ECUテストのプログラミングをもっと効率的に ～ CAPLの基礎と使用上のヒント&コツ ～

パート2: CAPLをより深く知り、効率的に使う

本シリーズのパート1では、プログラミング言語であるCAPLの基本的な概念を取り上げました。パート2では、イベントプロシージャーの時間的な挙動について説明します。また、すべてのユーザーがCAPLを「汎用プログラミング」と「条件付きコンパイル」の分野で効果的に活用できるよう、いくつかのヒントも紹介します。

実行モデル

CAPLとCあるいはC++との主な違いは、プログラム要素が呼び出されるタイミングとその方法にあります。たとえばCでは、処理シーケンスはすべて、一斉開始関数であるmain() から始まります。一方CAPLでは、すべてのプロシージャーが横並びで1つのプログラムに含まれ、それぞれが以下のような外部イベントに反応します。

> システムによるトリガー：これらのイベントには、on preStart、on start、on preStop、on stopMeasurementや時間制御およびキーボードイベントのon timerとon keyなどの、実行する測定の初期化や後処理に役立つものが含まれています

> バス通信によるトリガー：通信やエラー処理に関連したバスイベントに反応するイベントプロシージャーにはさまざまなタイプのものがあり、それらはバスの種類に大きく依存します。これらのイベントには、CANのon messageやon busOffや、FlexRayのon frFrameやon frStartCycleなどがあります

> Value Objectへのアクセスによるトリガー:これらのオブジェクトには、CANoe/CANalyzerでグローバルに使用できるシステム変数と環境変数のほか、バス通信をデータ解釈したシグナル値が含まれます。この解釈は専用のデータベースによって行われます。このコンセプトについては、このシリーズのパート3で取り上げます

イベントプロシージャーはアトミック：

CANoeのシミュレーションモデルはイベント指向です。イベントプロシージャーでは、CANoeがすべてのアクションをモデルの観点から同時に、具体的にはトリガーイベントが発生したタイミングで実行します。PCで生じる実際の計算時間は無視されます。

シミュレーション時間とタイムスタンプ：

ただし、PCによって生成された実際のイベント、たとえばoutput() によるバス出力などには、リアルタイムクロックのタイムスタンプが与えられます。これらのイベントのシーケンスと時間ポイントは、バスのプロトコル、ドライバー、ハードウェアの特性から影響を受けます。仮想バスではそのような影響を及ぼすパラメーターの一部を排除できます。その場合バスイベントは同時に開始され、たとえばCANであれば、output() から複数のメッセージを出力する際、確実にアービトレーションを実施して送信できます。

システム変数の更新：

ユーザーはCAPLを使ってプログラムの外にある環境変数やシステム変数を変更することも可能です。CAPLはイベント処理と同じタイミングで値を変更しますが、その変更はそのイベント処理が完了するまでは変数に反映されません。イベント処理中に読み取りアクセスを行うと、その変数に新しい値が設定されているように見えても、常に古い値が返されます。これにより、値の変化が、ある特定の時点での1回しか発生しえないというメリットがあります。

実行モデルは状況に依存：

CANoeやCANalyzerでは、さまざまな方法でCAPLが使用されます。そのため、その実行モデルも必ずしも一様ではありません。CANoeシミュレーションの仮想ノードはバス上に並行して存在し、それらは互いに完全に独立しています。トリガーイベントは常にすべてのプログラムにて有効です。対照的に、測定設定のノードやCANalyzerのノードは順次処理、すなわち各ノードが次のノードへと出力を渡していく形で処理されます。処理を進めるためには、渡されたイベントを明確に次のノードに渡さなければなりません。on * およびon [*] のプロシージャーは、その目的で用意されています。

その他、テストプロシージャーが外部イベントを待機できるタイプのテスト用プログラムがあります。CAPLの場合、そのようなイベントにはシミュレーション時間を用いて処理を再開します。対照的に、通常のイベントプロシージャーでは、ループ処理など待機状態が発生するとシミュレーションシステム全体が停止します。これはCAPLを使用する場合によくあるエラー原因の1つです。そのため、外部DLLのビジーウェイトやウェイトコマンドは使用しないようにしてください。

効率的なCAPLプログラミングのヒント

C言語のプリプロセッサーは強力なツールである一方、使用を誤ってエラーを招く場合もあります。したがってCAPLでは、C言語でよく知られているプリプロセッサーディレクティブの一部のみをC言語と同等のセマンティクスで提供します。

#include：

include文、変数、プロシージャーなど、CAPLプログラムのセクションがすべて揃った、任意のファイルを読み込みます。Cとは対照的に、includeファイルのテキストは、単にCAPLファイルに挿入されるのではなく、セクションに挿入されます。includeファイルのすべてのセクションが、親となるCAPLファイルにあたかも最初から含まれていたかのように、全体に適用されます。CAPLではセクションの順序にはまったく意味はありません。そのため、コンパイラーは重複しているすべてのシンボルをエラーとして報告します。また、includeファイルと親ファイルのコードとデータは相互に使用することが可能です。

先ほど重複シンボルは避けると述べましたが、これには1つ例外があります。on start、on preStart、on preStop、on stopMeasurement は、includeファイルと親ファイルの両方に存在することもあるということです。これらの関数では、最初にincludeファイルのコード、次に親ファイルのコード、というように順次実行されます。つまり、includeファイルは「データ型の宣言」、「変数の定義」、「(インライン) 関数ライブラリーの提供」という3つのタスクの実行に使われることになります。

#pragmaライブラリー：

CAPLプログラムは、適切なCAPL DLLインターフェイスさえ実装すれば、他言語で作成されたWindows DLLを使用することができます。これらのDLLは 直接、#pragma library("capldll.dll") というディレクティブでリンクできます。

マクロ：

CAPLには多数のマクロが事前に定義されており、ユーザーはそれらをコードや条件付きコンパイルで使用できます。コードで使用されるマクロは、コード内の任意の場所で制限なく使用できます。C言語とは対照的に、マクロは文字列定数、変数の識別子、関数名の中で自由に使用できます。マクロは常に%文字で始まり、%文字で終わります。また、基本的に汎用的なプログラムを記述するのに使用されます。

利用可能なコードマクロには、使用中のノード名、現在のチャンネルインデックス、現在のネットワーク名、バスのタイプが含まれます。このコードは %FILE_NAME% を含むファイル、または、%BASE_FILE_NAME% を含む現在コンパイルされているプログラムファイルにアクセスできます。includeファイルの場合、後者コードは親ファイルを示します。以下に簡単な例を2つ示します。

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

別途、コードセクションを条件付きでコンパイルするための #if、#else、#elif、#endif などのマクロも事前に定義されています。これらを使えばプログラム内で、プログラムタイプの仮想ノード、測定ノード、テストプログラムをはじめ、使用されているCANoeのバージョンも識別できます。下記は #pragma message を使用する例です。

```
#if (TOOL_MAJOR_VERSION == 7 && TOOL_MINOR_VERSION
== 5 && TOOL_SERVICE_PACK < 2) || CANALYZER
#pragma message("This program needs at least CANoe 7.5
SP 3")
#endif
```

#pragma message：

#pragma message ディレクティブを使用すれば、ユーザーはコンパイル処理中に独自のメッセージ、つまり、現在コンパイルされているCAPLプログラムのバージョン番号などのメッセージを出力することができます。このメッセージはコンパイラーからのその他のメッセージ、警告、エラー、通常のメッセージなどと一緒に表示されます。

(別稿パート 3 へ続く)

CAPLとは？

CAPLはVector Informatik（ベクター本社）によって開発された、Cライクな手続き型プログラミング言語です。プログラムブロックの実行はイベントによって制御されます。CAPLプログラムは専用のブラウザーで開発およびコンパイルされます。これにより、システム変数をはじめ、データベースに含まれているあらゆるオブジェクト (メッセージ、シグナル、環境変数) へのアクセスが可能になります。さらに、CAPLは事前定義された関数を数多く用意しており、開発/テスト/シミュレーションツール「CANoe」や「CANalyzer」に活用できます。

本稿は、2014年3月CiA発行の『CAN Newsletter』に掲載されたベクター執筆による記事内容を和訳したものです。

執筆者：

Marc Lobmeyer (Dipl.-Inf.)
1994年よりCANoeおよびCANalyzerの開発者としてVector Informatikに勤務。

Roman Marktl (Dipl.-Ing)
2012年よりCANoeおよびCANalyzerのCAPL機能関連のプロダクトマネージャーとしてVector Informatikに勤務。

■ 本件に関するお問い合わせ先
ベクター・ジャパン株式会社
営業部
（東京）　TEL：03-5769-6980　FAX：03-5769-6975
（名古屋）TEL：052-238-5020　FAX：052-238-5077
E-Mail：sales@jp.vector.com