計算機アーキテクチャ　93－9
（1992. 3. 23）

リアクションにもとづくリアルタイムＡＩのためのＯＳ

An Operating System for Real-Time AI, based on Reactions

鈴木達郎　　杉村利明　　菅原昌平　　日比野靖
Tatsuo Suzuki, Toshiaki Sugimura, Shouhei Sugawara, Yasushi Hibino
NTT ヒューマンインタフェース研究所
NTT Human Interface Laboratories

[Abstract]　Real-Time AI Systems in the real world have various features besides time constraints. IT is shown that reactions are effective for such features, and an operating system can be constructed with a reactive mechanism as a basic function. Reactions, which are multiplexed with priorities, are realized as independent reactive function calls driven by software interruptions in a process.

１．はじめに

現実の世界を相手にしたリアルタイムAIシステムは時間的な制約だけでなく多くの問題（特徴）を持っている。本稿では、外界に即応するリアクション処理がこれらに有効であることを示し、リアクション機構を基本機能として提供するOSの構成法を示した。
リアクションは、各プロセスにおける強制関数呼び出し(reactive function call)として実現する。このリアクション処理を優先度に基づいて多重制御することで、即応性や柔軟性などのリアルタイムAI処理に必要な機能の多くを効率的に実現できることを示した。
このシステムはLISPマシンELISの基本OSカーネル機能として実現している。[1,2]

２．リアルタイムＡＩとは

これまでもリアルタイム処理と呼ばれるものは存在する。制御システム、監視システム、オンライン処理など現実の世界と強く結びついた応用を持っている。またわれわれ人間も現実の世界の大きな要素であり、これを対象とするヒューマンインタフェースでもリアルタイム処理は重要な位置を占めてきている。
ここで述べるリアルタイムＡＩの対象とするものも、従来からのリアルタイム処理と同じである。しかしその手法は異なる。
従来はリアルタイム性を設計時に設定していた。すなわち、すべての状況を予想しておき、最悪の場合でも問題をこなせるように、設計時に一定の処理時間を設定しておくという考え方である。リアルタイム性を保証するために、「処理時間を前もって正確に見積ること」が、重要な課題であった。しかし、予想した最悪の場合を越えた時は、突然悲劇的な結果になる事が多い。

これに対してリアルタイムＡＩは、現実の世界という不確定な環境の下での柔軟で臨機応変な対応を目指したものである。リアルタイムＡＩでも一定時間内の応答が不可欠であるが、大きな違いは、その一定時間が前もってわからないということである。[3]
従って、時々刻々変化する現実の世界を相手に、その場その場で事態の緊急度を把握し、許された範囲で最大限の努力を行うようにプログラムしなければならない。

リアルタイムＡＩでは、いざという時にはすぐに「それなりの」回答を出せるように常に準備して置かなければならない。(Realtime Constraint)[4]
そのために、環境の変化に即座に対応する「その場しのぎ」的な解や、長期的な展望にもとづくプランニングなど時間要素の異なる多くの方法を並行に行い、事態に即して使い分ける必要がある。

３．現実世界の特徴とリアクションの重要性

リアクションとはセンサからの信号（またはそれに対する処理結果）に直ちに反応して実行する処理のことである。リアルタイムＡＩが対象とする現実の世界は以下のような特徴を持っているが、リアクションはこれらに対して有効である。

（１）動的環境(dynamic environment)

考えている最中にもどんどん環境が変化する。ある時点の情報をもとに厳密にスケジュールして処理を行っても、その処理の途中で、状況が大きく変化してしまい、せっかくの計算結果がもはや使えないかも知れない。環境とのインタラクションにより、変化を迅速に検出し、その時々での新しいゴールを定めねばならない。

長期的な視点での計画も、変化に応じて再構成が必要になる。リアクションにもとづく再プランニングやリフレクションが重要になる。

（2）不確定さ(uncertainty)

初期のＡＩにおける「積木の世界」では世の中すべてがわかっていたので、すべてを見通したプランニングも可能だったが、現実世界の環境においては、すべての知識を持つことは不可能である。
完全な情報のない状態で目的を遂行するには、「よくわからないけどとりあえずやってみる」というリアクション的動作が行動上も重要になってくる。人工昆虫ロボットなどは、比較的簡単なリアクションだけでも十分知的に行動することが知られている。[5]

（3）環境との大量の情報のやりとり (highly interactive)

現実の世界では、環境とのインタラクションが頻繁にかつ大量にある。すべての情報を直接処理するのは大変なので、センサなどで前処理を行い、重要なことが生じた時のみ、情報がＡＩシステムに渡される構成を取ることが多い。重大時にはリアルタイムＡＩは緊急に対応しなければならない。
また大量の入力情報を均等に処理するのではなく、場合によっては、ある程度予測して能動的に選択する必要がある。状況に応じて入力の対象の範囲や頻度を制御する焦点制御(focusing)や、場合によっては環境を変えるような行動も重要である。
以上述べてきたように、リアルタイムＡＩが必要とする即応性や変化に追従できる柔軟性の多くは、リアクションによって実現することができる。また、リアクションと推論やプランニングなどと組み合わせることの効果も大きい。

４．リアルタイムＡＩの動作例

ロボットなどの移動体が現実の世界の中を動きまわる処理を例に取って考える。

＜問題設定例＞
移動体はある目的の場所に到達することを最終ゴールとしている。大体の地図などの情報は持っているが、途中でどんな障害があるかは行ってみないとわからない。

プログラムは以下のようなアクションの集合体として作ることができる。
(1) ゴールへ行くまでの道を、地図をもとに計画し、いくつかのサブゴールを設定する。
(2) サブゴールに向かって進む。
(3) 障害物に近づきすぎたら、離れる。
(4) 障害物にぶつかったら、移動を停止する。壊れていないか、再出発できるか、自己チェックする。
(5) ある時間が経っても、サブゴールに近づかないなら（見つけた新たな障害物を地図に登録するなどして）計画を立て直す。
(6) 電源／エネルギー源などに異常を見つけたら、助けを呼ぶ。

これらのアクションは、それぞれ優先度付けされた独立の動作だと捉えることができる。電源異常は最優先である。障害物などに無関係の時は、単にサブゴールを目指して進めばよい。障害物に出くわした時は、サブゴールのことは後回しにして、障害物を回避することを優先する。
これらの動作は、対応する条件（センサなどの処理の結果、構成されるイベント）に基づいており、条件が満たされるとリアクションとして起動される。
この例において、イベントと優先度（数が大きいほど優先度が高い）の割付は、例えば以下のようにすればよい。

	優先度	イベント	処理
(1)	4	初期処理	プランニング
(2)	6	(1)の結果	サブプランニング
(3)	12	（障害物との距離＜α）を検出	反対方向に移動
(4)	16	衝突検出	停止／検査／再開
(5)	8	（時間経過＞β）を検出	リプランニング
(6)	20	異常検出	停止／連絡

この様な設定で、簡単なシミュレーションを行うと、障害物を避けながら目的地に近づくのが見られる。
一般のリアルタイムＡＩ処理でも、この例のように、短期間のリアクションから長期間のプランニングまでの様々のアクションが混在した集合として動作が行われると考えられる。

想定されるリアルタイムＡＩシステムの全体的な構成は例えば図1.のようになる。

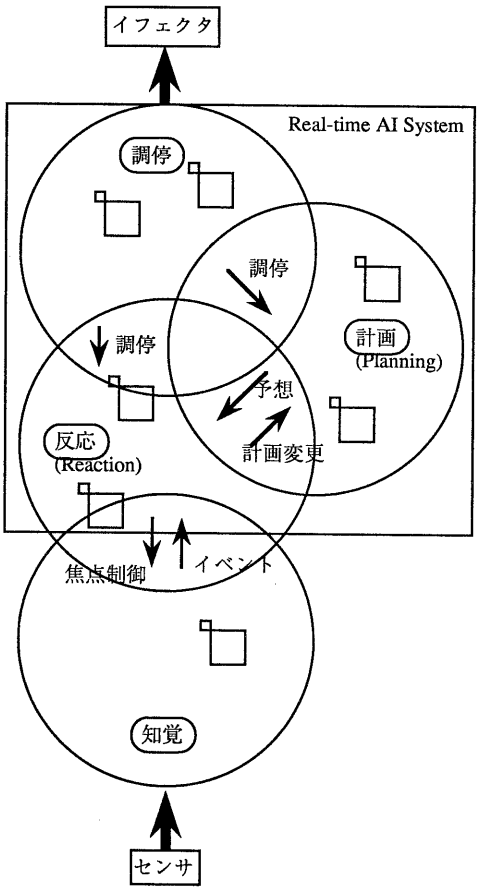

図1. リアルタイムAIシステムの構成

5．リアクションの実現法と特徴

リアクションとはセンサからの信号（またはそれに対する処理結果）に直ちに反応して実行する処理のことである。リアクションはセンサからの要求による割り込み処理と考えることができる。割り込み処理は正に外部からの非同期で緊急の処理要求に対処するためもので、プロセッサにはそのためのサポート機構が用意されている。(ELISの場合はマイクロプログラムで監視している。)
しかし、ハードウェアによる割り込み処理は、種類も限られており、大きな処理をするのにも適していない。また、特別なハンドラを用意するなど、アプリケーションに開放されていない特別な処理である。

そのような制約から逃れて、アプリケーションとして自由に記述するため、プロセスに対するソフトウェア割り込みとして、リアクションを実現する
ることにした。

リアルタイムＡＩ処理におけるリアクション実現の要求条件をまとめると以下のようになる。

（1）複数のリアクション処理が共存する多重構造にすること。
（2）リアクション処理ごとに優先度を持ち、完全プリエンプション方式で実行順序を制御できること。
（3）リアクションどうしで、同期／通信などの制御が可能なこと。
（4）リアクションの駆動／抑制／中止などの制御が行えること。
（5）リアクションの起動におけるオーバヘッドを小さくすること。

ＬＩＳＰマシンＥＬＩＳの上に、これらの要求条件を満たすように、リアクションをサポートするリアルタイムＯＳを作成した。
実現の考え方は、リアクションの契機となるソフトウェア割り込みが発生すると、対応するプロセスでの現在のコンテキスト上で、新たに別の関数（リアクション処理）呼び出しを強制挿入して実行させる(reactive function call)、というものである。
この「ソフトウェア割り込みによる関数実行」という考え方は、ＥＬＩＳ上の言語（かつＯＳ）ＴＡＯで提供されているし[6]、ＵＮＩＸでの signal 処理などでも、ソフトウェア割り込み(kill)を契機に、関数が実行される機能を持っている。
本システムでは、これをさらに発展させて、ソフトウェア割り込みによる動的な関数呼び出し即ちリアクションを一つの独立な実行単位とした。
各リアクションに独立な優先度を持たせ、多重化を可能にし、スケジュールの対象としたことが特徴である。リアクションの起動（これを駆動／drive と呼ぶ）に伴うオーバーヘッドは、
通常の関数呼び出し程度に抑えられることが期待できる。
また言語処理系から見ると各リアクションの実行は関数コールになっている。状況の変化のため古くなったリアクションの畳み込みなどに言語機能を用いることができ、上に述べたリアルタイムＡＩの特徴に合った制御が可能である。

6．アクションとイベント

6.1 イベント

プロセスに対してソフトウェア割り込みをかける時、オリジナルなＴＡＯではプロセスを指定して起動する形式を取っていた。ＵＮＩＸの signal 機能でも、関数 kill でプロセスＩＤを指定している。
ここでは、イベントと呼ぶプロセスと独立な実体を介して、間接的に駆動を行うようにした。このため、緊急事態を検出する（センサなどの）処理は、それに対応するリアクションの処理を実行するプロセスの存在を意識する必要はなく、対応するイベントを発火するだけでよい。
またプロセス側は、特定のイベントが発火した時に行うべきアクションを、前もって登録しておくだけでよい。自動的にその時が来れば駆動されることになる。
イベントとプロセスとリアクション駆動の関係を図2.に示す。

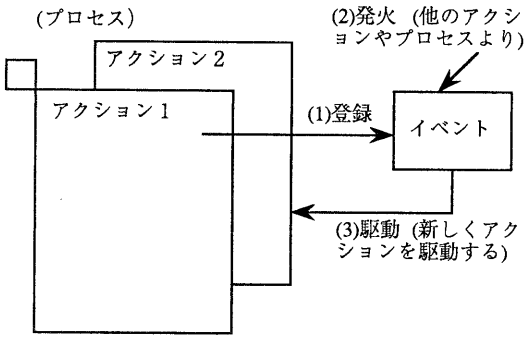

図 2. イベントによるプロセスのアクション駆動

6.2 リアクションの優先度

イベントに対してリアクションを登録する時に優先度を指定する。イベントにより駆動されたリアクションは、そのイベントの優先度を自分の優先度とする。したがって、優先度の高いイベントにより駆動されたリアクションは高い優先度を持ってスケジュールされる。

7．アクションの多重制御

7．1　プロセスの状態遷移

プロセスは複数のリアクションの集合からなる。プロセスの実行とは、これらのリアクションが駆動されることである。それぞれのリアクションは、独立に実行状態(ready, wait, run, stop)と優先度を持つ。
あるプロセス内で優先度の一番高いリアクション（代表リアクション）の状態が、そのプロセス全体の実行状態と優先度（代表優先度）として扱われる。複数のプロセスどうしは、それぞれの代表優先度でスケジュールされる。スケジュールは完全プリエンプション方式で、優先度の高いプロセスは、待ち状態の時に限って、優先度の低いプロセスに制御明け渡す。
一つのプロセス内のリアクションどうしのスケジュールも完全プリエンプション方式で、優先度のみで制御される。ただし、代表リアクションがたとえ待ちになっても、優先度の低い他のリアクションには制御は移らない。この場合は、他の（代表優先度の低い）プロセスに制御が移ることになる。
なお、同一プロセス内で優先度の同じリアクションどうしの場合は、「遅いもの勝ち」とし、最新の状況にもとづくものを優先することができる。（現在のバージョンでは速いもの勝ちである）
駆動要求が出ているが、まだ実行が開始されていない（代表アクションより優先度が低いため割り込めない）リアクションは、プロセスごとにキューに蓄えられる。そのリアクションより高い優先度のリアクションが終了後に、割り込みが成功して、始めて実行される。

リアクションとプロセスの状態遷移を図3.に示す。

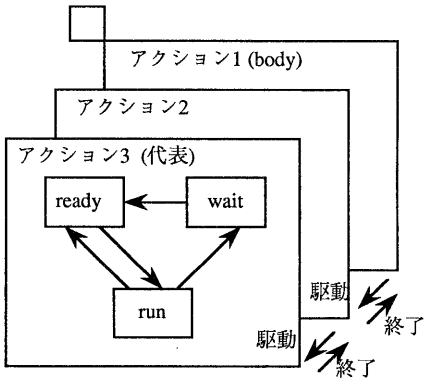

図 3. アクションとプロセスの状態遷移図

7．2　イベント管理

イベントはプロセスと独立な存在である。イベントに対するプロセスからの命令は、登録と発火と制御に分かれる。「登録」とは、そのイベントが発火した時に、プロセス自身が実行すべきリアクションを、前もって指定しておくことである。登録時には、リアクションの実行関数と引き数の組および優先度を指定する。プロセスがあるイベントにアクションを登録した時、そのプロセスとイベントとは結合されているという。登録時にfor-everを指定すると、陽に解除しない限り、ずっと有効であり、何度でも駆動実行される。（デフォルトは発火ごとに自動的に解除される設定になっている。）
「発火」とは、そのイベントを発生させることである。その結果、結合してあったプロセスに対してリアクションが駆動される。また、発火させるプロセスから、駆動されるプロセスへトークンとして値を送ることも可能である。
イベントに対する「制御」には、
（1）発火しても駆動を行わない、無効制御
（2）溜っている発火の保存制御
（3）複数のプロセスと結合している場合、駆動するプロセスを選択する方式
(a) ラウンドロビン：発火要求ごとにプロセスを一つづつ順に繰り返し駆動する。
(b) ブロードキャスト制御：発火要求ごとにプロセスを同時に駆動する。
などがあり、細かいリアクション制御が可能になっている。

7.3 リアクション間の協力

イベント駆動などで生成されたリアクションどうしは互いに協力して処理を進めることが必要である。リアクションはプロセスではないが、以下のような協力が可能である。

(1) リアクション間のデータ共有：
同一プロセス内のリアクションどうしは関数入れ子の関係なので言語としてのグローバル変数はそのまま共有される。
また共有パッケージという拡張LISP機能が提供されている[7]。共有パッケージ内のグローバル変数を参照することで異なるプロセス間のデータ共有が実現される。

(2) リアクション間の同期／通信：
各リアクションどうしがイベントを介して同期を取る機構が提供されている。あるリアクションが関数 wait-event を実行して「待ち」の状態になっている時、これを優先度の高い別のリアクションまたは別のプロセスから関数 signal-event により再開することができる。

(3) 非同期入出力の実現
入出力デバイスの終了割り込みを別のリアクションで受けることで非同期入出力が実現できる。入出力を命じたもとのリアクションは入出力終了を待たずに他の処理を継続できる。(その間に別のリアクションが非同期に入出力処理を実行する。)
入出力結果が必要な時点になれば結果が既に得られているかどうかで同期を取ればよい。ELISではこの方法でメールボックス機能を実現している。

(4) 実行中のリアクションの変更/解除
環境の変化のため、実行中のリアクションを、変更や解除することは、リアルタイムAIにおいて重要である。ある期間過ぎたものや、条件の合わなくなったものを削除したり、再設定する機能が必要になる。
タイマイベントなどで変更期日を設定し、内側のリアクションから関数を畳み込むことで実現できる。

(5) 焦点制御
リアルタイムAIシステムでは、状況に応じて注意を向ける範囲を限定したい場合がある[8]。
例えば、4章の例で、障害物との距離測定に前後左右の4方向があった時、前進している時は前、後退している時は後ろに注意を向けることを焦点制御と呼ぶ。
焦点制御は、イベントの優先度を再設定したり、有効無効を設定したりすることで可能である。また無効期間中に溜まった発火要求をためておくか、捨ててしまうかの指定も可能である。

8．考察

非同期な処理の切り替えを通常はプロセスの切り替えによって実現している。応答性を重視するリアルタイム処理では、このプロセス切り替え時間が性能に大きく影響してくる。
ＭＡＣＨのthreadなどの「軽いプロセス」は、プロセスの走行環境を切り替えないことでオーバヘッドを小さくしている。本システムのリアクション制御も、非同期な強制関数呼び出し(reactive function call)で処理を切り替え、走行環境は切り替えないことでオーバヘッドの短縮をねらっている点は共通である。（LISPマシンELISでは全空間実メモリであり、もともとプロセスごとの走行環境変更のオーバヘッドは実は大きくない）

ただし、リアクションの多重化は、言語から見ると、実質的に関数呼び出しの入れ子になっている。最低限の環境の保存／復帰などの処理は
言語レベルで対処されている。そのため、関数入れ子を扱う言語機能（例えば、ＬＩＳＰの catch/throw、unwind-protect）などをリアクション間の制御に効果的に利用できることが特徴になっている。
また、リアクション処理の新規生成／削除が関数コール／リタン程度の手間で実現することができるので、次々に発生するリアクション処理に適している。
一方、関数呼び出しの入れ子は、内側（呼ばれる側）が終了しない限り、外側（呼ぶ側）へは制御が戻ってこない。従って、高優先度のリアクション内でＩ／Ｏなどの待ちが生じても、内側の低優先度のリアクションには制御が移らない。この場合は他のプロセスに制御が移ることになる。同一プロセス内でのリアクション間は厳密な優先度制御になっているとも言える。
ただし、優先度の高いリアクションは、実行を「反射的に開始し、すぐ終了する」という性質を通常持っており、他のリアクションが待ちになることは少ない。また４章に述べた例での衝突検出や異常検出の例のように、他の処理をすべて止めて実行するべきものも多い。

９．まとめ

リアクション機構をOSの基本機能として提供することでリアルタイムAIシステムを効率的に実現できることを示した。
本システムはすでに実現され監視システムにおいて実用にも供している。ただし、この応用システムの構成はかなり単純で、ここに述べた機能を使いこなしているわけではない。
実際にリアクションの数が多くなってくると、相互の関係の記述が複雑になってくる。高度なリアルタイムAIを実現するには、この<複雑さ>が大きな課題である。

[参考文献]

[1] 渡辺、川村、日比野：新ELISのCPU-LSIの開発、信学全大、C-131、'89秋.
[2] 鈴木、家吉、菅原、杉村：新ELISシステム概念、信学全大、D-137、'89秋.
[3] O'Reilly, C. A. and Cromaty, A. S., : "'Fast' is not 'real-time': Designing effective real-time AI systems", Proc. of SPIE vol.548, 1985.
[4] Lesser, V. B., Pavlin, J. and Durfee, E., : "Approximate Processing in Real-Time Problem Solving",AI Magazine, spring 1988.
[5] Brooks, R. A. : "A Robust Layered Control System for a Mobile Robot", IEEE Journal of Robotics and Automation, RA-2(1), April 1986.
[6] Takeuchi, I., et al. : TAO Reference Manual, Aug 1987.
[7] 杉村、岸田：ELIS Common LISP のマルチプログラミング機能、39回情処全大、5Q-1、1989.
[8] Laffey, T. J., et al. : "Real-Time Knowledge-Based Systems" AI Magazine, sping 1988.