メディア処理基盤プロジェクト

オペレーティングシステムAGにおける適応的資源管理のためのエージェント構成手法

大久保 英嗣
理工学研究科

概要 AGは，CPUやメモリなどの計算機資源の適応的管理を目的とするオペレーティングシステムである．AGは，計算機資源を抽象化するマイクロカーネルと，マイクロカーネル上でこれらの抽象化された資源を管理するリフレクティブエージェントの集合から構成される．エージェントは，計算機資源の状態やアプリケーションの動作の変化に応じて，自律的なリフレクションを使用して適切な資源管理手法を選択する．本稿では，リフレクションを使用したエージェントの構成手法について述べる．

Media Infrastructure Project

A Construction Method of Agents for Adaptive Resource Management in AG Operating System

Eiji Okubo
Graduate School of Science and Engineering

Abstract: AG is an operating system that aims at adaptively managing computer resources such as CPU, memory and so on. AG consists of a micro kernel and set of agents. The micro kernel abstracts computer resources. Each agent runs on the micro kernel and manages the computer resources. By means of an autonomous reflection, each agent selects an appropriate resource management method according to changes in the state of the computer resource as well as actions of the application program. In this article, a construction method of agents by using reflection is described.

1. はじめに

Bluetoothや無線LANデバイスの普及に伴い、PDAやノートPCなどの携帯端末をネットワークに接続して利用する形態が一般的になってきている．さらに，USBやIEEE1394などの活線挿抜式デバイスの普及により，計算機資源の追加・削除を動的に行うことが可能になっている．これらのことから，今後，計算機資源の追加・削除や移動による通信方式の切替えなどが頻繁に発生し，システムの構成が流動的に変化すると考えられる．

このような複雑かつ動的に変化する環境を想定したオペレーティングシステム(以下，OSと記す)の研究として，OSの動的再構成に関する研究がある．これらの研究は，OSの構成を変更することで，環境の変化に適応することを目的としている．そのために，ユーザまたはプログラマが再構成のための命令を発行したり，カーネルモジュールを事前に用意することで対応している．従って，OSは，環境の変化に受動的に適応することしかできない．しかし，上述のような複雑な環境の場合，受動的な適応だけでなく，能動的な適応が必要になると考えられる．

以上の背景から，我々は，環境の変化に適応して自律的に資源管理を行うOSであるAGを開発している．AGは，資源の利用状況やアプリケ

ーションの動作状況といった環境の変化に適応して，OS自体があらかじめ用意した資源管理モジュールを自律的に切替えることで，適応的な資源管理を実現することを目的としている．このため，AGは，マイクロカーネルと，マイクロカーネル上で適応的な資源管理を行うエージェントと呼ばれるシステムサーバから構成される．エージェントは，他のエージェントやユーザプロセスからの要求を処理する即応型スレッドと，資源の利用状況を観測・解析しリフレクションのための計算を行う熟考型スレッドから構成される．熟考型スレッドは，資源の利用状況の変化に基づいて適切な資源管理手法を選択し，即応型スレッドの振舞いを変更することができる．AGでは，これらのスレッドにより，資源の状況の変化に応じた自律的かつ適応的な資源管理を可能としている．

我々は，これまで，AGのマイクロカーネルとエージェントの構成手法について研究を行ってきた．本研究では，エージェントによる適応性とその構成手法の有効性を実証するために，本構成手法を用いて，AGマイクロカーネル上にスケジューリングを行うエージェントを実装し評価を行った．その結果，本構成手法が，OSの即時性を損なうことなく，リフレクションによる適応的なスケジューリングの実現が可能であることを確認することができた．また，実際にスケジューラを動作させ，プロセスへの公平なCPU割当てが適応的に実現できることも確認した．

以下，本稿では，2章でAGの概要について述べ，3章でエージェントとその構成手法について述べる．最後に，4章でまとめと今後の検討項目について述べる．

2. AGの概要

AGの基本構成を図1に示す．AGは，特権レベルで動作するマイクロカーネルと，ユーザレベルで動作するプロセスから構成される．マイクロカーネルは，計算機資源を抽象化し，実行環境をプロセスに提供する．プロセスは，マイクロカーネル上で動作する．プロセスには，アプリケーションとしてのユーザプロセスと，システムサーバの役割を持つエージェントとがある．以下，本章では，AGの各構成要素の機能と役割について述べる．

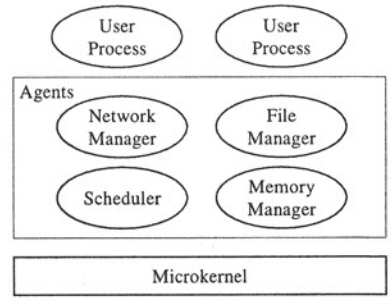

図1 AGの構成

2. 1 プロセス

AGでは，プロセス・スレッドモデルを採用している．プロセスは，実行実体であるスレッドを複数動作させることができる．プロセスとスレッドの生成・破棄は，動的に行うことが可能である．各プロセスには，独立した仮想アドレス空間が割り当てられるため，プロセスは互いに保護される．スレッドには，実時間制約があるリアルタイムスレッドと，実時間制約のない非リアルタイムスレッドとがある．

AGでは，プロセスをユーザプロセスとエージェントの2種類に分類している．ユーザプロセスは，ユーザが使用するプロセスである．エージェントは，OSとしてのサービスをユーザに提供するシステムサーバの役割を持つ．AGでは，動作環境や計算機構成に応じて，システムプログラマがエージェントを自由に作成・変更することを想定している．また，AGもしくはユーザにより，システムを構成するエージェントを必要に応じて動作・停止させることで，計算機構成の変化に柔軟に対応することができる．AGのエージェントには，図1に示す4つがある．各エージェントは，以下に示す機能を持つ．

- スケジューラ

  スレッドのスケジューリングを行う．リアルタイムスレッドと非リアルタイムスレッドの両方を扱うことが可能である．

- メモリマネジャ

  プロセスへ割り当てるメモリ量を決定する．また，主メモリ上の空きページ枠と使用ページ枠を管理する．
- ネットワークマネジャ

  プロセス間通信と通信デバイスの管理を行う．また，通信デバイスのデバイスドライバに相当する機能を持つ．
- ファイルマネジャ

ファイルと2次記憶装置の管理を行う．また，2次記憶装置のデバイスドライバに相当する機能を持つ．

エージェントは，ユーザレベルで動作するプロセスであるため特権命令を発効できない．従って，エージェントが行う処理は，特権命令を使用しない範囲に限られる．例えば，スケジューラではスケジューリングを行うが，ディスパッチはマイクロカーネルが行う．エージェントが特権命令を発効しなければならない場合は，システムコールを用いてマイクロカーネルに代行させる必要がある．

2．2 マイクロカーネル

マイクロカーネルは，計算機資源を抽象化し，プロセスの実行環境を提供する．また，マイクロカーネルは，特権レベルでのみ実行できる処理をエージェントとユーザプロセスに代わって行う．マイクロカーネルは，以下の構成要素からなる．

- ドライバと各種モジュール：ディスパッチャ，メモリマッパ，タイマハンドラ，キーボードドライバ，CGAドライバ
- 割込みハンドラ
- 各種データ構造：ページテーブル，ページリスト，プロセスリスト，スレッドリスト

ディスパッチャはスレッド切替えを行い，メモリマッパはページフォルト発生時に物理メモリの割当てを行う．ディスパッチャとメモリマッパは，それぞれプロセスリストとスレッドリスト，ページリストとページテーブルを参照して動作する．これらは，スケジューラとメモリマネジャも使用する．割込みハンドラは，すべての割込みを一括して受け付ける．割込みは，エージェントによって処理されるべきものであれば，割込みハンドラによってエージェントに渡される．また，割込みがマイクロカーネル内で処理されるべきものであれば，処理を担当するカーネルモジュールあるいはデバイスドライバに渡される．

マイクロカーネルは，割込みやタイムアウトなどのイベントが発生したときに呼び出すべきエージェントを知っている必要がある．AGのマイクロカーネルでは，エージェントが自分自身をマイクロカーネルに登録するために，registerThread()システムコールを提供している．本システムコールは，引数としてエージェントの機能を実現するスレッドの識別子を持つ．これにより，マイクロカーネルが適切なエージェントを呼び出すことが可能となる．

3．エージェント

エージェントは，従来のマイクロカーネル方式のOSにおけるシステムサーバと同様に，OSとしての機能を実現するプロセスである．AGには，図1に示すように，基本となる4つのエージェントがある．新しい機能を持つエージェントの追加や既存のエージェントの変更は，システムプログラマやユーザによって行われる．

エージェントとシステムサーバが異なるのは，エージェントに適応性がある点である．エージェントの適応性とは，エージェントが管理する計算機資源の利用状況と，その資源を利用する他プロセスの資源利用における特性に応じて，エージェントの内部処理を動的に変更することである．従来の資源管理の多くは，資源管理手法を大域的に用いている．しかし，プロセスによる計算機資源の利用特性は，プロセスごとに異なる．このことから，AGでは，プロセスごとに資源管理手法を替えることで，より高い適応性を実現している．

OSは，アプリケーションと異なる動作特性を持つ．従って，従来のリフレクション手法を，そのま

まエージェントに適用することは適切でないと考えられる．そこで，我々は，OSの動作特性に適したリフレクションを実現するエージェントの構成方式を考案し，AGに採用している．本章では，AGで用いるエージェントの構成方式に関して，OSの動作特性に基づくリフレクション手法とエージェントの環境の2つの視点から述べる．

3．1 リフレクション手法

リフレクションには，構成的リフレクション(structural reflection)と，振舞いに関するリフレクション(behavioral reflection)とがある．構成的リフレクションでは，クラスや構造体といったプログラムの構成を外部のメタプログラムが変更する．これにより，プログラムの機能を動的に追加・削除することが容易となる．しかし，その実現には，構成的リフレクション機能を提供する言語を用いる必要があり，汎用性に欠ける．一方，振舞いに関するリフレクションでは，計算機資源の状態やアプリケーションの動作状況に応じて，要求を処理するメソッドの切替えを行う．これにより，同一メソッドの呼出しでも実際に呼び出されるメソッドが変わるため，呼出しを行ったプログラムからはメソッドの振舞いが変化したように見える．

AGでは，計算機資源の利用状況とアプリケーションの動作に応じて，エージェントにおける処理を変更することで適応性を実現する．このことから，AGでは振舞いに関するリフレクションを採用した．OSは，ユーザからの要求や割込みなどをトリガとして動作する．システム性能を向上させるためには，OSにおける処理を可能な限り小さいオーバヘッドで行う必要がある．しかし，リフレクションは，処理が複雑で大きなオーバヘッドを伴う．従って，エージェントが割込みなどの処理を行うたびに，リフレクションに関する処理を行うことは，システム性能を劣化させることになる．そこで，AGでは，要求を処理するコンテキストとリフレクションに関わる処理のコンテキストを分割し，それぞれを異なるスレッドで実現している．本手法では，以下に挙げる2種類のスレッドを用いる．

- 即応型スレッド(reactive thread)：OSとしての処理を担当するスレッドである．ユーザプロセスや他のエージェントからの要求に対して，即時的に反応することが求められる．
- 熟考型スレッド(deliberative thread)：リフレクション処理を担当するスレッドである．

エージェントは，受け付けた要求を即応型スレッドにより処理する．熟考型スレッドは，エージェントの動作，資源の利用状況，各アプリケーションの状態を観測し，リフレクションが必要であれば適切な処理を行う．

ここで，先に述べたエージェントの振舞いは，即応型スレッドによって実現されているとみなすことができる．これは，計算機資源やアプリケーションから要求された処理を即応型スレッドが担当しているためである．即応型スレッドの動作を特徴づけているものは，メソッドとして実現される処理メカニズムと処理時に用いるパラメータである．エージェントは，使用するメソッドとパラメータの値を変更することで，リフレクションを実現する．使用するメソッドとパラメータの値の変更は，熟考型スレッドが行う．

メソッド呼出しを行う際に，通常のメソッド名を用いる方法では，熟考型スレッドにおける変更が不可能である．そこで，AGでは，メソッドセレクタを用いてスレッドを管理する．メソッドセレクタには，処理名とその処理を実際に行うメソッドへのポインタが対で格納されている(図2参照)．

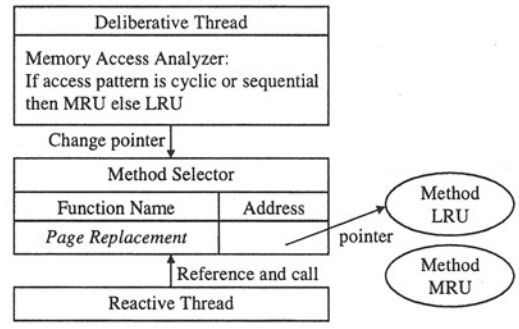

図2 メソッド管理

図2は，メモリマネジャにおけるページ置換え

アルゴリズムの切替えを表している．図2では，LRU(Least Recently Used)とMRU(Most Recently Used)の2つのメソッドがエージェント内に実装されている．ページフォルトが発生すると，メモリマネジャの即応型スレッドに制御が移行する．即応型スレッドは，置換えるページを選択するために，ページ置換えメソッドを呼び出す．このとき，即応型スレッドは，必要とする処理の名前(図2ではPage Replacement)をキーとしてメソッドセレクタを検索し，メソッドへのポインタを取得する．その後，そのポインタを介してメソッドを呼び出す．

熟考型スレッドは，メソッドセレクタにおけるポインタを変更する役割を持つ．熟考型スレッドは，アプリケーションのページアクセスパターンを解析し，現在と異なるメソッドを用いた方が効率が良くなると判断したときにポインタを変更する．図2の例では，メモリアクセスパターンがサイクリックもしくはシーケンシャルであればMRUを使用するように変更し，それ以外の場合はLRUを使用するように変更する．

パラメータによるリフレクションの場合も，メソッドセレクタと同様にリフレクション処理が行われる．即応型スレッドと熟考型スレッドは，パラメータを共有する．即応型スレッドは，パラメータに基づいて処理を行う．熟考型スレッドがパラメータを適宜変更することで，リフレクションが行われる．

3．2 エージェントの環境

リフレクションを行うためには，環境を観測してその状態を把握する必要がある．エージェントが認識することのできる環境が複雑なほど，正確で柔軟なリフレクションが可能になる．しかし，複雑な環境の観測には，高いコストが伴う．従って，適切な環境の定義が必要である．

AGでは，エージェントの環境とは以下の3つの要素から構成されるものと定義している．
(1)要求を発行するユーザプロセス及びエージェント
(2)管理対象となる計算機資源
(3)制御対象であるその他の資源及びデータ
この定義のうち，(2)と(3)に関しては，あらかじめシステムプログラマによって，各エージェントに与えられるものとする．(1)に関しては，エージェントが要求を受け取った時点で，発行元のプロセスを環境に組み込むことで対応できる．従って，この定義に従うと，エージェントは環境を簡単に見つけることが可能となる．

3．3 環境の観測

前節の環境要素(2)と(3)は，エージェントが直接管理するため，観測を行うことが容易である．しかし，(1)は，エージェントが独立したプロセスであるため，それらの状態を得ることが困難である．このような環境要素に対する観測手法として，ポーリング方式と通知方式が考えられる．ポーリング方式では，エージェントが観測対象のプロセスに対して，一定期間毎に環境の状態を能動的に取得する．この方式では，観測周期を変えることで，観測結果の精度を調整することができる．しかし，観測周期が短いと，無駄な観測によるオーバヘッドが増加する．逆に，観測周期が長いと，重要な変化を見逃してしまう可能性が高くなる．

通知方式では，ポーリング方式とは逆に，環境が能動的にその変化を観測者に通知する．この方式では，変化が起きたときのみ通知することで無駄な観測が生じない．また，変化を見逃すことがなくなるという利点もある．しかし，この方式には，以下に示す問題点がある．

- 変化を通知するためのプロトコルをすべてのプロセスに実装する必要がある．
- プロセスは，自身がどのエージェントの環境に組み込まれているか意識しなければならない．
- プロセスは，内部で起こりうる変化を自身で検知する必要がある．
- プロセスは，どの変化をどのエージェントに通知するかを知っている必要がある．

これらの問題は，あらかじめプロセスに必要な

情報や手段を与えておくことで回避することが可能である．しかし，AGでは，動的にプロセスやスレッドが生成・削除されるため，このような回避方法をとることが困難であると考えられる．また，この方式では，エージェントの構成が環境に依存するため，柔軟なシステム構成を実現することが困難になる．

そこで，AGでは，ポーリング方式および通知方式の問題を解決するために，要求に基づく観測手法を用いる．本手法は，要求の変化が環境の変化であるという考え方に基づく．すなわち，環境に何らかの変化が起きると，その変化は要求に現れるという考え方である．

例えば，図3で用いた例を考える．ここで，メモリマネジャが受け取るページフォルトが，メモリ割当ての要求であるとする．プロセスがランダムアクセスを行っている間は，ページフォルトが発生するアドレスもランダムである．プロセスがシーケンシャルアクセスを行っている場合，ページフォルトが発生するアドレスは，連続したアドレスになる．従って，ページフォルトが発生したときのアドレスの変化から，プロセスのメモリアクセスパターンが変化しているか否かを判断することが可能である．同様に，プロセスからのメモリ割当て要求が頻繁であったり，要求するメモリ量に大きな変化があれば，そのプロセスが何らかの変化をしたと見なすことができ，その変化に対する対応をとることが可能になる．以上から，本手法の利点をまとめると以下のようになる．

- 要求を発行したプロセスのみを環境に組み込むため，プロセスの追加・削除という環境の変化に対応が容易である．
- プロセスは，エージェントを意識する必要がない．
- 観測のためのプロトコルやインタフェースを必要としない．

3．4　エージェントの構成

以上の考察から，AGでは，プロセスであるエージェントを図3のように構成している．すなわち，図3に示すように，エージェント内部には，即応型スレッド，熟考型スレッド，メソッドセレクタとメソッド，共有パラメータ，要求内容のログがある．

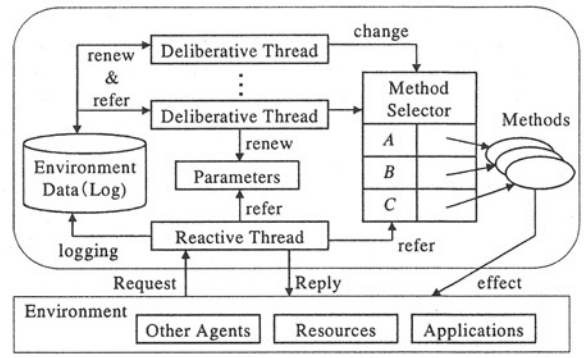

図3 エージェントの構成

即応型スレッドは，他のエージェントやユーザプロセスから要求を受け取り，メソッドセレクタと共有パラメータを用いて要求を処理する．共有パラメータは，エージェント内で熟考型スレッドと即応型スレッドにより共有されるパラメータであり，これらは要求の処理時に利用される．即応型スレッドは，要求を処理するときに，必要であればその内容をログとして残す．このログは，熟考型スレッドが環境の変化が起きたか否かを判断するために用いられる．従って，すべての要求をログとして保存するのではなく，エージェントが必要とする部分のみを保存する．従って，ログの内容，保存方法，保存期間は，エージェント毎に異なる．

熟考型スレッドは，起動されると環境内容のログを解析する．その結果，エージェントが対応することができる環境の変化が見つかれば，熟考型スレッドは，パラメータまたはメソッドセレクタを更新することでリフレクションを行う．また，熟考型スレッドは，必要に応じて複数用意することが可能である．例えば，メモリマネジャにおいて，メモリアクセスパターンの解析とプロセスに割り当てるメモリサイズや領域の調整を異なる熟考型スレッドを用いて行うことが考えられる．さらに，熟考型スレッドを動的に起動・停止することで，環境の変化への対応をより柔軟に行うことも可能となる．

4. おわりに

本稿では，流動的に変化する分散環境において，リフレクションを用いて適応的な資源管理を実現するOSであるAGについて述べた．特に，その構成要素であるエージェントの構成手法について述べた．提案手法では，エージェントが行う処理を即応型スレッドと熟考型スレッドの2つのスレッドで分担し，熟考型スレッドが即応型スレッドを制御する形でリフレクションを実現する．今後は，AGで必要とするメモリマネジャなどの未実装のエージェントを設計・実装していく予定である．

参考文献

[1]元濱努，大久保英嗣 他: エージェント指向オペレーティングシステムAGにおける通信デバイス切替え手法，2004年電子情報通信学会通信ソサイエティ大会論文集，B-6-78，pp. 78-79 (2004).

[2]瀧本栄二，大久保英嗣 他: AGマイクロカーネルにおけるリフレクティブエージェントの構成手法とスケジューラへの適用,コンピュータシステム・シンポジウム論文集，pp. 129-138 (2004).