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

Teil 2: CAPL besser verstehen und effektiv anwenden

Der erste Teil dieser Artikelreihe behandelte die grundlegenden Konzepte der Programmiersprache CAPL. Der nun folgende zweite Teil erläutert das Zeitverhalten von Ereignisprozeduren. Zudem gibt es für alle Anwender Tipps für ein effektives Arbeiten mit CAPL rund um die Themen „generisches Programmieren" und „bedingtes Compilieren".

Das Ausführungsmodell

Ein wichtiger Unterschied zwischen CAPL und C sowie C++ ergibt sich aus der Vorgehensweise, wann und wie Programmelemente aufgerufen werden. In C beginnen beispielsweise alle Verarbeitungsabläufe mit der zentralen Startfunktion *main()*. In CAPL hingegen enthält ein Programm eine ganze Sammlung gleichberechtigter Prozeduren, die jeweils auf externe Ereignisse reagieren:

- Vom System ausgelöst: Diese Ereignisse sind einerseits die zur Initialisierung und Nachbereitung des Messungslaufs nutzbaren Ereignisse *on preStart, on start, on preStop* und *on stopMeasurement,* andererseits die Zeitsteuerungs- und Tastaturereignisse *on timer* und *on key.*
- Durch Buskommunikation ausgelöst: Die Ereignisprozeduren, die durch Busereignisse wie Kommunikation oder Fehlerbehandlung auftreten, sind vielfältig und stark bustypabhängig. Beispiele sind hier *on message* und *on busOff* bei CAN oder *on frFrame* und *on frStartCycle* bei FlexRay.
- Durch Zugriff auf ein *Value Object* ausgelöst: Hierbei handelt es sich einerseits um System- und Umgebungsvariablen, die in CANoe/CANalyzer global zur Verfügung stehen, sowie andererseits um Signalwerte, die einer Dateninterpretation der Buskommunikation entsprechen. Das Interpretieren führen spezielle Datenbanken aus. Auf dieses Konzept wird Teil 3 dieser Artikelreihe eingehen.

Ereignisprozeduren sind atomar:

Auch das Simulationsmodell von CANoe ist ereignisorientiert. In Ereignisprozeduren führt CANoe aus Modellsicht alle Aktionen gleichzeitig aus, nämlich zum Zeitpunkt des auslösenden Ereignisses. Die tatsächliche Berechnungsdauer auf einem realen PC wird also vernachlässigt.

Simulationszeit und Zeitstempel:

Ein im PC erzeugtes reales Ereignis, etwa eine Busausgabe durch *output()*, erhält allerdings einen Zeitstempel von der Echtzeituhr. Die Reihenfolge und Zeitpunkte dieser Ereignisse können dabei durch Busprotokolle, Treiber- und Hardware-Eigenschaften beeinflusst werden.

Im Falle eines simulierten Busses entfallen einige der genannten Einflussgrößen. Dort werden die Busereignisse gleichzeitig initiiert, was zum Beispiel bei CAN zu einem gesicherten Arbitrieren mehrerer durch *output()* ausgegebener Botschaften führt.

Aktualisieren von Systemvariablen:

Mit CAPL können Anwender auch außerhalb des Programmes sichtbare Umgebungs- oder Systemvariablen ändern. CAPL vollzieht Wertänderungen einer Variablen erst nach Ende der aktuellen Ereignisbearbeitung, aber weiterhin mit der Zeit des eben behandelten Ereignisses. Ein lesender Zugriff innerhalb der aktuellen Prozedur liefert also auch nach scheinbarer Änderung einer solchen Variablen noch den alten Wert. Das hat den Vorteil, dass zu einem Zeitpunkt nicht mehrere Wertänderungen auftreten.

Das Ausführungsmodell ist situationsabhängig:

CAPL ist in CANoe und CANalyzer vielfältig einsetzbar. Daher variiert auch das Ausführungsmodell etwas: Die Simulationsknoten einer CANoe Simulation befinden sich parallel am Bus. Entsprechend sind sie unabhängig. Einmal ausgelöste Ereignisse werden immer an alle Programme verteilt . Im Unterschied dazu sind die Knoten im Messaufbau und in CANalyzer seriell hintereinander angeordnet: Jeder Knoten reicht seine Ausgaben an den nächsten weiter. Eintreffende Ereignisse benötigen zur weiteren Verarbeitung eine explizite Weiterleitung an den nächsten Knoten. Dafür stehen die Prozeduren *on ** und *on [*]* bereit.

Ein weiterer Programmtyp sind die Testprogramme, deren Testprozeduren auf externe Ereignisse warten können. Bei Eintritt eines solchen Ereignisses fährt CAPL dann mit dessen Simulationszeit fort. Im Gegensatz dazu legt ein Warten in normalen Ereignisprozeduren das gesamte Simulationssystem brach. Dies ist eine häufige Fehlerquelle bei der Anwendung von CAPL. Es ist daher nicht sinnvoll, ein Busy-Wait oder einen Wartebefehl in einer externen DLL zu verwenden.

Tipps zum effizienten Programmieren in CAPL

Der Präprozessor ist in der Sprache C ein mächtiges Hilfsmittel, das aber auch zu Unübersichtlichkeit und damit zu Fehlern führen kann. In CAPL gibt es deswegen nur eine Auswahl der von C bekannten Präprozessor-Direktiven mit vergleichbarer Semantik.

#include:
Include-Dateien enthalten beliebige, aber vollständige, Abschnitte eines CAPL-Programmes: *includes, variables* und *procedures*. Im Gegensatz zu C wird hier nicht einfach der Text der Include-Dateien in die CAPL-Datei eingefügt, sondern die Abschnitte. Dabei gelten alle Abschnitte der Include-Datei in der gesamten includierenden CAPL-Datei „als ob" sie darin enthalten wären. Die Reihenfolge der Abschnitte ist in CAPL sowieso unerheblich. Das bedeutet, der Compiler mahnt doppelt vorhandene Symbole als Fehler an. Außerdem dürfen sich Code und Daten aus includierender und Include-Datei gegenseitig verwenden. Als Ausnahme des eben erklärten Verbots doppelter Symbole dürfen *on start, on preStart, on preStop* und *on stopMeasurement* in der includierenden und in der Include-Datei koexistieren. Bei diesen Funktionen wird der Code nacheinander ausgeführt, erst der Code aus der Include-Datei, dann der Code aus der includierenden Datei.
Include-Dateien übernehmen damit drei Aufgaben: das Deklarieren von Datentypen, das Definieren von Variablen und das Bereitstellen einer (inline-) Funktionsbibliothek.

#pragma library:
CAPL-Programme können in anderen Sprachen erstellte Windows-DLLs mit einer passenden CAPL-DLL-Schnittstelle verwenden. Diese DLLs lassen sich direkt mit der Direktive *#pragma library(„capldll.dll")* einbinden.

Makros:
In CAPL stehen den Anwendern mehrere vordefinierte Makros im Code und für bedingtes Compilieren zur Verfügung. Makros für den Einsatz im Code lassen sich uneingeschränkt überall im Code verwenden. Im Gegensatz zu C ist der Einsatz auch beispielsweise völlig frei innerhalb von String-Konstanten, Variablen- und Funktionsnamen. Sie beginnen und enden immer mit einem %-Zeichen und dienen hauptsächlich zum Schreiben von generischen Programmen. Die verfügbaren Code-Makros umfassen unter anderem den Knotennamen, den Index des aktuellen Kanals, den Namen des aktuellen Netzwerks und den Typ des verwendeten Busses. Auf den Namen der enthaltenden Datei kann der Code mit *%FILE_NAME%* zugreifen, auf den Namen der aktuell übersetzten Programmdatei mit *%BASE_FILE_NAME%*. Letzteres ist bei Include-Dateien die includierende Datei. Hier zwei einfache Beispiele:

```
write(„The node name is %NODE_NAME%");
putValue(envChannel%CHANNEL%Var1, %CHANNEL%);
```

Für das bedingte Compilieren von Code-Abschnitten gibt es einen eigenen Satz vordefinierter Makros. Diese sind *#if, #else, #elif* oder *#endif*. Sie erlauben es, innerhalb eines Programmes nach den Programmarten Simulationsknoten, Messknoten und Testprogramm sowie der verwendeten CANoe-Version zu unterscheiden. Hier ein Beispiel zusammen mit einem *#pragma message:*

```
#if (TOOL_MAJOR_VERSION < 8)
#pragma message(„This program needs at least
CANoe 8 „)
#endif
```

#pragma message:
Die Direktive *#pragma message* ermöglicht es den Anwendern, eine eigene Meldung während eines Übersetzungsvorganges auszugeben, zum Beispiel die Versionsnummer für das gerade übersetzte CAPL-Programm. Diese erscheint zusammen mit den anderen Meldungen, Warnungen, Fehlern und allgemeinen Meldungen des Compilers.

Übersetzung der englischen Veröffentlichung im CAN Newsletter, Ausgabe 3/2014.

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.