WissenHeute Jg. 58 4/2005

Das Thema im Überblick

In welcher Form eine Software entwickelt wird hängt im Einzelfall von der konkreten Anwendung ab, die dahinter steckt. Ein in der Designphase entstandener Entwurf wird während der Implementierung in eine Programmierung umgesetzt, was besondere Sorgfalt verlangt, damit nachträgliche Änderungen vermieden werden. Im Beitrag werden Empfehlungen für die Nutzung verschiedener Systemfunktionen beschrieben, die für eine Programmierung in Frage kommen, wobei vorrangig die Komponenten Kommunikation und Endgerätekontrolle betrachtet werden. Darüber hinaus werden Mechanismen zur Fehlererkennung und -behandlung vorgestellt.

Besonderheiten der Anwendungsentwicklung für die mobile Nutzung – ein Leitfaden, Teil 4.

Die Autoren

Dipl.-Ing. Stefanus Römer arbeitet als Produktmanager bei T-Mobile International, wo er insbesondere für das Produkt Mobile IP VPN sowie für mobile Intranet-Access-Lösungen zuständig ist.

Dipl.-Ing. Marcus Freitag arbeitet als Senior Consultant in der DIALOGS Software GmbH in München.

Der in Heft 11/2004 begonnene Beitrag wird mit dem vierten Teil zum Thema Implementierung fortgesetzt. In der Phase der Implementierung wird der Entwurf aus der Design-Phase in eine Programmierung umgesetzt. Dabei werden die Komponenten und Interaktionen, die in dieser Phase definiert wurden, unter Auswahl geeigneter Ressourcen des Betriebssystems in Prozeduren, Daten und Funktionsaufrufe übertragen. Eine nachträgliche Änderung, die eine grundlegende Anpassung der Abläufe nach sich zieht, ist dann mit hohem Aufwand verbunden.
In den nachfolgenden Abschnitten werden zunächst die grundlegenden Konzepte vorgestellt. Es werden Empfehlungen für die Komponenten Kommunikation und Endgerätekontrolle[25] (s. Bild 12) und wichtige Hinweise zur Fehlerbehandlung gegeben.

Grundlegende Konzepte

Eine mobile Anwendung benötigt den direkten Zugriff auf verschiedene Funktionen des Betriebssystems, um Aufgaben wie beispielsweise:

- Start und Beendigung von Prozessen oder Threads[26],
- Interprozess-Kommunikation[27],
- Datenkommunikation oder
- Endgerätekontrolle

[25] Siehe hierzu den Beitrag „Besonderheiten der Anwendungsentwicklung für mobile Nutzung – ein Leitfaden (Teil 3)“, WissenHeute Nr. 3/2005, S. 158 ff.

[26] Ein **Thread**, vom englischen „thread“ (Faden, Gedankengang, Gewinde, Strang), stellt die kleinste Einheit eines ausführbaren Programmcodes dar, der durch das Betriebssystem, nach einer entsprechenden Priorität, Rechenzeit (Ressourcen) zugeteilt wird.

[27] **Interprozess-Kommunikation:** Kommunikation zwischen Prozessen.

ausführen zu können. Diese Systemfunktionen werden vom Betriebssystem mit Hilfe spezieller Programmierschnittstellen, den Application Programming Interfaces (API) und Bibliotheken (Libraries) bereitgestellt. Von besonderer Bedeutung sind in diesem Zusammenhang die folgenden APIs:

- Win32-API
  Die zentrale Programmierschnittstelle mit Zugriff auf Kernel[28]-Funktionen des Betriebssystems.
- WinSock-API
  Die Kommunikationsschnittstelle für TCP/IP-Programmierung für den Zugriff auf Netzfunktionen und das Internet unter Microsoft Windows.
- RAS[29]-API
  Die Programmierschnittstelle zum Programmieren von Einwahlverbindungen und zur vereinfachten Endgerätekontrolle.

Die genannten APIs bieten dem Entwickler einen direkten Zugriff auf „hardwarenahe" Systemfunktionen, in dem sie eine Menge von abstrakten Operationen und Datentypen bereitstellen.

Im Gegensatz zu Bibliotheken, wie beispielsweise die Microsoft Foundation Classes (MFC) oder das NET Framework, kann der Entwickler mit diesen APIs systemnahe Programme mit voller Kontrolle über Kommunikations- und Hardware-Funktionen erstellen. Dies ist gerade bei der Erstellung der Endgerätesteuerung für mobile Anwendungen wichtig. Außerdem wird eine Übertragung auf andere Rechnersysteme ermöglicht und die Komplexität reduziert sich.

Es gibt viele weitere APIs oder Software Developer Kits (SDKs), die für die Programmierung der eigentlichen Anwendungskomponente benötigt werden. In diesem Beitrag wird lediglich auf die Komponenten „Kommunikation" und „Endgerätekontrolle" eingegangen, weil sich die Anwendungslogik in jedem einzelnen Projekt unterscheidet und somit nicht allgemein betrachtet werden kann. Ein weiterer Schwerpunkt liegt auf der Umsetzung der Komponentenarchitektur (s. Bild 12), wie sie im dritten Teil dieser Beitragsserie zum Thema Design vorgestellt wurde. Diese Architektur wird in der Implementierungsphase mit Hilfe von Prozessen und Threads umgesetzt, deren allgemeine Grundlagen nachfolgend vorgestellt werden.

Allgemeine Eigenschaften von Prozessen
Ein Prozess repräsentiert die zentrale Ausführungseinheit innerhalb eines Betriebssystems. Bei Windows Betriebssystemen verfügt jeder einzelne Prozess über einen virtuellen Adressraum von vier Gigabyte und mindestens einen primären Thread.

Ein Prozess wird genutzt, wenn eine bestimmte Aufgabe innerhalb der Programmausführung von anderen Komponenten getrennt werden muss. Ein Beispiel hierfür ist die Benutzerschnittstelle, die getrennt von der eigentlichen Kernanwendung und bei Bedarf in einer anderen Programmiersprache erstellt werden kann. Durch diese Art der Ausführung in einem eigenen Prozess ist gewährleistet, dass der Benutzer während einer aufwendigen Verarbeitung oder eines zeitraubenden Kommunikationsvorgangs nicht „blockiert" wird.

Ein weiteres Beispiel ist die Endgerätesteuerung einer mobilen Anwendung. Sofern diese Aufgabe von einem speziellen Prozess übernommen wird, ist sichergestellt, dass die Kernanwendung nicht beeinträchtigt wird, sobald eine schwer wiegende Störung mit dem Mobilfunkgerät oder dem Endgerätetreiber auftritt. Derartige Hardware-Fehler sind gerade bei frühen Firmware[30]-Releases nicht selten und müssen stets Berücksichtigung finden.

Allgemein lässt sich festhalten, dass ein separater Prozess immer dann Anwendung findet, wenn:

- eine Aufgabe von anderen Anwendungskomponenten getrennt werden muss oder soll,
- die Stabilität eines Endgerätes oder eines Endgerätetreibers nicht gewährleistet werden kann oder
- Komponenten in verschiedenen Programmiersprachen erstellt werden sollen.

Durch den Einsatz separater Prozesse ergeben sich allerdings auch Nachteile, die berücksichtigt werden müssen:

- Prozesse belegen mehr Systemressourcen. Daher sollte eine Anwendung stets so wenig Prozesse wie möglich nutzen. Kleinere Aufgaben können von Threads übernommen werden.
- Der Einsatz verschiedener Prozesse erfordert eine zusätzliche Interprozess-Kommunikation zur Steuerung des Anwendungsablaufs und zum Austausch von Daten.
- Systemressourcen, die von mehreren Prozessen genutzt werden, müssen mit Hilfe von Synchronisationsmechanismen vor Konflikten geschützt werden.

Allgemeine Eigenschaften von Threads
Neben Prozessen gehören Threads zu den zentralen Ausführungseinheiten innerhalb eines Betriebssystems. Die MSD Library definiert einen Thread wie folgt:

„Ein Thread ist eine elementare Einheit, der vom Betriebssystem Prozessorzeit (Central Processing Unit [CPU] time) zugewiesen werden kann. Ein Thread kann jeden Teil einer Anwendung ausführen, insbesondere Teile, die zur gleichen Zeit von anderen Threads ausgeführt werden. Alle Threads eines Prozesses teilen sich den gleichen virtuellen Adressraum, die gleichen globalen Variablen sowie die gleichen Betriebssystemressourcen, die dem Prozess zugeteilt werden."

Ein Thread kann zum Beispiel eingesetzt werden, um Operationen im Hintergrund durchzuführen, ohne die Anwendung insgesamt zu blockieren. So läuft beispiels-

[28] **Kernel:** In der Informationsverarbeitung die grundlegenden, unverzichtbaren Funktionen des Betriebssystems (Operating System) mit den am häufigsten benötigten Programmteilen, auch als Core („Betriebssystemkern") bezeichnet.

[29] **RAS:** (Remote Access Service) Telekommmunikations-Dienst, der es einem entfernten Benutzer ermöglicht, sich über eine Telefonverbindung in ein Netzwerk einzuwählen und auf dessen Ressourcen und Dienste zuzugreifen.

[30] **Firmware:** Gesamtheit der zur Hardware gehörenden Steuerprogramme (Routinen), die einzelne Komponenten eines Prozesses direkt kontrollieren und bereits durch den Hersteller in den Festwertspeicher geschrieben werden.

weise eine Rechtschreibprüfung einer Textverarbeitung parallel zur Texteingabe.

In einer Kommunikationsanwendung können Threads dazu genutzt werden, gleichzeitig Befehle auszuführen. Ein Thread kann beispielsweise ankommende Client-Verbindungen annehmen, ein weiterer Thread kann eingesetzt werden, um Daten von den Clients zu empfangen und ein dritter Thread ist zur gleichen Zeit für das Versenden von Daten zuständig. Der Einsatz von Threads ist sinnvoll, wenn:

- Steuerungs- oder Monitoraufgaben implementiert werden müssen,
- die Anwendung eine stets wiederkehrende Aufgabe durchführen muss,
- eine zeitaufwendige Operation im Hintergrund durchgeführt werden soll oder
- der Prozess in der Lage sein muss, externe Ereignisse (Events) zu erkennen.

Der Einsatz von Threads kann sich allerdings auch nachteilig auswirken, weil sie alle mit dem gleichen virtuellen Adressraum eines Prozesses arbeiten. Falls bei der Ausführung eines Threads Fehler auftreten, kommt bisweilen der gesamte Prozess zum Erliegen. Darüber hinaus sind Konflikte möglich, weil sich mehrere Threads den gleichen Programmcode und die gleichen Daten teilen.

Daher erfordert die Programmierung mit Threads große Sorgfalt. Mit jedem neuen Thread steigt die Komplexität der Anwendung. Der Entwickler muss stets analysieren, welchen Einfluss jede einzelne Operation innerhalb eines Threads auf andere Threads haben könnte. Fehler treten zwar eher selten auf, haben allerdings eine große Wirkung auf die gesamte Applikation (z. B. unkontrollierter Zustand des Systems [Absturz], Datenverlust, Datenverfälschung) und sind außerdem schwer zu lokalisieren. Immer wenn eine Anwendung mit Hilfe mehrerer Prozesse oder Threads implementiert wird, besteht das Risiko von Zugriffskonflikten. Sobald sich Prozesse und Threads die gleichen Ressourcen teilen, muss es einen Mechanismus geben, der eine konfliktfreie Zuteilung oder Synchronisation gewährleistet (Synchronisations-Mechanismus). Zugriffskonflikte sind ebenfalls schwer zu lokalisieren. Häufig treten Schwierigkeiten erst dann auf, wenn die Software bereits fertig gestellt und der Debug-Modus[31] deaktiviert wird oder ein schnellerer Prozessor zum Einsatz kommt. Beide Maßnahmen führen zu einer beschleunigten Programmausführung und erhöhen somit die Wahrscheinlichkeit von Zugriffskonflikten. Vorsicht ist ebenfalls geboten, wenn eine Multi-Prozessor-Umgebung genutzt werden soll. Auch wenn eine Anwendung auf einer Ein-Prozessor-Maschine einwandfrei funktioniert, bedeutet dies nicht automatisch, dass sie auch auf einer Multi-Prozessor-Maschine stabil bleibt.

Endgerätekontrolle

Mobile Endgeräte werden in der Regel mit einem speziellen Endgerätetreiber für Windows Betriebssysteme ausgeliefert. Nach der Installation des Treibers sind die Endgeräte wie ein Standardmodem über verschiedene Hardware- und Software-Schnittstellen ansprechbar. Die meisten GPRS-Endgeräte werden über einen Communications-(COM-) Port angesteuert. Dies gilt auch für Endgeräte, die nicht direkt über die klassische Schnittstelle RS232[32], sondern z. B. über einen Universal Serial Bus (USB), Bluetooth oder einen Peripheral Component Interconnect-(PCI-)Bus angeschlossen werden. In diesen Fällen wird das Endgerät über einen virtuellen COM-Port angesprochen.

Die bekannteste Software-Schnittstelle ist die serielle Schnittstelle, die über ITU-T V.250[33] AT-Kommandos[34] angesprochen wird. Daneben steht als weitere Schnittstelle das bereits erwähnte RAS-API zur Verfügung, welches Steuer- und Monitorfunktionen auf einer höheren Abstraktionsebene ermöglicht. Darüber hinaus bieten einige Hersteller spezielle Bibliotheken und Software Developer Kits (SDKs) für bestimmte hardwarenahe Funktionen an, die vom Betriebssystem nicht unterstützt werden. Nachfolgend werden die serielle Schnittstelle und das RAS-API vorgestellt. (Eine genaue Anleitung, wie diese genutzt werden, um mobile Endgeräte anzusteuern, ist im ACDG der T-Mobile beschrieben.)

Verwendete Abkürzungen

AT	Advanced Technology
COM	Communications Port
CRC	Cycling Redundancy Check
IrDA	Infrared Data Association
ITU	International Telecommunications Union
LZH	Lempel, Ziv und Haruyasu
MPEG	Moving Picture Experts Group
PCI	Peripheral Component Interconnect
PIN	Personal Identification Number
PUK	Personal Unblocking Key
RAS	Remote Access Service
USB	Universal Serial Bus
WinSock	Windows Socket

Die serielle Schnittstelle erlaubt den direkten Zugriff auf das Mobilfunkgerät mit vielen verschiedenen Funktionen, wie z. B. Abfrage des Mobilfunknetzbetreibers, Eingabe der Personal Identifikation Number (PIN) oder des Personal Unblocking Key (PUK), Anmelden des Endgerätes im GPRS-Netz (Attach) oder Aufbau einer Datenverbindung. Mit Hilfe von AT-Kommandos werden in der Regel fast alle erdenklichen Funktionen ausgeführt, bis auf eine: paketorientierte Datenkommunikation über GPRS oder UMTS. Der Grund hierfür liegt darin, dass sich GPRS-Endgeräte nicht ganz so verhalten wie Modems, sondern eine Point-to-Point-(PPP-)Endstelle darstellen (Bild 18). Dies bedeutet, dass zwischen PC und GPRS-Endgerät eine PPP-Verbindung aufgebaut wird, was bei einer klassischen Verbindung zwischen zwei Modems nicht zwingend der Fall ist. Falls dennoch die Kom-

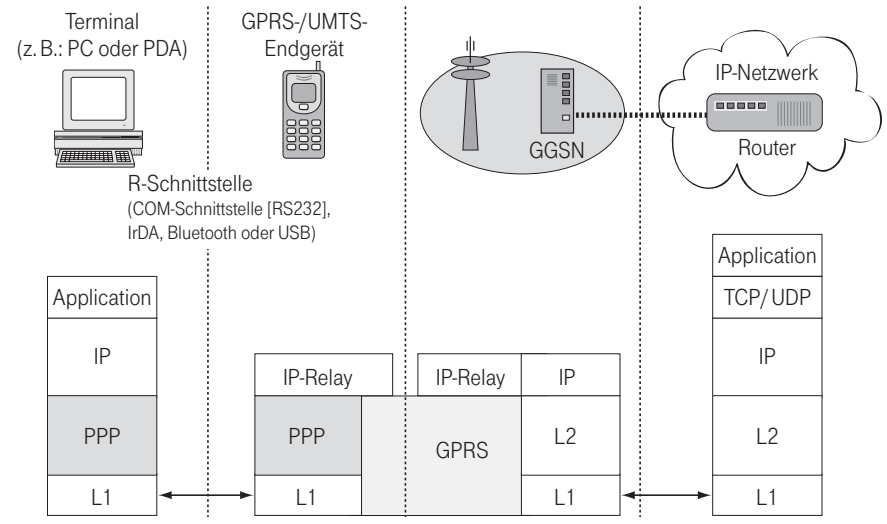
[31] Der **Debug-Modus** erlaubt es, Programme Schritt für Schritt durchzugehen, um Laufzeitfehler zu suchen und Programmabläufe zu testen. Parallel dazu kann der Quellcode Zeile für Zeile untersucht, Werte und Variablen abgefragt und das Programm angehalten werden.

[32] **RS232:** weit verbreitete Schnittstelle zur Datenübertragung mit geringer Datenrate (20 kbit/s) über kurze Entfernungen (15 m).

[33] **ITU-T V.250** ist eine Standardisierung der International Telecommunications Union. Die ITU untergliedert sich in drei Sektoren, wobei die Ergänzung –T für Telekommunikation steht. Die ITU-Empfehlungen werden mit Buchstaben gekennzeichnet, der den Anwendungsbereich beschreibt, gefolgt von einem Punkt und einer Zahl. Die V-Serie beinhaltet Regelungen zur Datenübertragung in Fernsprechnetzen über Datenschnittstellen und Modems.

[34] **AT-Kommandos** gehören zum Hayes-Befehlssatz, der eine Gruppe von Kommandosequenzen enthält, mit denen die Funktionsweise von Modems und ISDN-Adaptern gesteuert werden kann. AT steht als Abkürzung für „Attention" immer zu Beginn einer Kommandosequenz.

Bild 18 Ansteuerung von GPRS-/UMTS-Endgeräten über PPP



COM	Communications Port	GGSN	Gateway GPRS Support Node
IP	Internet Protocol	GPRS	General Packet Radio Service
IrDA	Infrared Data Association	UDP	User Datagram Protocol
PPP	Point-to-Point Protocol	PDA	Personal Digital Assistant
L	Layer	UMTS	Universal Mobile Telecommunications System
TCP	Transmission Control Protocol	USB	Universal Serial Bus

munikation wie im Bild 19 über AT-Befehle gesteuert werden soll[35], muss innerhalb der eigenen Applikation das vollständige PPP-Protokoll implementiert werden. Um diesen erhöhten Aufwand zu vermeiden, ist die Nutzung des RAS-API oder spezieller SDKs des jeweiligen Herstellers in Kombination mit dem WinSock-API zur Kommunikationssteuerung möglich. Der Einsatz der seriellen Schnittstelle ist dennoch sinnvoll, weil diese sehr interessante Funktionen, wie beispielsweise das Auslesen der Signalstärke, bietet.

Session Management

Im dritten Teil dieser Beitragsserie wurde auf die Bedeutung eines vollständigen Sitzungsmanagements für die Stabilität der Kommunikation hingewiesen. Sofern das Sitzungsmanagement während der Design-Phase mit Hilfe eines Zustandsautomaten oder einer Protokollmaschine vollständig spezifiziert wurde, ist die Implementierung sehr einfach. Bestehen jedoch Lücken in der Spezifikation, treten sie hier unmittelbar auf. Daher wird empfohlen, mit der Umsetzung erst dann zu beginnen, nachdem die Spezifikation im Einzelnen analysiert und für gut befunden wurde.

Ein Zustandsautomat oder eine Protokollmaschine beschreibt wie sich das System bei bestimmten Ereignissen in Abhängigkeit von seinem jeweiligen inneren Zustand verhält. Damit das Systemverhalten vorhersehbar ist, muss es einen definierten Anfangszustand geben. Eine Protokollmaschine lässt sich sowohl als Zustandsdiagramm (Bild 20), als auch in Form einer Zustandstabelle (Tabelle 4) darstellen. Die Visualisierung mit einem Zustandsdiagramms erleichtert es dem Leser, komplexe Zusammenhänge besser zu verstehen und eventuelle Unstimmigkeiten oder Lücken leicht zu erkennen. Eine Zustandstabelle wiederum eignet sich eher als Vorlage für den Entwickler, um die Zustandsmaschine in eine Programmierung umzusetzen. Durch die Übertragung eines Diagramms in eine Tabelle vollzieht der Entwickler bereits den ersten Schritt der Implementierung.

Um diese Tabelle zu erstellen, müssen sämtliche Zustände mit allen möglichen Ereignissen kombiniert werden. Für jede einzelne Zustands-Ereignis-Kombination ist zu beschreiben, welche Aktion auszuführen ist und in welchen Folgezustand das System durch das Auftreten des Ereignisses überführt wird. Die Zustandstabelle besteht aus vier Spalten. Jede Zustands-Ereignis-Kombination repräsentiert einen Durchlauf und belegt eine Zeile in der Tabelle.

Ein einfaches Beispiel für einen Zustandsautomaten ist das Starten und Stoppen eines Automotors. In diesem Modell gibt es die beiden Zustände „Motor an“ und „Motor aus“. Die externen Ereignisse sind „Zündung anschalten“ und „Zündung ausschalten“. Hieraus ergibt sich das einfache Zustandsdiagramm (s. Bild 20). Die entsprechende Zustandstabelle ist in Tabelle 4 enthalten.

Die Zahl der Reihen entspricht dem Produkt aus der Zahl der Zustände und Ereignisse. In manchen Fällen ist kein Zustandsübergang möglich, das heißt es wird keine Aktion durchgeführt und der ursprüngliche Zustand bleibt erhalten. Bei der Software-Entwicklung ist diese Situation oftmals kritisch. Die Ausführung einer Aktion kann in manchen Fällen sogar zu Schäden führen. Wird im beschriebenen Beispiel bei laufendem Motor die Zündung nochmals eingeschaltet, so kann dies zu einer Beschädigung des Anlassers führen.

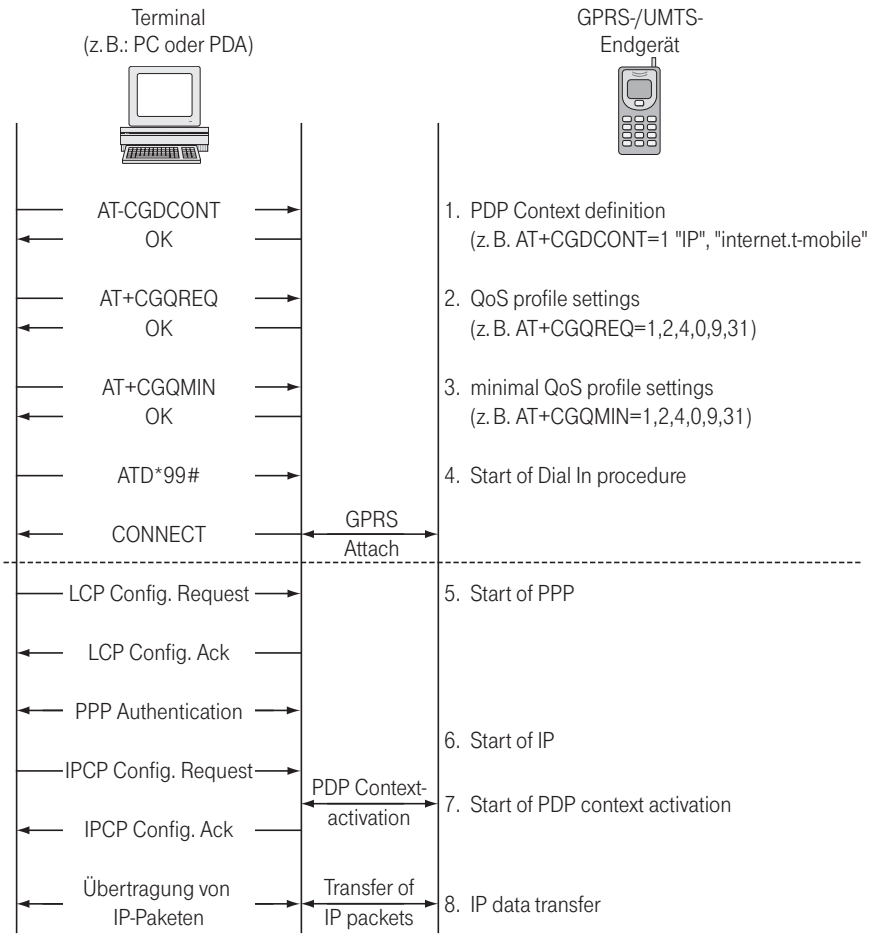
Optimiertes Kommunikationsprotokoll

Die Leistungsfähigkeit eines Kommunikationsprotokolls hängt im Wesentlichen vom Protokolldesign ab. Gerade für Protokolle, die in der mobilen Datenkommunikation zum Einsatz kommen, sollte der Kommunikationsablauf möglichst effizient und der Protokoll-Overhead möglichst gering sein.

Beim Entwurf eines Kommunikationsprotokolls stellt sich die Frage, wie Informationen über die Funkverbindung zu übertragen sind. Ein Protokolldatenblock kann mit einem Postpaket verglichen werden. Ein Paket besteht aus einem Inhalt und einer Adressangabe. In Fortführung dieser Analogie besteht ein Protokolldatenblock aus einem Header, der Informationen zur Identifikation des Pakets enthält und der angibt, wie mit dem Paket zu verfahren ist und einem Nutzdatenfeld.

[35] Details zur Ansteuerung eines GPRS-Endgerätes über AT-Befehle sind im ACDG der T-Mobile nachzulesen (www.t-mobile.de/entwickler).

Bild 19 Verbindungsaufbau mit AT-Befehlen



GPRS	General Packet Radio Service	PPP	Point-to-Point Protocol
PDP	Programmed Digital Processor	Ack	Acknowledgement
IP	Internet Protocol	AT	Attention
QoS	Quality of Service	LCP	Link Control Protocol
PDA	Personal Digital Assistant	IPCP	Internet Protocol Control Protocol

Bild 20 Beispiel eines Zustandsdiagramms

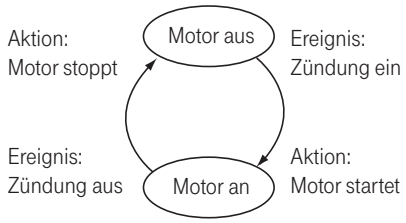


Bild 21 Typischer Aufbau eines Datenpakets

Abschließend enthält ein Datenpaket in der Regel eine **Prüfsumme** (Checksum), anhand derer der Empfänger die Unversehrtheit (Integrität) des Pakets überprüfen kann. Der allgemeine Aufbau eines Datenpakets ist in Bild 21 dargestellt.

Header

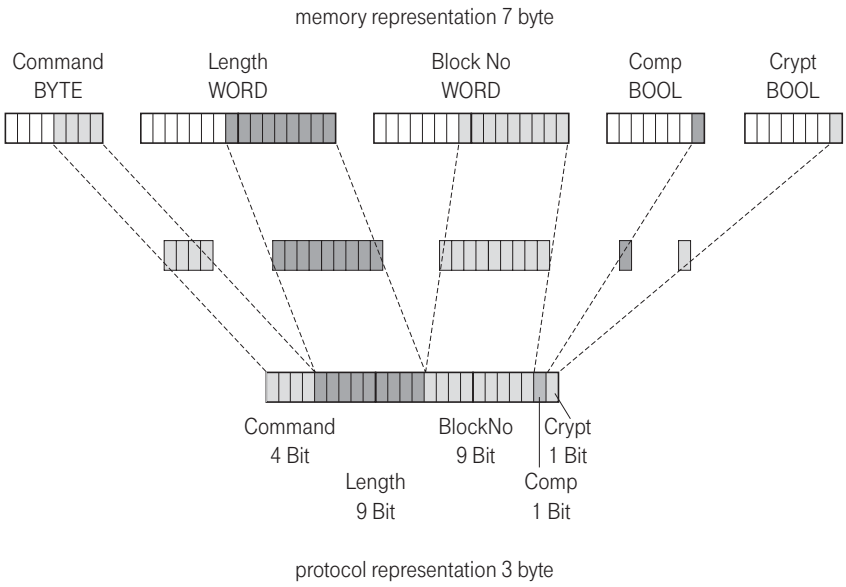
Der Header ist bei einer Datenübertragung der wichtigste Teil des Protokollblocks. Auf der Grundlage des Headers erkennt der Empfänger den Pakettyp, die Paketlänge und den Absender. Typische Bestandteile des Headers sind:

- Kommandotyp
  Der Kommandotyp gibt an, wie der Empfänger den Protokollblock interpretieren muss. Typisch hierfür sind Befehle wie LOGIN_REQUEST oder LOGIN_RESPONSE (s. Bild 16).
- Blocklänge
  Abhängig vom genutzten Basisprotokoll (UDP oder TCP) werden die Daten kontinuierlich in einem Datenstrom (Stream) versendet. Häufig werden dabei mehrere Kommandos kurz hintereinander übertragen, manchmal nur Fragmente eines einzigen Kommandos. Ankommende Daten werden auf der Empfängerseite zunächst in einem Puffer (Ring-Buffer) zwischengespeichert, um anschließend die vollständigen Kommandos korrekt zusammensetzen zu können. Damit dies gelingt, wird die Header-Information über die Blocklänge benötigt. Die Blocklänge wird in der Regel über alle Paketteile, also einschließlich Header und Prüfsumme, angegeben.
- Sequenznummer
  Sofern für die Datenübertragung ein Basisprotokoll ohne Datensicherung wie UDP genutzt wird, kann es zu Mehrfachübertragungen oder zu einer veränderten Reihenfolge kommen. Anhand der Sequenznummer erkennt die Applikation auf der Empfängerseite doppelte Datenpakete und ist somit in der Lage, die ur-

Tabelle 4 Zustandstabelle

Zustand	Ereignis	Aktion	Neuer Zustand
Motor aus	Zündung ein	Motor startet	Motor an
Motor aus	Zündung aus	Keine Aktion	Motor aus
Motor an	Zündung ein	Keine Aktion	Motor an
Motor an	Zündung aus	Motor stoppt	Motor aus

Bild 22 **Beispielhafte Transformation des Headers von einer Codierung mit Standarddatentypen zu einer bitweisen Codierung**



sprüngliche Reihenfolge wieder herzustellen.

- Zusätzliche Flags
  Flags[36] werden genutzt, um zusätzliche Informationen zu übertragen. Beispiele hierfür sind ein Komprimierungs- oder ein Verschlüsselungs-Flag. Diese Informationen zeigen dem Empfänger, wie die empfangenen Datenpakete zu behandeln sind.

Nutzdatenfeld

Das Nutzdatenfeld enthält die Anwendungsdaten des Protokollkommandos. Nicht alle Kommandos benötigen zusätzliche Daten, so dass in manchen Fällen das Nutzdatenfeld leer bleibt, wie beispielsweise beim Logoff-Kommando.

Prüfsumme

Die Prüfsumme wird über den Header und das Nutzdatenfeld berechnet. Sie bildet den letzten Teil eines Protokolldatenblocks. Anhand der Prüfsumme kann der Empfänger die Integrität des Protokolldatenblocks überprüfen. Sollte ein Teil des Datenpakets bei der Übertragung verfälscht worden sein, so ergibt sich bei der Nachberechnung der Prüfsumme eine Differenz. Ein häufig genutzter Algorithmus zur Fehlererkennung, der beispielsweise von den Protokollen IP, UDP oder TCP genutzt wird, ist der Cyclic Redundancy Check (CRC). Bei dieser Methode wird üblicherweise ein Kontrollbitmuster gebildet, dass eine Länge von 16 Bit oder 32 Bit besitzt. Sofern das genutzte Kommunikationsprotokoll UDP oder TCP als Transportprotokoll einsetzt wird, ist die Berechnung einer weiteren Prüfsumme nicht notwendig.

Datenreduktion

Eine Reduzierung der zu übertragenen Datenmenge hat gleich mehrere Vorteile:

- Die Anwendung läuft schneller.
- Die Kosten für die Datenübertragung werden reduziert.
- Der Bedarf an Bandbreite für die Anschaltung des Kundennetzes wird kleiner.

Der beste Weg, die Datenmenge zu reduzieren, besteht darin, unnötige Daten nicht zu senden. In Teil 3 wurde beschrieben, wie sich durch Zusammenfassen mehrerer Transaktionsschritte die übertragene Datenmenge verringern und der Programmablauf beschleunigen lässt. In diesem Teil wird nun erläutert, wie die Datenmenge mit Hilfe von allgemein verfügbaren Komprimierungsverfahren und Umcodierung weiter reduziert werden kann.

Hier besteht zunächst die Möglichkeit, den Header des Kommunikationsprotokolls dadurch zu verkleinern, dass keine Standarddatentypen wie BYTE, WORD oder BOOL verwendet werden, sondern die Information vor der Übertragung bitweise codiert wird (Bild 22). Die Nutzung der Standarddatentypen eignet sich gut für die Bearbeitung der Header-Information durch die Programmlogik, wohingegen eine bitweise Codierung zu einer deutlichen Einsparung bei der Datenmenge führt.

Darüber hinaus lassen sich die Nutzdaten mit allgemein bekannten Verfahren in einem speziellen Format komprimieren. Auf der Empfängerseite werden anschließend die Daten zunächst dekomprimiert, bevor sie weiter verarbeitet werden können.

Es gibt verschiedene allgemein verfügbare Komprimierungsverfahren. Hierbei wird grundsätzlich unterschieden in:

- verlustbehaftete und
- verlustfreie Verfahren.

Bei verlustbehafteten Verfahren wird durch die Komprimierung gleichzeitig die Struktur und die Qualität der Daten verändert, so dass das Originalformat auf der Empfängerseite nicht mehr hergestellt werden kann. Ein Beispiel hierfür sind MPEG-Verfahren[37], die für Bilder oder Videos eingesetzt werden, wobei die Bildauflösung verringert wird.

Beim Einsatz eines verlustfreien Verfahrens hingegen können die ursprünglichen Daten vom Empfänger wieder vollständig rekonstruiert werden. Bekannte verlustfreie Kompri-

[36] Ein **Flag** ist eine zweiwertige Variable innerhalb eines Registers in einem Prozessor, die zur Kennzeichnung eines bestimmten Zustandes verwendet wird. In der Regel handelt es sich um ein einzelnes Bit. Häufig zeigen Flags in einem Übertragungsprotokoll den Beginn und das Ende eines Datenpakets an.

[37] Siehe hierzu den Beitrag „Die Standards MPEG-4 und MPEG-7 in den Multimedia-Diensten", Unterrichtsblätter Nr.7/2000, S. 326 ff.

mierungsalgorithmen sind beispielsweise das Huffmann-Verfahren oder der LZH-Algorithmus. Hierzu ist eine plattformunabhängige Bibliothek verfügbar, die in jede Applikation integriert werden kann. Diese Bibliothek trägt die Bezeichnung ZLIB und ist unter www.zlib.org abrufbar.

Fehlerbehandlung

Fehlererkennung und Fehlerbehandlung spielen bei der Entwicklung mobiler Applikationen eine sehr wichtige Rolle. Die Anwendung muss in der Lage sein, einen Kommunikations- oder Endgerätefehler zu erkennen und zu behandeln, bevor der Anwender ihn bemerkt. Sie müssen daher im Hintergrund vorgenommen werden, ohne die Nutzung wesentlich zu beeinträchtigen.

Im Folgenden wird der Umgang mit Kommunikations- und Endgerätefehlern erläutert.

Kommunikationsfehler – WinSock Error Handling

Das WinSock-API bietet verschiedene Funktionen zur TCP-Programmierung. Jede Funktion kann zu einem Fehlerzustand führen. Daher muss der Entwickler bei jedem Funktionsaufruf das Ausführungsergebnis prüfen. Im Falle eines Fehlers speichert die WinSock-Datei den entsprechenden Fehlercode in einer so genannten globalen Variablen. Durch den Aufruf der Funktion WSAGetLast Error() kann auf diesen Fehlercode zugegriffen werden. Dabei ist es wichtig, dass dieser Aufruf unmittelbar im Anschluss an den fehlerhaften Funktionsaufruf getätigt wird, weil der Fehlercode anderenfalls von einer anderen Funktion überschrieben wird. Die WinSock-Datei verwaltet für jeden einzelnen Thread eine eigene globale Fehlervariable, so dass ein Funktionsaufruf mit einem anderen Thread den Fehlerzustand nicht überschreiben kann. Es gibt drei typische Phasen, in denen Kommunikationsfehler behandelt werden müssen:

- beim Verbindungsaufbau zwischen Client und Server,
- beim Versenden und Empfangen von Daten und
- während einer aktiven Verbindung ohne augenblicklichen Datenverkehr (Leerlaufmodus).

Jede Phase wird nachfolgend separat beschrieben. Dabei wird gezeigt, welche WinSock-Fehler auftreten können und wie diese in der jeweiligen Phase zu interpretieren sind.

Verbindungsaufbau

Hierzu wird die Funktion Connect() aufgerufen. Ein Fehler in dieser Funktion mündet in einen der folgenden Fehlerzustände:

- WSAETIMEDOUT (Error code: 10060): Der Server antwortet nicht innerhalb der vorgegebenen Zeitspanne.
- WSAECONNREFUSED (Error code: 10061): Der Server lehnt ausdrücklich den Verbindungswunsch ab.
- WSAEHOSTDOWN (Error code: 10064): Der Server ist über das genutzte Netz

Tabelle 5 Möglichkeiten der Fehlerbehandlung

Fehler	Interpretation und Empfehlung
WSAETIMEDOUT	Routing-Fehler Das Connect-Kommando bzw. die Antwort ging bei der Übertragung zum Server oder zum Client verloren. Maßnahme: Grundsätzliche Verfügbarkeit der IP-Verbindung mit Hilfe von PING oder TRACERT von beiden Seiten überprüfen. Zu große Netzlaufzeit Die Netzlaufzeit ist zu groß, so dass die Operation in eine Zeitbegrenzung (Time out) hineinlief. Maßnahme: Erneuter Aufbau der Socket[38]-Verbindung. Falls dies nicht hilft, die GPRS-Verbindung erneut aufbauen. Ebenfalls möglich: Reset des Endgeräts und Neuanmeldung beim Server durchführen.
WSAECONNREFUSED	Anwendungsserver nicht verfügbar Der Anwendungsserver wurde gestoppt. Maßnahme: Betrieb eines Backup-Servers. Überlast des Anwendungsservers Die Ressourcen des Servers sind ausgelastet. Maßnahme: Erneuter Versuch nach einigen Sekunden, danach weitere Versuche jeweils mit exponentiell ansteigender Wartezeit.
WSAEHOSTDOWN	Hardware-Fehler Der Server steht nicht zur Verfügung. Maßnahme: Betrieb eines Backup-Servers. Routing-Fehler Es besteht keine Verbindung zum Server. Maßnahme: Grundsätzliche Verfügbarkeit der IP-Verbindung mit Hilfe von PING oder TRACERT von beiden Seiten überprüfen.
WSAEHOSTUNREACH	Routing-Fehler Die IP-Route zum Server steht nicht zur Verfügung. Maßnahme: Überprüfung der Routing- Konfiguration auf Client- und Server-Seite. Grundsätzliche Verfügbarkeit der IP-Verbindung mit Hilfe von PING oder TRACERT von beiden Seiten überprüfen.
WSAECONNRESET	Keine Verbindung Die Verbindung wurde vom Server auf Grund von Überlast oder Systemfehlern beendet. Eine weitere Ursache kann eine Störung der GPRS-Verbindung sein. Maßnahme: Erneuter Verbindungsaufbau auf IP-Ebene. Falls dies nicht ausreicht, zusätzlich Neuaufbau der GPRS-Verbindung.

nicht erreichbar. Die Server-Maschine ist ausgeschaltet.
- WSAEHOSTUNREACH (Error code: 10065): Es gibt keine IP-Verbindung zum angegebenen Server.

Die Tabelle 5 gibt Empfehlungen zur Fehlerbehandlung.

Versenden und Empfangen

Tritt in der Phase der Datenübertragung ein Fehler auf, kann die Ursache in der Unterbrechung der Verbindung liegen. Folgende Fehlermeldungen sind möglich:

- WSAETIMEDOUT (Error code: 10060)
- WSAEHOSTUNREACH (Error code: 10065)
- WSAECONNRESET (Error code: 10054)

Fehler im Leerlaufmodus (Idle state)

Der Fehlercode eines Funktionsaufrufs des WinSock-APIs zeigt einen Fehler, der eindeutig einer bestimmten Aktion zugeordnet werden kann. Ein Client, der regelmäßig Daten vom Server empfängt, sendet nur dann einen Funktionsaufruf, wenn ihm der Empfang von Daten durch das Betriebssystem signalisiert wird. Bleibt dies aus, d. h. befindet sich das System im Leerlaufmodus, so wird keine WinSock-Funktion aufgerufen und somit auch kein Fehler erkannt.

Um einen Fehlerzustand zu erkennen, muss ein so genannter Keep-Alive-Mechanismus implementiert werden, bei dem beide Seiten regelmäßig Testmeldungen schicken und anhand ihres Ausbleibens auf der Empfangsseite einen Fehlerzustand ableiten. Obwohl auch das WinSock-API selbst einen solchen Mechanismus anbietet, wird empfohlen, innerhalb der Kommunikationskomponente einen eigenen Mechanismus zu implementieren. Sofern ein geeignetes Zeitintervall für den Versand gewählt wird, hat dies keinen entscheidenden Einfluss auf die Kommunikationskosten und bringt eindeutige Vorteile bei der Fehlererkennung.

Fehlercodes des RAS-API und deren Bedeutung

Genauso wie das WinSock-API liefert auch das RAS-API umfangreiche Fehlercodes im Falle einer Fehlfunktion der Endgeräte. Fehler treten auf dieser Ebene überwiegend während des Verbindungsaufbaus und weniger während einer laufenden Verbindung auf. Die Funktion RASGetConnectStatus liefert den aktuellen Verbindungsstatus und gegebenenfalls den entsprechenden Fehlercode. Es gibt zahlreiche mögliche RAS-Fehlerzustände (siehe www.microsoft.com). Im ACDG werden diejenigen detalliert ausgeführt, die typisch für GPRS-Verbindungen sind. *(Br)*

Der Beitrag wird fortgesetzt.