenannte Semaphore	158
11-3	Synchronisierter Produzenten-/Konsumenten-Zugriff zweier Threads	160
11-4	Erzeugen und Schließen eines benannten Semaphors	161
11-5	Erzeugung einer Mutex-Variablen	162
11-6	Lesen der Thread-Attribute	163
11-7	Implementierung eines Mutex-Schutzes für eine Zugriffsfunktion	164
11-8	Klasse <code>CMutex</code>	166
11-9	Guard-Klasse	168
11-10	Typische Bedingungsvariablen-Anwendung	171
11-11	Code-Beispiel für die Freigabe einer Bedingungsvariablen ...	172
11-12	<code>CBinarySemaphore.h</code>	173
11-13	Objektorientierte Implementierung <code>CBinarySemaphore.cpp</code> .	176
12-1	Enum für einfache Events	187
12-2	Dispatcher für Enum-Events	187
12-3	Event als C-Struktur	188
12-4	Event als C-Struktur mit Union	188
12-5	Event als C-Struktur mit Funktionszeiger und Parameter ...	189
12-6	Implementierung der <code>dispatch</code> -Methode der Struktur	189
12-7	Event als Klasse <code>CMessage</code>	191
12-8	Event-Klasse mit eigenen Methoden und Attributen	192
12-9	Verwendung von Event-Pool	193
12-10	Event-Basisklasse	195
12-11	Beispiele für MOST-Events gemäß MOST-Spezifikation ...	195
13-1	Queue-Klassendefinition, Prototyp	198
13-2	Eine Queue-Implementierung	201
13-3	Produzent-Konsument mit Counting-Semaphor	203
13-4	Teilimplementierung einer internen Queue	205

13-5	Die Queue-Klasse <code>CCommQueue.h</code> , Prototyp	206
13-6	Basisklasse <code>CCommQueue.cpp</code> , Implementierung	211
13-7	Lokaler Dispatcher	213
13-8	Registrierung und Deregistrierung in <code>CDispatcher</code>	214
14-1	Methoden zum Beispielautomaten	225
14-2	Exit-Methoden von <code>CFSM1</code>	228
14-3	Die <code>on_Entry</code> -Methoden	229
14-4	Beispiel von Event-Handler-Methoden	230
14-5	Basisklasse <code>CBasisState</code>	233
14-6	Der <code>inEntry</code> Oberzustand A	235
14-7	Klasse für SuperState A, Implementierung	236
14-8	Zustandsmaschine <code>CFSM3</code> , Header	236
14-9	Konstruktor von <code>CFSM3</code>	237
14-10	Reaktion von <code>CFSM3</code> auf <code>EV_1</code>	237
14-11	Zustandsmaschine <code>CFSM3</code> , Ereignisverteilung	238
14-12	Wirkung von Event als Parameter	239
14-13	Verbesserte Aufzählungsmethoden, Klassendeklaration	241
14-14	Dispatch-Methode von <code>CSuperStateA</code>	242
14-15	Verbesserte Aufzählungsmethode, Implementierung	245
14-16	Definition eines Funktionszeigers auf eine <code>CFSM</code> -Methode	248
14-17	Makro für den Funktionszeiger	248
14-18	Basisklasse <code>CFSM</code>	249
14-19	SuperState als abgeleitete Klasse	250
14-20	FSM <code>CFSM5</code> mit Funktionen als Zuständen	251
15-1	MOST-Event, Struktur	259
15-2	Beispiel eines <code>NetBlock</code> -Aufrufs	260
15-3	Aufbau der zentralen Registry	264
15-4	Beispiele für die Adressierung gemäß MOST	265
15-5	MOST-Sequenz	266
15-6	Die Sammlung vieler Most-Konstanten, <code>MOSTConst.h</code>	271
18-1	Schnittstelle <code>IMessageHandler</code>	282
18-2	Ausschnitt aus <code>CMessage.h</code>	285
18-3	Der Kontext-Header	291
18-4	Beschreibung des Komponentenkontexts, <code>CComponentContext.h</code>	293
18-5	Die <code>createComponentContext</code> -Methode der Klasse, Implementierung	295
18-6	<code>sDescriptionTable</code> für die Erzeugung von Kontexten	297
18-7	Klasse zur Erzeugung und Verwaltung der Kontexte	298
18-8	Code zur Erzeugung von Kontexten und Datencontainer	300
18-9	Pseudocode für <code>main</code>	300
18-10	Auszug aus der <code>run</code> -Methode des Admin-Tasks, Watchdog-Loop	302
18-11	Auszug aus der <code>run</code> -Methode des Admin-Tasks, Watchdog-Trigger	303

18-12	Auszug aus der <code>run</code> -Methode einer Komponente	303
18-13	<code>handleTick</code>	306
18-14	Auszug aus der Klasse <code>CDispatcher</code>	310
18-15	Klasse für Softtimer	311
18-16	Timer-Manager, <code>CTimerManager.h</code>	313
18-17	Auszug aus <code>CObserverRegistry.cpp</code>	319
18-18	Headerfile der Klasse <code>CComponentContext</code>	321
18-19	Komponentenservice-Klasse, <code>CComponentServices.cpp</code>	323
18-20	<code>init</code> -Methode, Klasse <code>AComponentBase</code>	324
18-21	Hilfsklasse <code>CSystemEventHandler</code>	325
18-22	<code>cleanup</code> -Methode, Klasse <code>AComponentBase</code>	325
18-23	<code>run</code> -Methode, Klasse <code>AComponentBase</code>	326
19-1	Start des <code>MostDispatcher Receiver</code> -Threads	333
19-2	Zeiger auf Struktur <code>MostHandle</code>	334
19-3	<code>MOST</code> -Empfänger-Klasse <code>CMostdispatcherReceiver</code>	335
19-4	<code>init</code> -Methode von <code>Dispatcher-Main-Thread</code>	336
19-5	Reaktionen von <code>Dispatcher-Main-Thread</code>	337
19-6	Reaktion auf <code>Lock</code> - und <code>Unlock</code> -Events	337
19-7	Verteilung der <code>MOST</code> -Nachrichten im <code>System-Dispatcher</code>	340
20-1	Datencontainer-Struktur für <code>AudioDiscInfo</code>	352
20-2	Struktur <code>StringBuffer</code>	353
20-3	Struktur <code>DataBuffer</code>	353
20-4	<code>MOST</code> -Datencontainer für <code>AudioDiscPlayer</code>	355
20-5	Einige <code>get</code> -Methoden für <code>AudioDiscInfo</code>	356
20-6	Implementierung der <code>get</code> -Methoden für <code>AudioDiscInfo</code>	357
20-7	Abstrakte Klasse <code>ADataContainerAccessor.h</code>	361
20-8	Teilimplementierung <code>CAudioDiscPlayerDCAccessor</code>	361
20-9	<code>Enum</code> -Index für den Datenzugriff	362
20-10	Adapterklasse zum Lesen einer <code>MOST</code> -Nachricht	365
20-11	Einige Methoden aus <code>CMOSTInStream</code>	367
20-12	Dekodierung	369
20-13	Weitere Dekodiermöglichkeit	371
20-14	Objekte zur Kompilierzeit	372
20-15	Objekte im <code>Code</code> - und <code>Datensegment</code>	373
20-16	<code>MOST</code> -Datentypen	374
20-17	<code>CTypeInfo</code> und <code>CMOSTEnum</code> zur Beschreibung der <code>MOST</code> -Datentypen	375
20-18	Die Methode <code>invokeRead</code>	376
20-19	Hilfsstruktur	377
20-20	<code>ModelTypeEnum</code>	377
20-21	Basis-Struktur <code>CModelInfo</code>	378
20-22	Beschreibungen von <code>MOST</code> -Properties, Prototyp	380
20-23	Polymorphie-Ersatz für <code>CModelInfo</code>	381
20-24	Modell für <code>Array of Record</code> , <code>CArrayRecordProp.h</code>	381
20-25	<code>CArrayRecordProp</code> -Struktur für <code>AudioDiscInfo</code>	382

20-26	Vom <code>mType</code> abhängige Lesefunktion	383
20-27	Lesefunktion für Array of Record	386
20-28	Lesen von Enum	386
22-1	Persistenz-Manager, Prototyp	394
23-1	Schnittstellenbeschreibung für die Code-Generierung	396
23-2	Komponenten für den Code-Generator	397
23-3	Generierung des Projekts	397
23-4	Starten einer Komponente als Thread	398
23-5	Starten einer Komponente als Prozess am Beispiel MostDispatcher	398
23-6	Der Watchdog, der innerhalb <code>main()</code> ausgeführt wird	399
23-7	Schleife zum Erzeugen der Kontexte	400
23-8	Projekt mit Amp-, Debug- und Dispatcher-Komponente	401
23-9	<code>DebugConfig.cpp</code> am Beispiel eines Audioamplifiers	401
23-10	Beispiel für den Audioamplifier	402
23-11	<code>AudioamplifierPort.cpp</code> , <code>AudioamplifierPort.h</code>	403
23-12	<code>AudioamplifierHandler.cpp</code> und <code>AudioamplifierHandler.h</code>	404
24-1	Implementation of <code>CCommQueue</code>	408
25-1	Timer und Zeitmessung	413
25-2	OS-Abhängigkeiten für eine <code>sleep</code> -Methode	413
25-3	Abhängige <code>includes</code> für <code>Sem.take</code> mit <code>timeout</code>	414
25-4	OS-abhängige <code>includes</code> für <code>CInetAddr.h</code> , Header	415
25-5	OS-abhängige Implementierungen in <code>CInetAddr</code>	415
25-6	OS-abhängige <code>includes</code> und Definitionen für <code>CSockAcceptor</code> , Header	416
25-7	OS-abhängige Implementierungen für <code>CSockAcceptor</code>	417
25-8	OS-abhängige <code>Includes</code> für <code>CSockConnector</code> , Header	418
25-9	OS-abhängige Implementierungen für <code>CSockConnector</code>	420
25-10	OS-abhängige <code>Includes</code> für <code>CSockStream</code> , Header	421
25-11	OS-abhängige Implementierungen für <code>CSockStream</code>	422
25-12	OS-abhängige Terminierung von Socket	422
25-13	Tastatur- und Keyboard-Eingaben	423
25-14	Abhängige Definitionen für Shared-Memory	424
25-15	<code>PosixShm.h</code> für WIN	426
25-16	<code>PosixShm.cpp</code> für WIN	431
25-17	OS-abhängige <code>CBinarySemaphore.h</code>	433
25-18	OS-abhängige Implementierungen für <code>CBinarySemaphore</code> ...	438
25-19	Abhängige externe <code>const</code> -Definitionen	439

Abbildungsverzeichnis

2.1	Beispiel eines Infotainment-Systems	7
2.2	Beispiel eines Funktions-Blockbilds einer Headunit	9
3.1	MVC-Architektur am Beispiel Tuner	14
5.1	Big/Little-Endian	24
6.1	Speichermap, siehe auch [Mad]	30
6.2	Heap- und Stack im Adressraum	32
6.3	Speicher-Layout eines Objekts mit virtueller Methode	45
6.4	Speicher-Layout eines B-Objekts mit virtueller Methode	47
6.5	Impliziter „Up-Cast“, Speicher-Layout	50
6.6	Speicher-Layout zweier virtueller Klassen-Objekte	51
6.7	Mehrfach-Vererbung und falscher Cast	53
7.1	Abhängige Objekte	64
7.2	Klassendiagramm Allocator	73
7.3	Statisches Allocator-Objekt	75
8.1	Speicherlayout nach fork	95
8.2	Nachrichtenkopplung	98
8.3	Speicherkopplung	105
9.1	Pipe-Erzeugung, fork und Schließen der Pipe-Enden	123
9.2	Kommunikation mit Pipes	124
9.3	Message-Queue Funktionen	135
9.4	QNX-Message-Funktionen	137
9.5	Bildliche Darstellung eines Shared-Memory Bereichs	138
9.6	Zeitmessungen der unterschiedlichen IPC-Methoden, [Klein] . .	149
11.1	Zustandsdiagramm der möglichen Semaphore-Zustände	155
11.2	Semaphore zwischen zwei Prozessen	157

11.3	Zustandsdiagramm der möglichen Mutex-Zustände	162
11.4	Zeitverhalten der Synchronisationsmethoden	177
11.5	Deadlock-Beispiel	178
11.6	Begrenzte Inversion	179
11.7	Prioritätsinversion	180
12.1	Die Nutzung eines Pools zur Event-Erzeugung	192
13.1	Symbolische Queue	197
13.2	Reale Queue	199
13.3	Handshake zwischen Produzenten und Konsument	208
14.1	Zustandsloses Objekt	215
14.2	Zustandsbehaftetes Objekt	216
14.3	Bedingter Zustandsübergang	218
14.4	Zustandsübergangstabelle	218
14.5	Aktionen im Zustandskontext	219
14.6	Zustandshierarchie	219
14.7	Übergänge in der Zustandshierarchie	220
14.8	Default-Start- und End-Zustände	220
14.9	Statechart mit Gedächtnis	221
14.10	Statechart mit tiefer Historie	222
14.11	Statechart mit Parallelismus	222
14.12	Zustandsübergänge in Hierarchie	223
14.13	Beispiel eines Statecharts	224
14.14	Abgeändertes Statechart	232
14.15	Klassendiagramm des Zustandsmusters	233
14.16	Die FSM im Oberzustand	243
14.17	Klassendiagramm von CFSM5	247
15.1	MOST-Ring mit Komponenten	254
15.2	Mehrere Function-Blocks in einem Gerät	256
15.3	MOST-Funktionsaufruf, Beispiel	260
15.4	Beispiel einer DeviceID-Tabelle	260
15.5	Beispiel einiger FblockID-Tabellen	261
15.6	Verwendung der FktIDs am Beispiel eines Multimedia-Changers	262
15.7	OPTypes für Methoden und Eigenschaften	262
15.8	Datentypen in MOST, aus [MOST]	263
15.9	Zentrale Registry	264
15.10	Beispiel einer Notifikation	265
18.1	Tasks der Headunit	281
18.2	Klassendiagramm zu einer Komponente	289
18.3	Kontext-Header der Komponenten	291
18.4	Kontext einer Komponenten	294
18.5	Eingebauter Watchdog-Mechanismus	304

18.6	Tickservice	305
18.7	Mechanismus zum Auffordern eines Zustandswechsels	307
18.8	Empfang eines Tick-Events, Timer-Manager und -Client	317
18.9	Beispiel einer Komponente (logisches Tuner-Device)	328
19.1	Main-Dispatcher-Komponente	332
19.2	MOST-Adapter	333
20.1	Blockbild des logischen Geräts	341
20.2	Anforderungen der HMI an das logische Gerät	345
20.3	Zustände des logischen Geräts	346
20.4	Daten-Container für die HMI	349
20.5	Container für Soll- und Ist-Daten	350
20.6	Klassendiagramm des Datencontainers	351
20.7	Struktur des Datencontainers	352
20.8	Adapter für die HMI zum Lesen des Datencontainers	356
20.9	Versenden eines HMI-Befehls	357
20.10	Expertenwissen in der Gerätesteuerung	359
20.11	Hierarchie der elementaren MOST-Datentypen	374
20.12	Vier Varianten von <code>CModelInfo</code>	380
21.1	HMI-Komponente	389