アクティブソフトウェアからアクティブ・ウェブへ

京都情報大学院大学　教授
渡邉　勝正

新しく開発されるソフトウェアシステムは，要求の高度化に伴って，論理構造が複雑になり，規模が大きくなっていく一方である。これに対応する設計と実装の方針として，階層化と再利用と合わせて，アクティブソフトウェアの考えがある。

アクティブソフトウェアは，能動的に起動するモジュールや関数を要素として，ソフトウェアを構成する方法で，並列処理や分散処理に適合している。

一方，インターネットの発展で，ウェブ上の情報は急激に増大している。増加の傾向は益々強くなり，単純な検索方法では，真に希望するページを見つけることがむつかしくなる。

本論文では，アクティブソフトウェアの考えを述べて，ウェブプログラムの構成に適用することを提案する。ウェブ上の検索やセキュリティなどの問題が関連する。

1. はじめに

新しく作成されるソフトウェアへの要求の高度化に伴って，その論理構造は複雑になり，規模が大きくなっていく。一方，その設計と開発の期間を短縮すること，使用環境や要求仕様の変化に柔軟に対応することが求められている。

これに対応する設計方法と実装の方針として，アクティブソフトウェア（Active Software）の考えがある。[rl2000]

アクティブソフトウェアは，能動的に起動するソフトウェアモジュールや関数を要素としてソフトウェアを構成する方法で，並列処理や分散処理に適合している。ソフトウェアの階層的構成や，コンポーネントウェアと再利用などと関連している。また，ソフトウェアに動的な検証機能や誤りの検出機能をもたせることに適している。[kw2005]

一方，インターネット利用の拡大発展に伴い，ウェブ上の情報（ホームページ）が増大すること，ウェブサービスなどのウェブプログラミングの役割が大きくなることの傾向が見られる。増加の傾向が益々強くなるに従って，検索によって，真に希望するページを見つけたり，適切なウェブサービスを探したりすることがむつかしくなっている。また，外部からの不正な侵入や攻撃に対処することの重要性が高まっている。[ao2004]

本論文では，まず，アクティブソフトウェアの基本になる考えと特徴，その設計法を述べる。次いで，その構成法をウェブプログラミングや，ウェブ上の検索，および，セキュリティなどの問題に適用する方法を考察する。

「アクティブ」（Active，能動的）ということばは，「アクティブ制御」や「アクティブネットワーク」，あるいは，「アクティブ学習」など，技術の面や生活の面で様々に使われている。人間の活動の根本であり，組織や経営において求められる行動である。

2. 能動モジュールとアクティブソフトウェア

アクティブソフトウェアの構成原理は単純である。ソフトウェアの稼動中に，ある特定の「条件」が成立したときに行う「動作」を，次のように表記する。

定義1：@起動条件｛関数またはモジュール｝　□

これを能動関数（active function）または能動モジュール（active module）という（以下では，能動関数とする）。広い意味の，イベント駆動方式，あるいは，ルールベース実行である。ただし，1つのソフトウェア全体を，この形式だけで構成して表現するのではなく，一部の関数やモジュールに起動条件を自由に追加・記述することで良い。

例題1：株価の大幅な変動を検出して，取引を制限する。

```
ストップ高：
  @((新値-昨日の終値)>昨日の終値*上昇変動率)
  {ストップ高の警告を発生して,処置する手順}
ストップ安：
  @((昨日の終値-新値)<昨日の終値*下降変動率)
  {ストップ安の警告を発生して,処置する手順}
```

実際には，昨日の終値の範囲によって，変動幅が定められている。このような株式の値の動きを監視することと処理の考えは，為替レート（ドルと円のレート）の変動の監視と，それに伴う方策の問題にも共通する。　□

能動関数の起動は，計算の途中でいつでも条件の検出をするように指定しても良い場合（これを，“いつでも”モード，または，$モードという）と，検出する時点を適宜指定する場合（これを，“このとき”モード，または，@モードという）とがある。計算の途中で，どちらかのモードを指定する。

定義2：起動文で，起動モード（$または@）と対象の変数を指定する。

起動モード　対象変数のリスト（[実引数の並び]）；　□

計算の途中で，対象変数の値が更新されたとき，その変数を式の中に含む起動条件を評価して，起動モードに従って，関数本体（または，モジュール本体）を起動させる。起動文（定

義2)と起動条件(定義1)の関連は次のように結び付けられる。

定義3:{起動文で指定した変数vを起動条件iが含んでいる}かつ {起動条件iの論理式の値は真である} □

これは，関数名を指定して関数を直接呼び出すのではなく，起動条件に含まれる変数を通じて間接的に起動を駆けることである。"このとき"モード(@モード)では，複数の関数が起動することがある。

例題1a:$ 新値();の起動文によって，注目した株価が変化するたびに，例題1の2つの起動条件が評価される。この場合は，どちらか1つが真値になることがある。 □

例題2:サイズ Nの配列A(int A[N])について，添え字に使われている変数の値(k)をチェックする。

能動関数側:

```
@(N<=(k+2));/*サイズを越える可能性*/
int overIndex(){/*添字が限界を越えた処理*/}
@(k<0);/*添え字が負になる可能性*/
int negativeIndex(){/*添字が負になった処理*/}
```

配列参照側:

```
@k();/*参照の前に添え字の値をチェックする*/
A[k+2]=A[k]+A[k+1];
```

添え字の値kが，どちらかの条件に抵触することがある。配列の不当参照を防ぐ。 □

例題3:在庫管理では部品や商品の在庫数を常に監視して，製造や販売の損失を防ぐ。たとえば，自動車の製造では，ある車種の右ドアのガラスと左ドアのガラスの在庫数量は同じである。また，その数は，定常的な生産ラインを止めない限界値以上である。

能動関数側:

```
@(右ドアのガラス!=左ドアのガラス);/*ガラスの数量が異なる*/
int unEqualDoorGrass(){/*数量の検査と補充を促す処理*/}
@(右ドアのガラス<限界値*安全率);/*在庫切れを防ぐ*/
int littleStock(){/*在庫を補充する手配を促す*/}
```

生産計画側:

```
@右ドアのガラス();/*新しい製造指示の前に在庫をチェックする*/
newProductPlan(K);/*K台の自動車を製造する指示*/
```

この場合は，2つの起動条件が同時に真になることがある。 □

このような能動関数を用いて，アクティブソフトウェアの基盤を築くことができる。アクティブソフトウェア (active software) の概念は，文献 [rl200] で導入されたもので，つぎのような能動性の特徴をもっている。

(1)自分の状況を知る(self awareness)
(2)自分を調整できる(self regulating)
(3)自分の誤りを訂正する(self correcting)
(4)モジュールが移動する (nomadic or self moving)

つまり，計算の進行状況や計算環境の変動に対応して，能動的に処理を選択できること，目標とのずれを予測して動的に自分を調整できること，などの特徴をもつソフトウェアである。これによって，「複雑さと多様な変化に耐える強靭で(robust)安全な(safe)ソフトウェア」を実現することを目指している。

ここでは，アクティブソフトウェアの広汎な目標ではなく，その内の特徴(1)(self awareness)に焦点を当てている。それが他の特徴(2, 3, 4)を実現する出発点となる。

3. 動的検証と動的検出

能動関数は，プログラムのある時点で，動作{Q}の前条件(P)と後条件(R)によるアサーション検証法 P{Q}R [hoare1969] を動的に実現することに適用できる。これは，ある処理動作{Q}をする前に満足していなければならない条件(P)と処理動作{Q}の後で成り立つ条件(R)を示すことによって，プログラムの正しさを検証しようとするものである。

起動条件に(P)や(R)を持つ能動関数で，実行の正当性を確認するとともに，起動条件に(Pの否定，_P)や(Rの否定，_R)を持つ能動関数によって，正当でない実行状態を検出することができる。

例題4：2点(x1, y1)，(x2, y2)の線分の傾き(勾配)を計算する。この場合，線分が垂直(x1 == x2)のケースを除かないといけない。

前条件(P:x1!=x2)
動作 {Q:勾配=(y2−y1)÷(x2−x1)}
後条件(R:桁溢れなし)

これを能動関数によって表現する。前条件(P)とその否定(_P)をチェックする。

能動関数側:

```
@(x1==x2);/*垂直線の場合*/
float verticalCheck(){/*垂直線の場合の処理手順*/}
@(x1!=x2);/* 垂直線でない場合*/
float nonverticalLine(){勾配=(y2-y1)/(x2-x1);}
```

処理本体側:

```
@x1, x2();/*勾配を事前チェックする*/
/*勾配を用いた処理*/
```

後条件(R)のチェックは省略している。処理の後でRが成り立たない場合(_R)の発生も検出すると良い。 □

例題5：商品取引において，受注(商品名，受注数量，売り上げ単価)を受けたとき，受注処理をする前に，受注条件の妥当性をチェックする必要がある。

前条件(P:受注数量<=在庫量，売り上げ単価>=最低額)
動作 {Q:受注処理を行う}
後条件(R:計算式[売上金=売り上げ単価*受注数量]や在庫量の変化を確認する)

これを能動関数によって表現する。前条件(P)の否定(_P)を

チェックする。また，後条件（R）で処理の確認を行う。

能動関数側：

```
@（受注数量>在庫量）;/*_P:在庫不足*/
int zaikoHusoku（）{/*商品の生産または仕入れ処理をする*/}
@（売り上げ単価<最低額）;/*_P:受注単価が適切でない*/
int tankaKakunin（）{/*単価が低い理由を確認する*/;}
@（売上金!=売り上げ単価*受注数量）;/*_R:計算の間違い*/
float uriageAyamari（）{/*計算の照合結果を報告する*/;}
```

処理本体側：

```
@受注数量，売り上げ単価（）;/*_P:受注処理の事前チェック*/
受注処理（商品名，受注数量，売り上げ単価）;
@ 売上金，在庫量（）;/*R:受注処理の結果を確認する*/
```

処理前のチェックによって，取引上の誤りを防ぐことができる。とくに，株式の取引では，取引の株数と取引価格のチェックは重要である。処理後のチェック（_R）も不可欠である。 □

このように，能動関数は，「何時，どのような状態になったら，何をするか」を設計と構成の基本にするものである。それは，「何かの処理をするときに，どのような状態であるべきか」を確かめることと表裏の関係にある。能動関数は実行中のソフトウェアの異常事態を発見したり，正しく動作していることを確認したりすることに適している。

しかし，このような検出や確認を，つまり，対応する起動条件の評価を，プログラムの実行中に逐次に行うことは，実行効率の上で好ましいことではない。幸いなことに，コンピュータの性能が向上して，処理時間の遅延が目立たなくなっている。でも，応答時間に厳しいリアルタイム処理や，ソフトウェアの規模に合わせて能動関数の数が増えるものについては，重大な問題点になる。

これを改良するには，ある時点で評価すべき起動条件を厳選しておくこと，あるいは，起動条件を評価するハードウェア回路を構成して並列に実行することである。前者は，能動関数を含んだプログラムを言語処理するときの問題である。

後者は，能動関数の起動条件をハードウェアに変換する問題である。変換によって生成されたハードウェア記述から，書き換え可能なハードウェア（たとえば，FPGA:Field Programmable Gate Array）回路を構成する。その回路は，主プロセッサがプログラムに従って動作するのと並列に，起動条件の評価を行う。ある時点に対応する起動条件が真になると，主プロセッサに割り込み信号を送って，能動関数の本体を実行させる。

信号の大まかな流れを図1に示す。[kw2005]

（1）図中のレジスタ（たとえば，A，B，C）は，起動条件に使われている変数の値をもつ記憶用のレジスタである。これらの値は，メモリバスを監視していて，主プロセッサがメモリに書き込むときに，新しく取り込まれる。

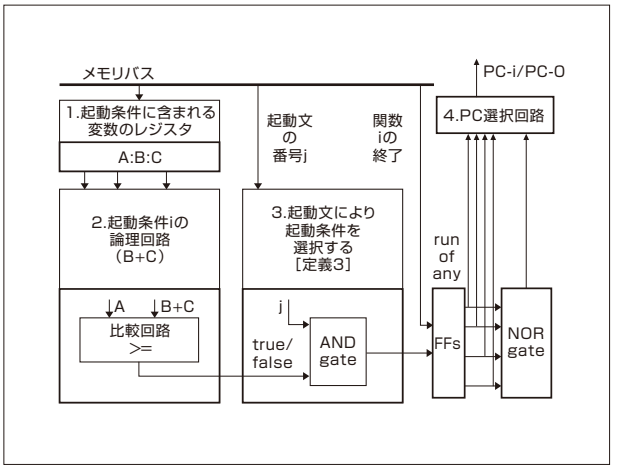

図1　書き換え可能な起動条件評価回路

（2）起動条件の論理回路は能動関数iの起動条件iを表す論理式の評価回路である。

図では，条件（A>=B+C）の比較回路を持っている。

（3）選択部は，起動文jについて，評価の対象となる起動条件iを選択する回路である。比較回路の論理値iと信号jの両方が1（真）のとき，選択回路のANDゲート（論理積）が1（真）となる。

その出力は，起動条件iに対応するフリップフロップi（1ビット記憶レジスタi）を1に設定する.

（4）PC選択回路では， フリップフロップiの出力が1になると，対応する能動関数iのプログラムカウンタi（PC_i）を取り出して，主プロセッサに知らせる。実行スケジュールの対象に加えられる。

能動関数iの実行が終了すると，対応するフリップフロップiを0（偽）にリセット（クリア）する。すべてのフリップフロップが0になると，NOR（論理和の否定）の出力が1となり，PC_0により，待機していた起動文jの次に実行の制御を戻す。

できれば能動関数の本体も書き換え可能な回路として実現すればよい。起動条件の評価や能動関数の本体を可変のハードウェアで実装すると，ソフトウェアの仕様の変更や，計算環境の変化に対しても，能動関数を差し替えることによって，柔軟に対応しやすい。

このような再構成可能コンピュータアーキテクチャは， 一般的な例外処理を効率良くサポートして，システムの実働を安定に保つ環境を与える。[mt2004]

4. ウェブプログラミングへの適用

アクティブソフトウェアの考えは，ウェブの環境にも適用できる。インターネット利用の拡大につれて，とくに，（1）外部からの不正な侵入と攻撃，（2）増大するウェブページの検索が重大な問題になっている。能動関数によってこれらの対応策を立てたい。

（1）については，まず，ウィルスの侵入がある。これに対しては，アンチウィルスやワクチンソフトウェアとして工夫されている。これには，ウィルスのパターンが判っていることが前提となっている。そのパターンが判れば，図1と同じように，ウィルスをチェックする再構成可能な回路を実現できる。比較回路に，ウィルスの実行パターンを持たせて，主プロセッサが実行する命令列を監視する考えである。パターンの照合には，たとえば，命令の実行履歴をハッシュ値に変換する方法がある。［ao2004］

つぎに，攻撃のひとつにDOS攻撃（Denial of Service）がある。大量にパケットが送られてきて，本来の仕事ができなくなる。これに対応するには，入ってくるパケットの個数を入り口でチェックすることである。

例題6：受信パケット数を制限する能動関数

@（パケットの個数 > 同時に受け入れるパケットの上限数）

¦/*DoS 攻撃と判断して，以後の受け入れを停止する*/¦　□

（2）のウェブ検索については，ウェブページの増大が急激で，その増大が留まらないことが問題の根源である。それにより，検索結果が大量になり，現在でも，直接，それをすべて観ることは到底できない状況である。

例題7：たとえば，キーワード“Google検索”で検索するとつぎの件数が出てくる。

YAHOOでの検索件数 3,380,000

Googleでの検索件数 43,300,000

.comのみ，日本語のみとすると，1,210,000に絞られる。

この検索結果の数値にどのような意味があるのだろうか。□

このような現象に対して，表示順位の決め方，順位を上げるページの作り方などで，いろいろの工夫がなされている。たとえば，他からの参照リンク数によってページに格付けをする方法が知られている。しかし，それで本当に観たいページが得られるのだろうかという疑問が残る。この問題についての解決法は単純には得られない。能動関数を用いて，つぎのような点に対応することを検討している。

a. 検索結果から同じURLを削除して表示件数を減らす。このとき，重なりの多いページの順位を上げる。

b. 多義語のキーワードについては，不要の意味をはずして，検索結果を絞る。

c. 逆に，同義語をもつキーワードについては，同義語による検索の幅を広げて，気がつかない検索結果を示す。

d. 検索の結果を見て，関連する他のサイトを紹介する。

e. ホームページを探すのに，検索だけに頼ることの弊害を避ける。

f. 古くなって不要と思われるページを対象からはずす。

アクティブソフトウェアの基盤になっている能動関数について，起動条件を表現する水準（レベル）を上げる必要があるかも知れない。今後に向けての問題である。

例題8：検索結果の「次のURL」が「既にリストしているURL」と重複しているとき，これをリストに含めない。

@（次のURL<=>リストされているURL［i］）;/*重複している*/

int removeURL¦「次のURL」はリストに入れない，URL［i］の順位を上げる¦　□

5. おわりに

能動的な働きは，日常の生活においても，組織の活動においても，期待される行動である。アクティブソフトウェアは，その概念をコンピュータソフトウェアに導入しようとしている。そのような動きをプログラムとして表現することは容易であるが，動きを効率よく実行するためには，並列に動作するハードウェアとの協調性が求められる。とくに，“いつでも”モード（$モード）では，「起動条件を常に監視して評価するハードウェア」の効果が期待される。

アクティブの考えは，インターネットおよびウェブの領域においても効果をもつ。外部からの攻撃に備えることと，検索要求に対して適切なURLを返すことに，有効な方策を提供したい。また，第3の問題として，増えていく一方のウェブページの中で対象にするものをどのように限定していくかに，ひとつの解決策を見出す手掛かりを与えていきたい。

アクティブソフトウェアの4つ目の特徴である「モジュールが移動する（nomadic or self moving）」を実現するとき，「何時，何処に移動するか」を決定する上で，能動関数を適用できると考えている。

■ 参考文献

［rl2000］Laddaga, R.: Active Software, Self-Adaptive Software, First International Workshop, IWSAS 2000, LNCS 1936, Springer, 2000, pp.11-19.

［kw2005］渡邉勝正，井上晶広，伴野充，蔵川 圭，中西正樹，山下茂：能動関数によるアサーション検証設計，コンピュータソフトウェア，Vol. 22, No. 3, 2005, 7, pp.76-91

［ao2004］岡崎篤也，中西正樹，山下茂，渡邉勝正：CPUによるマルウェアコードの実行防止，コンピュータセキュリティシンポジウム2004（CSS2004），2004, 10, pp.421-426.

［mt2004］伴野充，中西正樹，山下茂，渡邉勝正：再構成可能ハードウェアを用いたイベント指向型計算とその応用，第4回リコンフィギャラブルシステム研究会論文集，2004，9，pp.103-109.

［hoare1969］Hoare, C.: An axiomatic basis for computer programming, Comm.Of ACM, Vol.12, 1969, pp.576-580.

渡邉　勝正
Katsumasa Watanabe

京都大学工学部助手，助教授を経て，福井大学工学部教授，奈良先端科学技術大学院大学教授。2006年4月より，京都情報大学院大学教授。これまでに，ソフトウェアシステム，ハードウェア／ソフトウェア協調設計，並列・分散アルゴリズム，アクティブソフトウェアの研究を進めてきた。現在，人やコンピュータのアクティブな振る舞いに関心を持っている。