Steuergerätetests effizienter programmieren – Basics, Tipps und Tricks beim Einsatz von CAPL

Teil 1: CAPL-Basics

CAPL ist eine von Vector Informatik entwickelte Programmiersprache, die in den weit verbreiteten Software-Werkzeugen CANoe und CANalyzer zur Verfügung steht. In drei aufeinanderfolgenden Beiträgen werden zu CAPL-Grundlagen sowie Tipps und Tricks je nach Kenntnisstand der Anwender vorgestellt. Dieser erste Teil konzentriert sich dabei auf die Basics von CAPL. Er ist in erster Linie für Neulinge dieser Sprache interessant; für Kenner ergeben sich eventuell ein paar Einblicke in die Motivation für einzelne CAPL-Konstrukte. Der zweite Teil wird fortgeschrittene Funktionsweisen von CAPL behandeln. Der dritte Teil schließlich enthält eine Betrachtung zur Performance und zum Speicherbedarf sowie Tipps und Tricks zur Verwendung von Datenbanken und assoziativen Arrays.

Seit über 20 Jahren – damals zunächst in CANalyzer für DOS – lässt sich mit CAPL eine große Bandbreite von Aufgaben effizient umsetzen: von einfachen Stimuli bis zur Simulation komplexer Busteilnehmer. Im Folgenden wird CANoe stellvertretend für die beiden Produkte CANoe und CANalyzer genannt. Das Ziel von CAPL ist seit jeher, die jeweiligen Aufgabenstellungen so einfach wie möglich zu lösen. Typische Aufgaben sind das Reagieren auf empfangene Botschaften, das Testen und Setzen von Signalwerten oder das Senden von Botschaften. Hier sollte sich ein Programm auf genau diese Dinge beschränken und keinen weiteren Overhead erfordern.

Viele Programmieraufgaben, mit denen ein CANoe Anwender zu tun hat, mögen tatsächlich so kurz und trivial wie das unten aufgeführte Beispiel sein – viele andere Aufgaben sind es natürlich nicht. Deshalb wurde CAPL stetig über die Jahre erweitert, um auch bei komplexen Aufgaben als Werkzeug nach dem Grundsatz „so einfach wie möglich“ zur Verfügung zu stehen.

„CAPL“ steht dabei als Akronym für „Communication Access Programming Language“. Die ursprüngliche Ausrichtung auf CAN ist schon längst auf alle automobilen und einige weitere Bussyteme erweitert, wie beispielsweise LIN, FlexRay, MOST, J1587 aber auch ARINC und CANopen.

Die Syntax von CAPL lehnt sich – wie viele andere Sprachen auch – eng an die Syntax der Sprache C an. Wer sich mit C, C# oder verschiedenen modernen Script-Sprachen auskennt, findet sich auch in CAPL schnell zurecht. Ein paar Auffälligkeiten unterscheiden aber ein CAPL-Programm von einem C-Programm:

> CAPL-Programme sind event-orientiert. Das heißt, sie bestehen aus einzelnen Funktionen, die jeweils auf ein Event innerhalb des aktuell betrachteten Systems reagieren: den Empfang einer Botschaft, die Änderung eines Signals, das Ablaufen eines Timers oder auch eine Änderung in der „Umgebung“. So wird auf die Botschaft „EngineState“ reagiert: „On message EngineState“ **(Bild 1)**.

> CAPL-Programme verwenden spezifische Datenbanken für die Kon-zepte des aktuell betrachteten Systems. Botschaften und Signale erhalten dort Namen und können direkt mit diesem im Programmcode verwendet werden. In Bild 1 sind das die Bezeichner „EngineState“ für eine Botschaft und „EngineSpeed“ für ein Signal auf dieser Botschaft.

> CAPL-Programme geben dem Benutzer keine Pointertypen an die Hand. Damit sind eine Unmenge Programmierfehler und Ursachen für Programmabstürze, wie sie C-Programmierern häufig widerfahren, von vornherein ausgeschlossen. Da Pointer abgesehen von ihrer Fehleranfälligkeit auch ein sehr mächtiges Konzept darstellen, gibt es für manche Dinge in CAPL Ersatz, so zum Beispiel die assoziativen Arrays als Ersatz für dynamischen Speicher.

Eine wichtige Eigenschaft, die CAPL mit C verbindet, sei noch erwähnt: CAPL wird immer kompiliert, also in effizient ausführbaren, flexiblen Maschinencode übersetzt.

Beispiel: Ein simples CAPL-Programm

In diesem Abschnitt wird ein einfaches CAPL-Programm vorgestellt **(Bild 1)**, das eine der grundlegenden Aufgaben eines Busmonitorwerkzeugs erledigt: es lauscht am Bus und bereitet ein paar der Ereignisse auf dem Bus für die Beobachtung/Überwachung durch den Anwender auf. Es handelt sich hier um ein gekürztes Beispielprogramm von CANoe: Display.can aus dem Beispiel Easy.cfg. Das **Bild 1** zeigt das Programm. Im Folgenden wird erst die Gesamtfunktion kurz umrissen, bevor dann die einzelnen Abschnitte genauer beschrieben werden.

Aufgabenstellung

> Es soll ein CAN-Bus beobachtet werden, dessen Elemente wie Busknoten, Botschaften und transportierte Signale mit einer Datenbasis beschrieben werden.

> Wenn die Botschaft EngineState empfangen wird, dann soll das darauf enthaltene Signal EngineSpeed für die Darstellung in einem Anzeigepanel aufbereitet werden und an das Panel weitergereicht werden.

> Wenn die Botschaft LightState empfangen wird, dann sollen die darauf enthaltenen Signale HeadLight und Flashlight für die grafische Darstellung in einem Anzeigepanel aufbereitet werden und an das Panel weitergereicht werden.

Genaue Betrachtung des Programms

Die Zeilennummern sind nicht Bestandteil des CAPL-Programmes und sind hier nur eingefügt, um einfacher einzelne Zeilen oder Abschnitte referenzieren zu können. Um eine möglichst kompakte Darstellung zu ermöglichen, wurden hier öffnende Klammern nicht in einer separaten Zeile platziert.

In einem CAPL–Programm können globale Variablen und Konstanten definiert werden. Dies geschieht in dem Abschnitt „variables" (Zeilen 1..5). Global sind diese Konstanten und Variablen für dieses Programm: sie sind überall im Programm verwendbar, aber nicht in anderen Programmen innerhalb derselben Anwendung von CANoe. Die weiteren Abschnitte definieren Reaktionen auf Ereignisse (Zeilen 7..17) und eine Hilfsfunktion (Zeilen 19..28).

Die Zeilen 7..9 zeigen eine minimale Form einer Botschafts-Event-Prozedur. Diese Funktion wird genau dann aufgerufen, wenn diese Botschaft auf dem Bus übertragen wurde. Spezifisch für CAN bedeutet das: Der genaue Zeitpunkt ist der TX bzw. RX–Interrupt des CAN-Controllers, also direkt nach korrekt erfolgter Übertragung der Botschaft. Die Botschaft aufgrund derer die aktuelle Funktion aufgerufen wird, wird mit der Syntax „this" bezeichnet.

In Zeile 8 wird der Wert des Signals EngineSpeed aus der soeben empfangenen Botschaft (this) ausgelesen und mit einer Umrechnung (/ 1000.0) an eine Systemvariable zugewiesen.

Die Zeilen 11..17 zeigen eine Botschaftseventprozedur für die Botschaft LightState, die die Informationen zu einem Blinker überträgt. Die Verarbeitung ist ähnlich wie bei der Botschaft EngineState mit folgenden Besonderheiten: In Zeile 12 wird nun in der gerade übertragenen Botschaft (this) das RichtungsFlag (.dir) geprüft. Es sollen in diesem Programm nur empfangene Botschaften betrachtet werden (Wert RX), da auch eine vom Knoten selbst versendete Botschaft eine Event-Prozedur auslöst (Wert TX). In diesem Fall würde in Zeile 15 gegebenenfalls eine Fehlermeldung ausgegeben.

Da die Aufarbeitung des Signals für die Darstellung an der Oberfläche (einem Panel, in dem verschiedene Zustände mit verschiedenen Bitmaps dargestellt werden) etwas aufwändiger ist, wird die Implementierung in eine eigene Funktion ausgelagert: In Zeile 13 wird SetLightDsp mit den beiden benötigten Signalen der Botschaft als Parameter aufgerufen.

```
1.   variables
2.   {
3.     const long kOFF = 0;
4.     const long kON = 1;
5.   }
6.
7.   on message EngineState {
8.     @sysvar::Engine::EngineSpeedDspMeter = this.EngineSpeed / 1000.0;
9.   }
10.
11.  on message LightState {
12.    if (this.dir == RX) {
13.      SetLightDsp(this.HeadLight,this.FlashLight);
14.    } else {
15.      write("Error: LightState TX received by node %NODE_NAME%");
16.    }
17.  }
18.
19.  SetLightDsp (long headLight, long hazardFlasher) {
20.    long tmpLightDsp;
21.
22.    tmpLightDsp = 0;
23.    if(headLight == kON)
24.      tmpLightDsp = 4;
25.    if(hazardFlasher == kON)
26.      tmpLightDsp += 3;
27.    @sysvar::Lights::LightDisplay = tmpLightDsp;
28.  }
```

Bild 1: Ein einfaches CAPL-Programmbeispiel

Die Zeilen 19 bis 28 schließlich definieren eine eigene Funktion, die je nach Wert der übergebenen Signale unterschiedliche Werte in die Systemvariable LightDisplay im Namensraum Lights schreibt. Diese Variable wählt dann in dieser Demokonfiguration in einem Anzeigepanel die jeweils passende Bitmap aus.

Übersetzung der englischen Veröffentlichung im CAN Newsletter, Ausgabe 2/2014.

CAPL – „Communication Access Programming Language"
CAPL ist eine von Vector Informatik entwickelte prozedurale, C-ähnliche Programmiersprache. Die Ausführung wird von Progammblöcken durch Ereignisse gesteuert. CAPL-Programme werden mit einem eigenen Browser entwickelt und kompiliert. Dabei kann auf alle in der Datenbasis enthaltenen Objekte (Botschaften, Signale, Umgebungsvariablen) und Systemvariablen zugegriffen werden. Darüber hinaus bietet CAPL eine Vielzahl von vordefinierten Funktionen, die das Arbeiten mit dem Entwicklungs-, Test- und Simulations-Werkzeug CANoe und CANalyzer unterstützen.

Marc Lobmeyer (Dipl.-Inf.)
arbeitet seit 1994 bei Vector Informatik als Entwickler an CANoe und CANalyzer.

Roman Marktl (Dipl.-Ing)
arbeitet seit 2012 bei Vector Informatik als Produktmanager im Bereich für CANoe und CANalyzer.