ソフトウェアの動作検証支援システム ARVE

ARVE Aspect-Oriented Runtime Verification Environment

進　博正　■ SHIN Hiromasa
遠藤　侑介　■ ENDOH Yusuke
片岡　欣夫　■ KATAOKA Yoshio

東芝は，ソフトウェアの動作検証支援システム ARVE（Aspect-oriented Runtime Verification Environment）を開発した。このシステムは，検証対象の観測方法や分析方法をアスペクト指向のインタプリタ言語(注1)のスクリプト(注2)で記述し，自動的にスクリプトを解釈しながら動作検証することができる。また，このシステムを用いることによって蓄積した動作検証用のスクリプトを再利用することで，目的に応じたソフトウェアの動作検証環境を効率よく構築できる。

Toshiba has developed a software verification system called the aspect-oriented runtime verification environment (ARVE) to support runtime verification during software development.　This system provides a scripting environment to automate observation and analysis tasks in runtime verification.　By developing or reusing scripts for ARVE, users can effectively build a runtime verification system to meet their specific objectives.

1　まえがき

高性能なマイクロプロセッサの低価格化を背景に，身近な電子機器が大規模なソフトウェアを搭載することが多くなり，ソフトウェア製品の効率的な開発技術[1]が求められている。開発工程を設計と製造に分けると，ソフトウェアの開発は，ハードウェアの開発と比較して，大半の工程が設計に集中する点が特徴的である。ソフトウェアの品質保証には，設計工程の欠陥除去がすべてであり，設計の初期段階で不具合を取り除くフロントローディング設計と，設計の最終段階で欠陥を取り除く動作検証（又はテスト）が有効である。

品質要求の高いシステムソフトウェア（コンパイラ(注3)やオペレーティングシステムなど）製品の開発では，動作検証に多くの時間を費やすことが普通である。例えば，動作検証の目的で作成するソフトウェアの規模が，製品となるソフトウェアの規模を上回ることもある。一般のアプリケーションソフトウェアは，機能の豊富さと比べて開発期間が短いことが多く，システムソフトウェアと同じレベルの動作検証は難しい。検証の目的で作成するソフトウェアの生産性を改善できれば，動作検証の効率化に役だつと考えられる。ソフトウェアの再利用に関する検討は，これまで製品となるソフトウェアに対してが中心であり，動作検証用のソフトウェアに対しては十分に検討されていない。

（注1）　プログラムされた命令を，コンピュータが直接実行可能な形式にあらかじめ変換することなく，逐次解釈しながら実行することを意図して作られたプログラミング言語。
（注2）　命令を処理する手順をインタプリタ言語で記述した簡易プログラム，又はその処理手順。
（注3）　プログラミング言語で書かれたプログラムを，コンピュータが実行可能な形式に変換するソフトウェア。

東芝は，ソフトウェアの動作検証を短期間で効果的に行うため，動作検証に用いるソフトウェアの再利用を向上する動作検証支援システム ARVEを開発した。ここでは，ARVEの機能の概要及び応用事例を中心に述べ，フロントローディング設計との関係についても述べる。

2　システムの概要

ソフトウェアの動作検証システム ARVE[2]は，アスペクト指向の考え方を取り入れたインタプリタ言語を用いて，動作検証の対象（以下，検証ターゲットと記す）の観測方法や分析方法

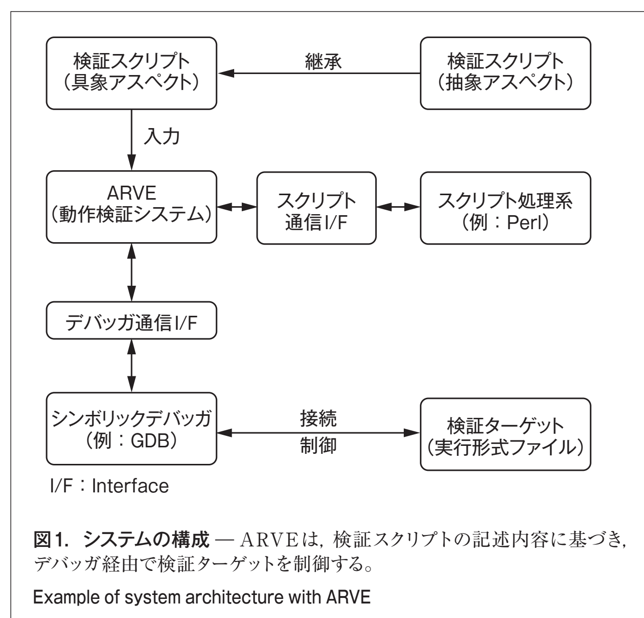

図1.　システムの構成 ― ARVEは，検証スクリプトの記述内容に基づき，デバッガ経由で検証ターゲットを制御する。

Example of system architecture with ARVE

をスクリプト（以下，検証スクリプトと記す）に記述し，そのスクリプトを逐次解釈しながら自動的に動作検証するシステムである（**図1**）。ARVEは，検証スクリプトの内容に従って検証ターゲットを検証するために既存のデバッガ(注4)を通して行うので，従来と同様に，組込みシステムを含む広範囲な動作環境に対応している。

2.1　アスペクト指向スクリプト

アスペクト指向[3]は，プログラミング言語の備えるモジュール化機構を拡張する近年の考え方の一つである。アスペクト指向を取り入れたアスペクト指向プログラミング言語の例は，AspectJ(注5)が有名である。AspectJは，オブジェクト指向（OO：Object Oriented）プログラミング言語 Java(注6)のモジュール化機構（クラス）を拡張して，複数のクラスに関与する処理内容をモジュール化する機構（アスペクト）を備える。アスペクト指向のモジュール化機構は，OOプログラミング言語への反省から生じたものの，プログラムコード上に一定の法則で散在する処理のモジュール化に有益なため，OO言語以外のプログラミング言語との組合せも有益である。

ARVEは，利用者が検証プログラムを記述するために，アスペクト指向のモジュール化機構を備えたインタプリタ言語を提供する。

検証プログラムの記述にアスペクト指向を用いる利点は，検証ターゲットに散在する処理（内部状態の観測や分析など）を検証スクリプトの形にモジュール化できる点にある。また，検証スクリプトの間の継承関係を用いると，検証スクリプトの再利用性を高められる。

検証内容の記述にインタプリタ言語を用いる利点は，検証内容を動作環境から独立に記述できる点，及び検証スクリプトの変更結果（検証対象の修正や追加など）をすぐに確認できる点にある。欠点は，実行時のオーバヘッドが大きい点にあるが，用途を動作検証に限定することで許容できる。

なお，現在のARVEが提供するインタプリタ言語は，インタプリタ言語Perl(注7)にAspectJの構文の一部を加えた言語である。

2.2　シンボリックデバッガの制御

ARVEは，検証スクリプトと検証ターゲットの結合にシンボリックデバッガ(注8)を利用するため，デバッグコンパイルで生成した実行形式ファイルを検証ターゲットとする。デバッガを用いる利点は，デバッガが大半のソフトウェア開発環境で利用できる点，デバッガが対象プログラムの記述言語（C/C++言語など）や動作環境（CPUアーキテクチャなど）への依存性を隠蔽（いんぺい）する点，及び動作検証の開始や終了のタイミングに関する自由度が高まる点にある。例えば，検証ターゲットが実行中でも，検証スクリプトの開始や終了を自在に行える。反対に欠点は，実行時のオーバヘッドが大きい点にあるが，先のスクリプトと同様に許容できる。

ARVEは，複数のデバッガに対応しており，現在はGNU（GNU is Not UNIX(注9)）やMicrosoft社の開発環境のコンパイラやデバッガ（GDB(注10)，WinDbg(注11)など）と組み合わせて利用できる。デバッガは，統合開発環境のGUI（Graphical User Interface）などがデバッガの機能を利用するために，制御インタフェースを提供している。ARVEもこの制御インタフェースを利用するが，個々のデバッガへ依存する部分を少なくするため，内部にデバッガドライバ層を設けている。新しいデバッガを利用する場合は，新たなデバッガドライバを作成して追加するだけでよい。

ARVEのデバッガドライバのインタフェースを**表1**に示す。このインタフェースを用いたデバッガ制御の概要を述べる。ARVEは検証セッションを開始するためデバッガの実行を開始（start）し，検証ターゲットを選択（load又はattach）する。検証スクリプトの内容と検証ターゲットのシンボル情報を検索（lookup）して，ブレークポイントの設定位置を決める。ARVEは検証ターゲットへブレークポイントを設定（insert）し，検証ターゲットの実行を開始（run又はcont）して，ブレークポイント待ち（wait）に入る。ARVEは，ブレーク位置で関連するアスペクトを評価し，検証ターゲットの実行を継続（cont）する。検証セッションを終了するときは，検証ターゲットのブレークポイントを除去（remove）した後，デバッガの実行を終了（stop又はdetach）する。

表1. デバッガ制御の概要
Debugger control interfaces of ARVE

名　称	概　要
start	デバッガの実行を開始する。
stop	デバッガの実行を終了する。
query	デバッガのコマンドを実行する。
load	ターゲットを選択しプロセスを初期化する。
attach	ターゲットを選択しプロセスに接続する。
detach	ターゲットのプロセスから切断する。
lookup	ターゲットのシンボルを検索する。
insert	ターゲットへブレークポイントを設定する。
remove	ターゲットからブレークポイントを除去する。
return	現在のサブルーチンを終了するまで実行を継続する。
run	ターゲットの実行を開始する。
cont	ターゲットの実行を継続する。
wait	ターゲット実行中にブレークポイントの発生を待つ。

（注4）　プログラムのバグの発見や修正を支援するソフトウェア。
（注5）　AspectJは，米国Palo Alto Research Center, Inc.の登録商標。
（注6）　Javaは，米国Sun Microsystems, Inc.の米国及びその他の国における登録商標又は商標。
（注7）　Perlは，米国Larry Wall氏らが開発したインタプリタ言語。
（注8）　ソースコード上の対応箇所を参照できるようになっているデバッガ。
（注9）　UNIXは，商標。
（注10）GDBは，米国GNUプロジェクトが開発したシンボリックデバッガ。
（注11）WinDbgは，米国Microsoft社が開発したシンボリックデバッガ。

3　システムの応用

ARVEを利用したソフトウェアの動作検証を説明するため，検証ターゲットが内部的に行うファイルの操作手順を例に述べる。

3.1　ファイル操作の手順

ソフトウェアとは計算機の行う処理の手順を記述するものであり，設計者の意図どおりに動作するソフトウェアには処理の手順に一定のパターンが存在する。例えば，ソフトウェアがファイルを操作する場合は，操作対象のファイルを開いた（open)後，必要な読み書き（read|write）を行い，不要になれば閉じる（close）という手順を経る。ファイル操作に関する正常な操作の順番は，正規表現（regular expression）と呼ばれるパターン記述の規則[4]を用いて“open（read|write)*close”と表現できる。ソフトウェア動作中の操作列がこのパターンを満たすことは，ソフトウェアが正常に動作していることの必要条件となる。

3.2　検証スクリプトの事例

与えられた操作列が正規表現で記述したパターンを満たすかどうかは，パターン記述の正規表現から決定性有限オートマトン（DFA）を構成して効率よく検査する方法[4]が知られている。ソフトウェアの動作中に操作列の検査処理を行い，パターン違反が発生すると検証ターゲットの実行を中断し，内部状態を報告する処理などを記述できる。

アスペクト指向を取り入れたインタプリタ言語を用いると，抽象アスペクト（abstract aspect）に正規表現の検査アルゴリズムを記述し，抽象アスペクトを継承した具象アスペクト（concrete aspect）に，操作に対応する対象プログラム上の位置，操作順序を規定するパターンやパターン逸脱を検出したときの処理内容を分けて記述できる（**図2**）。

図2の抽象アスペクトの概要は以下のとおりである。抽象アスペクトは，パターン検査用DFAの内部状態を保持する変数，コンストラクタ（開始前のサブルーチン），デストラクタ（終了時のサブルーチン）とアドバイス（実行中に所定の場所で呼ばれるサブルーチン）から成る。コンストラクタは，正規表現のパタンを受理するDFAを$M=(Q, \Sigma, \delta, q0, F)$と計算し，DFAの内部状態を記述する変数を初期値q0とする。デストラクタは，終了時のDFAの内部状態が受理状態Fになければ，エラーを報告する。アドバイスは，呼ばれた位置aと現在のDFAの状態qに対応する遷移が存在するときはDFAを遷移させ，対応する遷移が存在しなければエラーを報告する。なお，QをDFAの状態の有限集合，Σを操作の記号集合，δをDFAの遷移関数（$Q\times\Sigma\rightarrow Q$），q0をDFAの初期状態，FをDFAの受理状態（$F\subseteq Q$）と表記した。

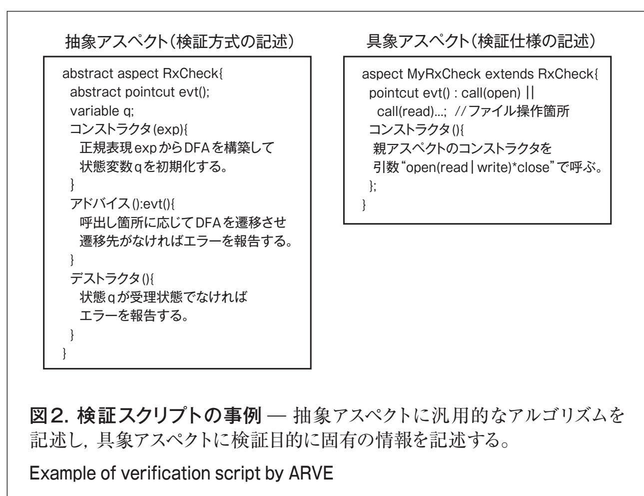

図2. 検証スクリプトの事例 ― 抽象アスペクトに汎用的なアルゴリズムを記述し，具象アスペクトに検証目的に固有の情報を記述する。

Example of verification script by ARVE

図2の具象アスペクトの概要は以下のとおりである。抽象アスペクトを継承し，アスペクトのポイントカット式の宣言により観測箇所を指定する。親クラスのコンストラクタに操作パターンを正規言語で指定する。エラー発生時の呼出し処理があれば，対応するサブルーチンを記述する。

図2のように，検証スクリプトの間の継承関係を用いて，汎用的なアルゴリズム部分（抽象アスペクト）と，検証対象に応じたカスタマイズ部分（具象アスペクト）に分けると，前者を再利用性の高い検証スクリプトとして分離できる。例えば，同じ検証スクリプトを用いて通信ソケットの操作手順に関する動作検証プログラムを作成できる（**図3**）。

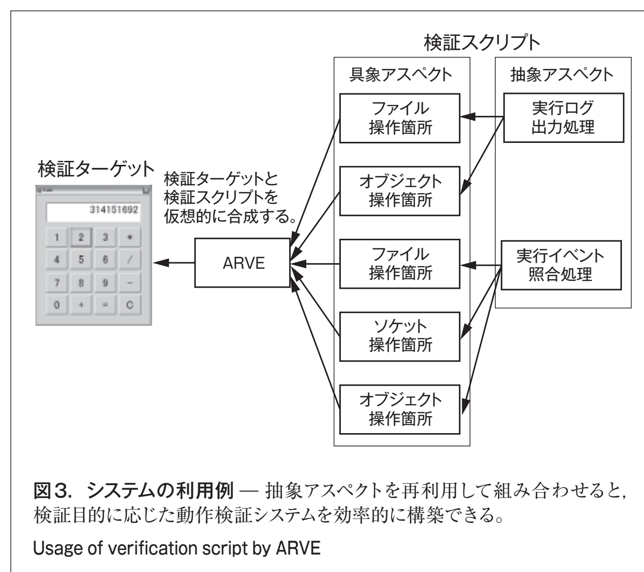

図3. システムの利用例 ― 抽象アスペクトを再利用して組み合わせると，検証目的に応じた動作検証システムを効率的に構築できる。

Usage of verification script by ARVE

3.3　検証スクリプトの活用

ソフトウェアの動作検証の詳細内容は，検証ターゲットに応じて千差万別であるが，検証ターゲットによらない部分をARVEの検証スクリプトの形で部品化できる場合もある。あらかじめ検証スクリプトの部品が存在すれば，図3に示すように，これら部品を組み合わせてカスタマイズすることで，動作検証の目的に応じた動作検証システムを効率よく構築できる。

この章で述べた形式言語に基づき動作検証を行う検証スクリプトは典型的な部品の事例であるが，もう少し単純な部品も考えられる。例えば，動作確認用のログ出力処理なども再利用可能なスクリプトの部品とし蓄積できる。ほかに，メモリなどの資源リーク(注12)を調べるため，資源の獲得や解放を記録して検査するスクリプトなども考えられる。またARVEを利用すると検証ターゲットを修正しなくとも，サブルーチンの差替えが可能であり，システムコールの失敗を模擬した障害試験のためのスクリプトも考えられる。多様な動作検証に応じた検証スクリプトの部品を蓄積して再利用することで，動作検証の効率改善が期待できる。

4　上流設計と動作検証

ソフトウェアの動作検証では“正しい動作”を定める検証仕様を過不足なく準備することは一般的に難しい。3.1 節で述べたファイル操作の場合は正規表現で記述したパターン記述が検証仕様の例となるが，“正しい動作”に関する必要条件であり十分条件でない。ただし，このような検証仕様は，必要条件でも動作検証には十分に有効なことが多く，また実装の詳細によらず設計の早期段階に本質的には決まることが多い。

設計の早期段階で設計仕様を計算機処理の可能な形式で記述できれば，動作検証に用いる検証仕様の生成に活用できる。例えば，上流設計時に形式的な仕様記述言語を用いてシステムの仕様を記述すると，動作確認で用いる検証仕様を形式的な仕様記述からアルゴリズム的に抽出するという考え方である。

現在，適合性テストの自動化の観点から，形式化したソフトウェアの仕様から，動作検証向けの検証仕様を生成する技術を開発している。

5　あとがき

ソフトウェアの動作検証支援システムARVEは，利用者が検証内容（観測対象や分析方法など）を記述するために，アスペクト指向のモジュール化機構を持つインタプリタ言語を提供する。アスペクト指向は，検証内容を検証ターゲットから分離して検証スクリプトに部品化するのに有効である。インタプリタ言語は，検証スクリプトの移植性や可変性を高める。ARVEの利用者は，部品となる検証スクリプトを開発して，部品を組み合わせることで，目的に応じた動作検証システムを構築できる。ARVEは，検証ターゲットと検証スクリプトの結合に既存のデバッガを利用するため，多様な動作環境に対応できる。

動作検証の有効性を高めるには，過不足のない検証仕様の準備が課題となる。今後，早期段階の設計仕様から動作検証に用いる検証仕様の生成技術も開発していく。

（注12）プログラムの不具合により，計算に利用した資源（メモリなど）が利用後に解放されず，資源が不足する現象。

文　献

(1) 片岡欣夫，ほか．特集：高信頼性組み込みソフトウェア開発．情報処理．47，2，2006，p.486－525.

(2) Shin, H., et al. ARVE: Aspect-oriented Runtime Verification Environment. In Proceedings of 7th International Workshop on Run-time Verification, to be published in LNCS. Oleg Sokolsky, Serbar Tasiran. Vancouver, British Columbia, Canada, 2007-03, Runtime Veritication 2007. Germany, Springer-Verlag.

(3) 千葉　滋．アスペクト指向入門．東京，技術評論社，2005，264p.

(4) J. ホップクロフト，ほか．オートマトン言語理論 計算論．東京，サイエンス社，2004．337p.

進　博正　SHIN Hiromasa
研究開発センター システム技術ラボラトリー研究主務。
ソフトウェアの設計技術の研究・開発に従事。
情報処理学会会員。
System Engineering Lab.

遠藤　侑介　ENDOH Yusuke
研究開発センター システム技術ラボラトリー。
ソフトウェアの設計技術の研究・開発に従事。
日本ソフトウェア科学会会員。
System Engineering Lab.

片岡　欣夫　KATAOKA Yoshio, Ph.D.
研究開発センター システム技術ラボラトリー主任研究員，博士（情報科学）。ソフトウェアの設計技術，品質向上技術の研究・開発に従事。情報処理学会会員。
System Engineering Lab.