Großer Beleg

Entwicklung eines CANopen-Netzwerkes

Jan Blumenthal

17.09.2001

Universität Rostock, Fachbereich für Elektrotechnik und Informationstechnik
Institut für Automatisierungstechnik

Vortragsgliederung

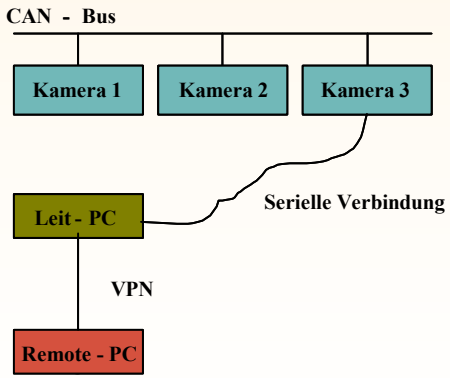
- Grundlagen von CANopen
- Zielsetzungen bei der Entwicklung
- Softwareüberblick
  - CANopen-API
  - CANopenConsole
- Zusammenfassung

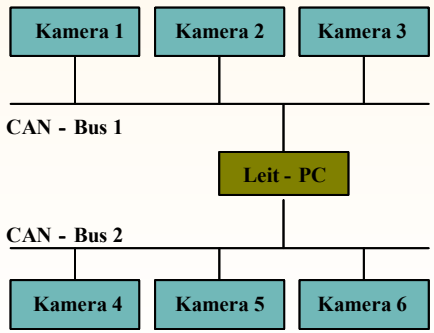
CANopen

- Protokoll für den CAN-Bus
- Implementierung verschiedener Netzwerkdienste:
  - Interne Datenbank (Object Dictionary)
  - Segmentierung von Datenströmen (SDO)
  - Echtzeitfähigkeit (PDO)
  - Netzwerkmanagement (NMT)
  - Fehlermanagement (NMT-Error Control)
  - Netzwerküberwachung (Heartbeat, Life guarding)
  - Synchronisation von Teilnehmern (SYNC)

Ziele der CANopen-API

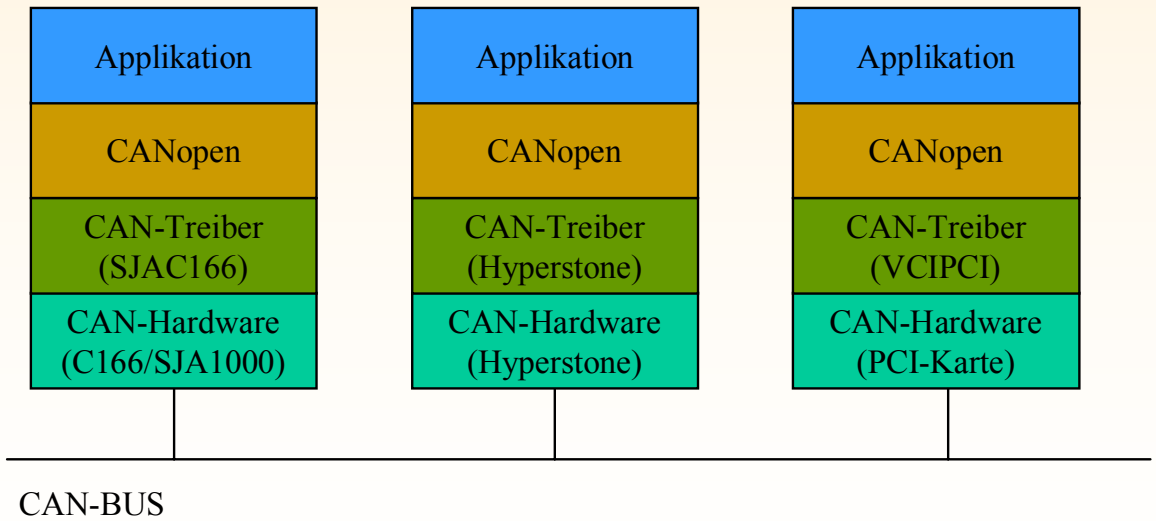
- Ziele:
  - Funktionsfähigkeit und Standardkonformität
  - Portabilität (PC, C166, Windows, Linux)
  - Hohe Performance und geringer Speicherverbrauch
  - Unterstützung mehrerer Instanzen
  - Fernwartung
  - Entlastung der Prozessanwendung
- Einsatzszenarien:



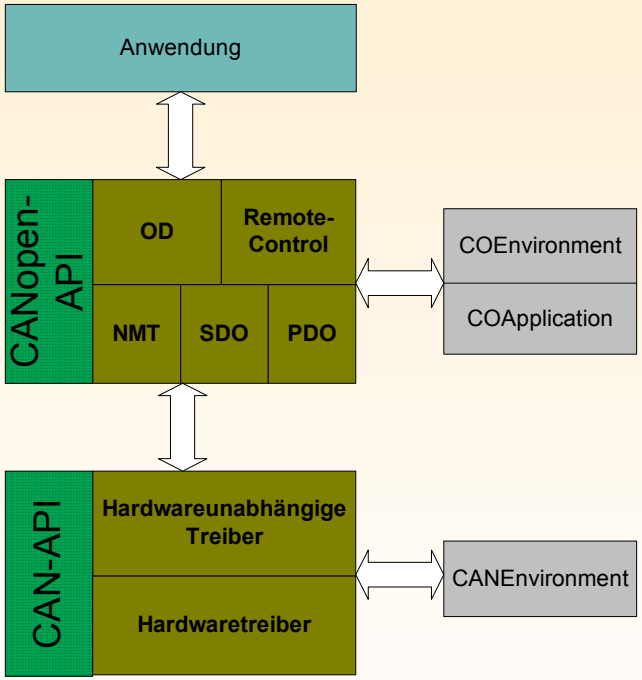


Schichtenmodell der CANopen-API

Vorteile:
- Standardisierte Schnittstellen
- Vollständige Kapselung
- Nutzung der CAN-API



Komponenten der CANopen-Software



← Prozessumgebung

← Verwaltung der CANopen Instanz
← Einbindung in das Betriebssystem

← Definition der CANopen-Anwendung

← Einbindung der Hardwaretreiber

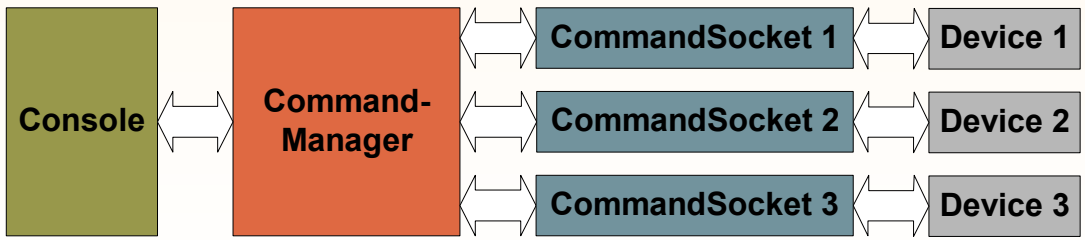
Besonderheiten der CANopen-API

- Benutzen von Handles
- Keine dynamische Speicherallozierung
- Nachrichtenverarbeitung:
  - Interrupt
  - Polling Loops
  - Ereignisgesteuerte Abarbeitung
- Callbacks
- Remote Objects
  - FPGA

CANopen Console

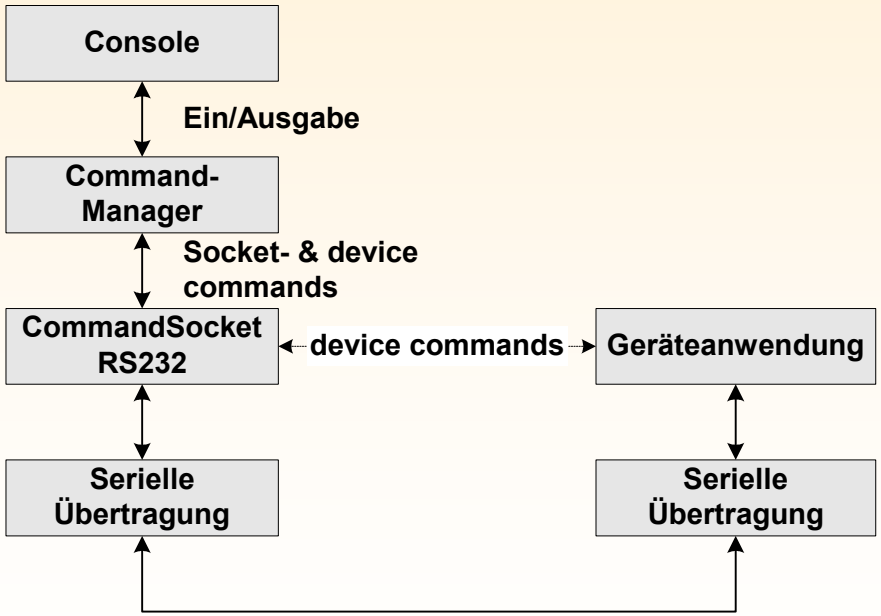
- Administrierungssoftware
- Eigenschaften:
  - Konsolenprogramm
  - Unterstützung mehrerer Instanzen
  - Skripting
  - Fernwartung
  - Kommunikation über Sockets
  - Objekt-orientierter Entwurf
  - Multithreading

- Aufbau:



Remote Control

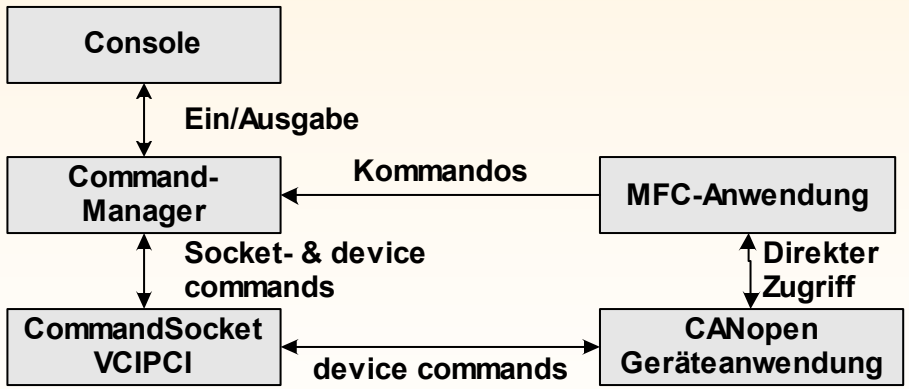
- Optionales Softwaremodul in der CANopen-API
- Übertragung von Kommandos an Geräteanwendungen
  Beispiel: *reset 1*



Kopplung der Software

Unterstützung verschiedener Administrierungsvarianten

- Graphisch (MFC)
- Textbasiert (Console)



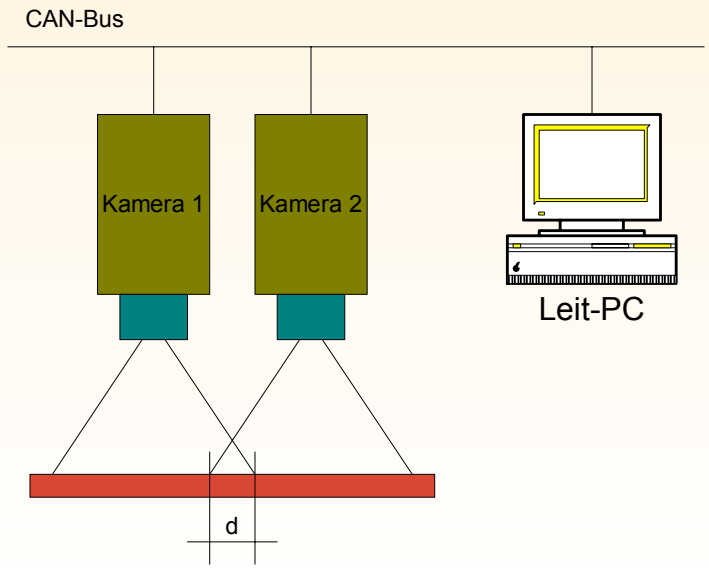
Zusammenfassung

	Stand	Zukunft
CANopen-API	- Portabel - Weitestgehend standardkonform - Fernwartbar - Multisessionfähig	- Echtzeittest - Konformitätstest
CANopen Console	- Windows Version - Telnet-fähig	- Portabilität der EClasses

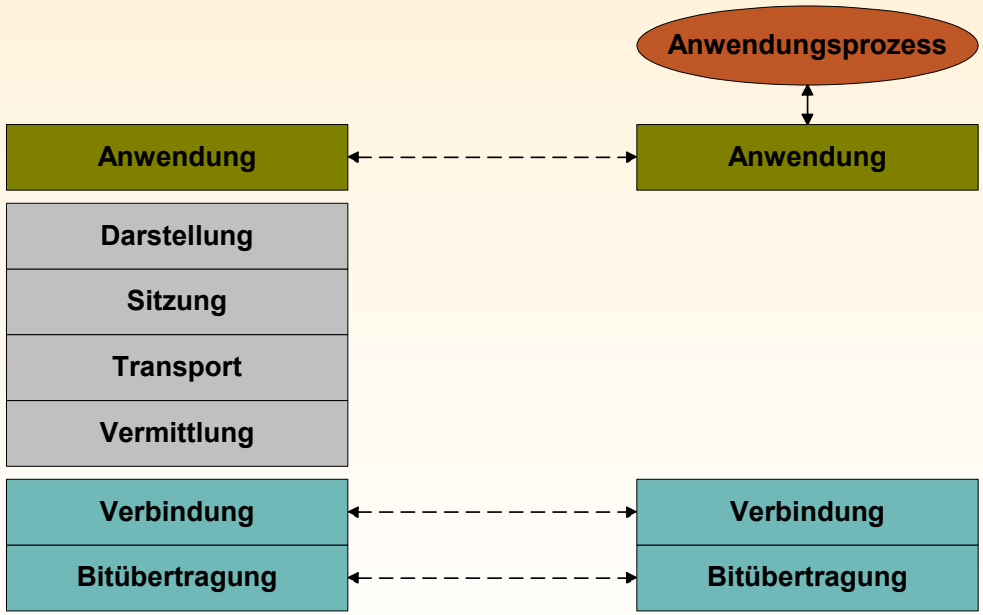
Vielen Dank

Beispielanwendung

- Steuerung eines Netzwerkes durch einen Leit-PC
- Kritischer Bereich ‚d‘



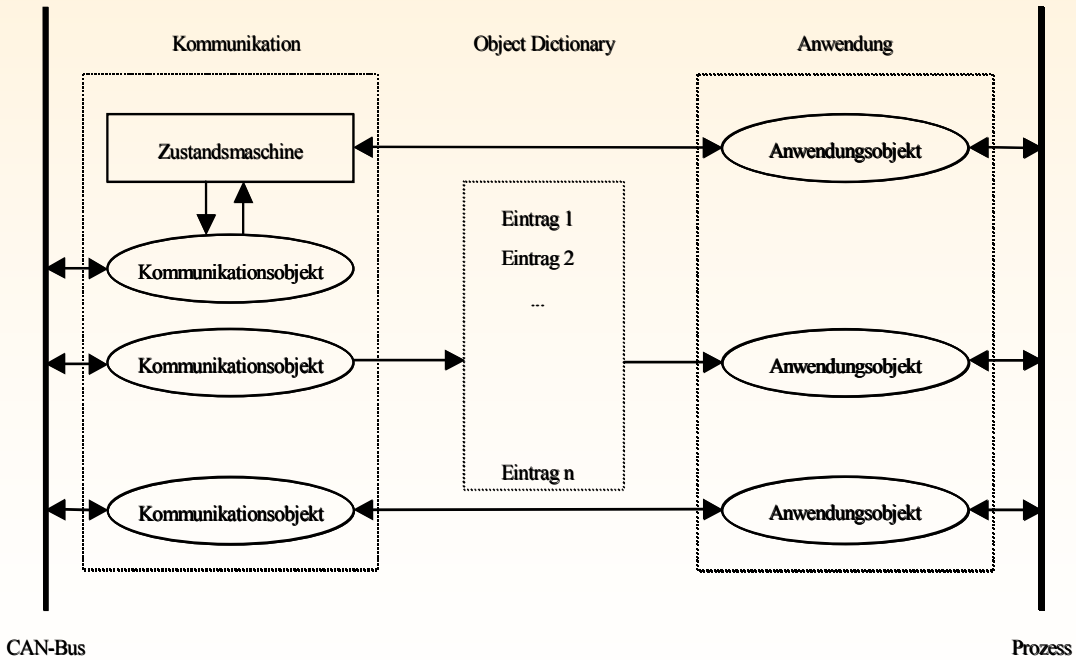
CANopen Referenz-Modell



Gerätemodell von CANopen

Object Dictionary:

- Interne Datenbank
- Unterschiedliche Datentypen (int, char)

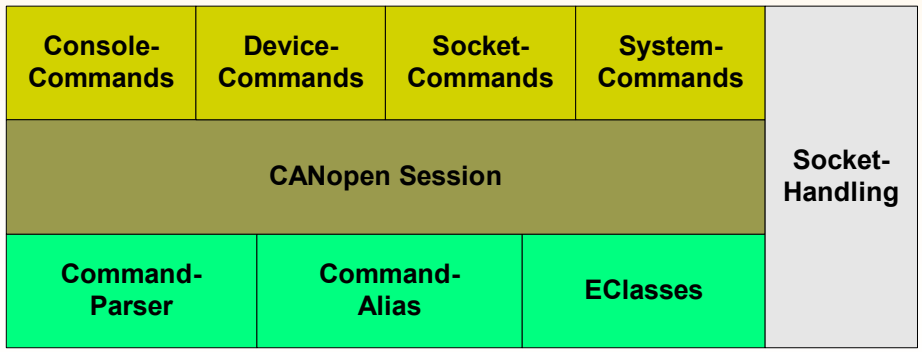


Typische Main-Funktion

```
#include <stdio.h>
#include <CANopen/CANopen.h>

OPTOERROR APP_ConfigureCANopen(CO_HANDLE *cohandle);     // external, defined in COEnvironment.c

void main()
{
        CO_HANDLE       handle;                          // CANopen Handle
        char            buffer[CONSOLEBUFFERSIZE];       // buffer for remote commands

        if ( ! (CO_InitCANopen(&handle, APP_ConfigureCANopen)) )        // init CANopen
        {
                TRACE1("Camera: %d ready!\n",(int)(handle.NodeID));
                while (1)                                // endless loop
                {
                        CO_MessageLoop(&handle);         // call message loop
                        if ( GetRemoteCommand(buffer) ) // new remote command received ?
                                CO_RemoteCommand(&handle, buffer);
                }
        }
        CO_EndCANopen(&handle);
}
```

Komponenten der CANopenConsole

- Kommandotypen:
  - Konsolenkommandos (help, quit)
  - Socketkommandos (socket, init)
  - Gerätekommandos (od, nmt)
  - Systemkommandos (dir, ls)
- Sessionhandling
- Socketverwaltung
- Kommandoparser
- Betriebssystemanpassung (EClasses)



EClasses

- Klassenbibliothek
- Sammlung einfacher Basisklassen
  - EThread
  - EEvent
  - EFile
  - EString
- Plattformunabhängig