TOPPERS 活用アイデア・アプリケーション開発コンテスト

部門　：（活用アイデア部門）　アプリケーション開発部門
作品のタイトル　：　Ruby 版 AUTOSAR 向けジェネレータ
作成者　：　富士ソフト株式会社（代表：鴫原一人）
対象者　：　TOPPERS/AP 開発者(特に BSW 開発を行う方)
使用する開発成果物　：
・TOPPERS/ATK2 R1.3.2 (Skyeye 簡易パッケージ)
・TOPPERS/A-COMSTACK R1.2.1
・TOPPERS 新世代カーネル用コンフィギュレータ(cfg.exe)
・TOPPERS 新世代カーネル用コンフィギュレータ仕様書
・TOPPERS 新世代カーネル用コンフィギュレータ内蔵マクロプロセッサ仕様書

目的・狙い

現在 TOPPERS/AP で使用されている TOPPERS 新世代カーネル用コンフィギュレータ(以後、現コンフィギュレータ)は、人が読み書きする静的 API によるコンフィギュレーション情報を対象としたツールであるので、AUTOSAR の人が読み書きしない XML によるコンフィギュレーション情報を取り扱うのには向いていない。そこで、現コンフィギュレータとは別に、AUTOSAR に特化したジェネレータ(以後、新ジェネレータ)を開発するべきと考えた。

アイデア/アプリケーションの概要

- 現コンフィギュレータ仕様から、AUTOSAR で必要となる機能に絞り込んだ新ジェネレータを開発する
- 「Ruby 版 TOPPERS コンフィギュレータ」と同様の理由で Ruby を使用する
- csv 形式の静的 API テーブルは、XML のコンテナ情報を記載し難いので、フォーマットを変更する
- ini ファイル、スキーマファイル等は本質的に不要であるので使用しない
- 各 BSW モジュールの tf で共通的に実装する処理を新ジェネレータに取り込む

1 現コンフィギュレータを AUTOSAR 向けに使用する上での問題点

1.1 入力ファイルの扱い

現コンフィギュレータは、人が静的 API を記述した cfg ファイルを入力することを前提としており、利便性を考慮して、整数型のパラメータにもマクロや計算式を記述できたり、コンパイラを利用してアラインをチェックできるなど、入力データに対する高レベルな機能を有している。しかし、AUTOSAR の XML では、パラメータを定義するコンテナの種別自体がデータ型毎に異なり、一般的なツールでは、厳格なデータ型チェックが行われるため、整数型パラメータにマクロなどの文字列を記述するとバリデーションエラーとなってしまう。また、AUTOSAR においては、入力情報となる XML ファイルは、人が記述することを想定しておらず、ツールによって生成されるものであるので、現コンフィギュレータのような高レベルな機能が必要となる可能性は低い。現コンフィギュレータには、不正な設定値があると、cfg ファイル内の対象の行数を表示する機能があり、XML の場合も XML ファイル内の行数を表示するが、前述の通り、XML を直接人が修正することは考えにくいため、行数ではなく、不正な設定値が入力されたコンテナ情報(参照パスなど)が表示されたほうが便利である。

1.2 現コンフィギュレータの AUTOSAR 対応の経緯

ATK2 開発当初、現コンフィギュレータには XML に対する処理が実装されていなかったため、XML に記述するコンフィギュレーション仕様を静的 API に置き換えた仕様を独自に作成し、静的 API によってコンフィギュレーションを行っていた。しかし、AUTOSAR メソドロジにおいて、OS(ATK2)だけ XML を使用しないわけにもいかず、XML の入力にも対応できるように現コンフィギュレータを修正し、一時、静的 API でも XML でも入力できる状態としていた。その後、XML 仕様の変更の度に静的 API を追従することが困難になったことや、両者をテストするコストの削減等の理由から、静的 API 対応は止め、XML のみ入力するようになった。この結果、ATK2 の tf には、静的 API と XML の差分を埋めるためのコードが散在してしまっている。仮に、最初から XML のみ対応するツールを開発するという流れであれば、現コンフィギュレータを使用せず、独自のツールを開発していたと考えられる。

1.3 pass1 を省略することの弊害

AUTOSAR においてジェネレータが必要となるモジュールは数十個存在するが、OS のようにマイコンやコンパイラに依存しないモジュールが過半数を超える。現コンフィギュレータでは、マクロなどで記述された数値を評価する処理(pass1)において、コンパイラを必要とするが、AUTOSAR のマイコンやコンパイラに依存しないモジュールにおいては、ジェネレータでコンパイラが必要となるのは不便である。これに対し、現コンフィギュレー

タにおいて、pass1 を飛ばし、pass2 だけを実行するオプション(--omit-symbol)を用意したが、pass1 を飛ばすことで、整数型に文字列を記載された場合などのチェックをすべて tf 側で処理する必要が生じている。

1.4 データ構造の差異

静的 API は、1 つのオブジェクトのコンフィギュレーション情報を 1 行(1 つ)の静的 API によって記述するため、リスト形式でコンフィギュレーションデータが tf に渡される。これは静的 API を処理するには最適な設計である。しかし、AUTOSAR の XML では、オブジェクトのコンフィギュレーション情報を階層形式で記述される。現コンフィギュレータでは、階層的に記述されたコンフィギュレーション情報をリスト形式で tf に渡すため、tf では必ず階層的に紐づく情報同士を関連付ける処理が必要になってしまう。また、リスト形式では、XML に記述された情報を直感的に参照することができず、開発・デバッグ効率が悪い。

1.5 ツールの名称

AUTOSAR において"ジェネレータ"と呼称しているツールは、TOPPERS における"コンフィギュレータ(cfg.exe)"を指している。また、AUTOSAR における"コンフィギュレータ"は、コンフィギュレーション情報(XML)を生成するツールを指している。この名称の差異により、AUTOSAR のジェネレータとして cfg.exe を使用すると、TOPPERS/AP 利用者の混乱を招く恐れがあるので、AUTOSAR 向けジェネレータは cfg.exe とは別ツールであることが望ましい。

1.6 小数の取り扱い

AUTOSAR には小数で定義するパラメータが多く存在する。現コンフィギュレータでは小数を扱えないため、"."の左右を抜き出し個別に演算する等の対応が、tf 側で必要となってしまう。つまり、AUTOSAR の小数を扱う全モジュールで冗長したコードを記述することになる。

1.7 実行モジュールのサイズ

現コンフィギュレータの実行モジュールは、元々5MB 程度のサイズであったが、XML 用ライブラリをリンクすることで、30MB というサイズになっている。静的 API を使用するだけの場合には、不必要に実行モジュールのサイズが大きくなってしまうことになる。

2 Ruby によるジェネレータ開発

前章で述べた現コンフィギュレータの問題点を解決するには、現コンフィギュレータの使用を止め、AUTOSAR 向けのジェネレータを別途開発する必要があると考えた。

まず、多くの開発者が使用する tf は、可読性、実装容易性、習得容易性を重視する必要がある。これを実現するには、独自言語をやめ、一般的に使用されている言語を採用するのが妥当と考え、簡潔に処理を記述でき、可読性が高いことで定評がある Ruby を選定した。Ruby は、日本発のプログラミング言語であり、世界的にも普及率が高いため、TOPPERS プロジェクトが採用する言語としても妥当である。なお、tf ファイル(tf で記述されたファイルを指す)相当のファイルを Ruby で実現したファイルの拡張子は、rb ではなく、trb とすることで明確に区別することにした。

Ruby は、実行中の Ruby ファイルから別の Ruby ファイルを呼び出して実行することができることから、コンフィギュレータ自体も Ruby で開発することにした。これにより、コンフィギュレータが解釈したコンフィギュレーション情報をそのままのデータで、trb ファイルに渡すことができる。

2.1 文字列＋数値の廃止

tf では、ユーザがシステムコンフィギュレーションファイル(以後、cfg ファイル)に記述した文字列と、その文字列をコンパイラを用いて評価した数値の 2 つのデータを 1 つの変数として保持するが、前章で述べた通り、AUTOSAR においては厳格なデータ型チェックが行われるべきであり、整数型のパラメータには整数値しか設定されない前提としてよい。従って、文字列＋数値という概念は、新ジェネレータでは廃止する。これに伴い、srec ファイル、syms ファイルに対する処理も不要となる。

2.2 ファイル出力

tf は、テンプレートファイルの名前の通り、基本的には tf ファイルに書いた内容を、ファイルに出力する前提となっており、出力内容を演算結果によって変更したい箇所にのみ$で囲んだマクロ命令を記述する形式となっている。しかし、現状の tf ファイルは、演算処理の実装の方が多くなり、テンプレートファイルという形式を取るメリットは小さくなっている。そこで、新コンフィギュレータでは、テンプレートファイル形式ではなく、ファイル出力クラスを用意し、出力する文字列をインスタンスに追加していき、最後にファイル出力を行うメソッドを呼び出す形式とした。

なお、Ruby はヒアドキュメントを使用できるので、まとまった固定的な文字列は tf 同様に、そのまま記述することができる。tf のように改行(NL)を明示的に記述する必要がないため、tf より可読性、実装容易性が高い。

2.3 trb に渡されるコンフィギュレーション情報

現コンフィギュレータでは、静的 API テーブルに記述された識別子をベースに、リスト形式でコンフィギュレーション情報が tf に渡される。新ジェネレータでは、XML に記述されたコンテナ情報をベースに、ハッシュ形式で trb へコンフィギュレーション情報を渡すこ

とにした。これにより、リスト形式の情報を階層的に紐付ける作業が不要となり、直感的にコンフィギュレーション情報を把握することが可能となった。

各コンテナには、エラー発生時の情報としてショートネームによるパス(:SN_PATH)と、同一コンテナの 0 から連番の ID(:ID)を付与して渡す。

```
{"SIGNAL_A"=>↓
  {:DEF_REF=>"/AUTOSAR/EcucDefs/Com/ComConfig/ComSignal",↓
   :ComBitPosition=>3,↓
   :ComBitSize=>1,↓
   :ComHandleId=>5,↓
   :ComSignalDataInvalidValue=>"true",↓
   :ComSignalEndianness=>"LITTLE_ENDIAN",↓
   :ComSignalInitValue=>"false",↓
   :ComSignalType=>"BOOLEAN",↓
   :ComTransferProperty=>"TRIGGERED",↓
   :ID=>0,↓
   :SN_PATH=>"/Com/Com/ComConfig_0/SIGNAL_A",↓
   :ComFilter=>↓
    {:DEF_REF=>"/AUTOSAR/EcucDefs/Com/ComConfig/ComSignal/ComFilter",↓
     :ComFilterAlgorithm=>"ALWAYS",↓
     :ID=>0,↓
     :SN_PATH=>"/Com/Com/ComConfig_0/SIGNAL_A/ComFilter"}},↓
```

図 1　trb に渡されるハッシュデータの例(ComSignal)

2.4 スキーマファイル使用の廃止

現コンフィギュレータでは、AUTOSAR から公開されている XML バリデーション用のスキーマファイルを使用している。しかし、現コンフィギュレータが対象としている ECU コンフィギュレーション情報を記述した XML は、パラメータのカスタマイズ容易性を高めるために、パラメータ毎にタグを定義するのではなく、パラメータ名を値として設定する形式となっている。このため、スキーマファイルによるバリデーションを行っても、ECU コンフィギュレーションの設定値までのバリデーションはできない。

スキーマファイルによるバリデーションがなくても、ECU コンフィギュレーションを記述した XML であるかどうかは簡単にチェックできるので、ユーザにスキーマファイルをダウンロードして配置してもらってまで実施する必要はないと考え、スキーマファイルの使用を廃止した。

2.5 csv ファイルのフォーマット変更

現コンフィギュレータでは、静的 API に対する静的 API 情報テーブルのフォーマットを踏襲し、tf へ渡す対象とするコンテナの情報を csv ファイルで定義する。この csv ファイルを作成する元となっているパラメータ情報は AUTOSAR から公開されている。
(AUTOSAR_MOD_ECUConfigurationParameters.arxml)
このパラメータ情報には、列挙型(ENUM)のパラメータに指定可能な値の情報も入っているが、csv フォーマットでは、列挙型の値の定義が困難であり、現コンフィギュレータでは、列挙型のパラメータに対する値の正当性チェックは、tf で実施している。新ジェネレータで

は、列挙型もパラメータ情報として入力することで、列挙型のパラメータに対する値の正当性チェックは新ジェネレータ側で行い、trb では不要とした。

また、AUTOSAR_MOD_ECUConfigurationParameters.arxml には、整数型、浮動小数点型に対する最大値、最小値の情報も入っているので、この情報も新フォーマットに記載し、新ジェネレータ側で設定値のレンジチェックを行うようにした。

ファイルフォーマットは YAML とし、図 2 のようにパラメータパスをキーとしたハッシュ形式で、データ型(:TYPE)、多重度(:LOWER、:UPPER)、列挙型に指定可能な値(:ENUM)、設定可能な最大値(:MAX)、最小値(:MIN)を記述する。csv ファイルではコメントが記載できないが、yaml ファイルであれば、多重度や列挙型の値を改変した際に、理由や要求タグをコメントとして残すことが可能である。なお、csv ファイル同様、yaml ファイルもツール(ABREX)により、AUTOSAR_MOD_ECUConfigurationParameters.arxml から生成可能とした。

```
/AUTOSAR/EcucDefs/Com/ComConfig/ComSignal/ComSignalInitValue:↓
  :TYPE: STRING↓
  :LOWER: '0'↓
  :UPPER: '1'↓
/AUTOSAR/EcucDefs/Com/ComConfig/ComSignal/ComSignalLength:↓
  :TYPE: INT↓
  :LOWER: '0'↓
  :UPPER: '1'↓
  :MAX: 4095↓
  :MIN: 0↓
/AUTOSAR/EcucDefs/Com/ComConfig/ComSignal/ComSignalType:↓
  :TYPE: ENUM↓
  :LOWER: '1'↓
  :UPPER: '1'↓
  :ENUM:↓
  - BOOLEAN↓
  - FLOAT32↓
  - FLOAT64↓
  - SINT16↓
  - SINT32↓
  - SINT8↓
  - UINT16↓
  - UINT32↓
  - UINT8↓
#  - UINT8_DYN  # 可変長データは未対応 [NCOMxxx]↓
  - UINT8_N↓
```

図 2　パラメータ定義情報の例

2.6 ini ファイルの廃止

現コンフィギュレータでは、AUTOSAR 向けに使用する際は設定情報を記述した ini ファイルを入力する必要がある。しかし、新ジェネレータではスキーマファイルを使用しないこと、パラメータ情報ファイル(yaml)に記載されたパラメータのみを trb に渡すことに限定することで、ini ファイルは不要とした。

3 現コンフィギュレータとの互換性確認

A-COMSTACK Release 1.2.1 の A-COM、A-CANIF を用いて、AUTOSAR のコンフォーマンステストを実施し、全件パスすることを確認した。

また、新ジェネレータ開発において、現コンフィギュレータおよび A-COM の tf ファイルにおける問題点を数多く検出した。新ジェネレータでは、これらの問題を解決済みである。tf ファイルに比べて、trb ファイルのコード行数は、どのモジュールも約 40%程度少なくなるため、新規 BSW の開発効率の向上も期待できる。

4 新ジェネレータのメリット

「ショートネームの重複チェック」や「ID が 0 から連番となっているかチェック」など、多くのモジュールで必要となる処理は、新ジェネレータ側で対応することにより、trb の冗長コードをなくすことができた。その他にも、boolean 判定関数(IS_TRUE)や、小数用割り算関数(DIV)など、tf であるがゆえに全 tf で必要となってしまう関数定義をなくすことができた。

```
#↓
# ID連番ソート＆連番チェック↓
# [COM401] ショートネーム重複はgen.rb側でチェック済み↓
#↓
↓
# [COM394_Conf][NCOM008] ComConfigurationId↓
$hCom[:ComConfig] = SortObjectByID($hCom[:ComConfig], :ComConfigurationId)↓
↓
# [COM175_Conf][NCOM008][NCOM003] ComIPduHandleId↓
$hCom[:ComConfig].each{|sCfgID, hCfgData|↓
  hCfgData[:ComIPdu] = SortObjectByID(hCfgData[:ComIPdu], :ComIPduHandleId)↓
}↓
↓
# [COM184_Conf][NCOM008] ComIPduGroupHandleId↓
$hCom[:ComConfig].each{|sCfgID, hCfgData|↓
  hCfgData[:ComIPduGroup] = SortObjectByID(hCfgData[:ComIPduGroup], :ComIPduGroupHandleId)↓
}↓
↓
# [COM165_Conf][NCOM008] ComHandleId↓
$hCom[:ComConfig].each{|sCfgID, hCfgData|↓
  hCfgData[:ComSignal] = SortObjectByID(hCfgData[:ComSignal], :ComHandleId)↓
}↓
```

図 3　冗長コード削減の例(ID ソート＆連番チェック)

リスト形式ではなく階層形式のハッシュデータでコンフィギュレーション情報を受け取れることで、より直感的に処理が記述できるようになり、開発コストを大きく削減できることを実感した。また、break、continue(next)といった制御文が使用できること、複数行のコメントアウト(=begin/=end)ができること、カバレッジが容易に取得できることなど、開発・デバッグ効率の向上が期待できる。

5 今後の展望

新ジェネレータの開発により、tf より trb のほうが開発効率が高いことが確認できた。また、tf という独自言語を学ばずとも、Ruby でジェネレータ開発が行えることで、

TOPPERS/AP 開発のハードルを低くすることができる。

ATK2 に関しては、現コンフィギュレータの高レベルな機能を多く使用しているので、新ジェネレータを単純に使用するだけでは移行は困難であるが、「Ruby 版 TOPPERS コンフィギュレータ」をクラス化しておき、ATK2 の trb から include して機能を使用するなどして、高レベルな機能が必要な BSW モジュールのみ使用するといった使い方を検討したい。

以上