

自律分散システムに関する一考察：大規模システムにおける ネットワークサービス提供のための基盤モデル

A study on an autonomous distributed system:
A model for network services in a very large distributed system

中島 達夫 所 真理雄
Tatsuo Nakajima Mario Tokoro

慶応義塾大学理工学部
Department of Electrical Engineering
Keio University

概要

インタネットワークの発展により大規模な分散システムの重要性が高まっている。大規模分散システムは、従来の分散システムと異なり多種多様なコンポーネント/メディアを多くの管理ドメインで共有しなければならない。そのため、各コンポーネントの自律性が非常に問題になってくる。現在、多くの分散システムはオブジェクトモデルをベースとしているが、ここでのオブジェクトは自律性への考察が不足していると思われる。本論文では、自律性を考慮したオブジェクトモデルである、自律オブジェクトモデルを提案する。このモデルでは、オブジェクトの自律性を高めるためポリシをオブジェクトから明確に分離する。また、オブジェクトをグループ化しそれらの間で共有されるポリシをメタポリシとして実現することによりオブジェクト間の協調動作を可能にする。

A very large distributed system becomes more important because of an evolution of internetworks. In a very large distributed system, various components and media must be shared among many administrative domains. Then, an autonomy of each component must become a material for discussion. In the presence, many distributed systems are based on an object model. But, there is a room for much argument in connection with an autonomy of an object. In this paper, we propose an autonomous object model in which a policy is clearly separated from an object and, a meta policy which is shared with an object group make an cooperation among members in a group possible.

1 はじめに

高速な広域ネットワークの発達によりインタネットワークの有効性が高まってきている。そこでは、ネットワークは、遠隔ログイン、ファイル転送、メールなどの従来のサービスだけではなく、オンラインデータベースやマルチメディア通信などにより、現在のメディアを統合するメタメディアへと発展する可能性を持っている。しかし、これを実現するためには、ネットワーク上の各コンポーネントを有機的に統合できるような大規模な分散システムを構築する必要がある。ここでは、各コンポーネントは様々なサービス/メディアをサポートしなければならないため、それらの自律性が非常に問題となってくる。しかし、現在多くの分散システムがベースとしているオブジェクトモデルは自律性の検討が不足していると考えられる。本論文では、従来のオブジェクトモデルの問題点を検討し自律性を考慮した自律

オブジェクトモデルを提案する。このモデルでは、次の3つの点で従来のオブジェクトモデルと異なっている。1つ目は自律的な通信のサポートである。従来のオブジェクトモデルでは、メッセージは受け取られると必ず実行されるとされていた。しかし、自律システムでは状況に応じてそのメッセージの実行を制御できなければならない。2つ目は、オブジェクトのグルーピングである。ここでは、オブジェクトをグループ化することによりそれを1つのオブジェクトとして扱えることを可能とする。これにより、メッセージを送るオブジェクトを特定しなくてすむようになる。3つ目は、オブジェクトを制御するためのポリシを明確にオブジェクトから分離することである。また、グループ間で共有するポリシをメタポリシとして定義できるようにすることにより、グループのメンバー間での協調動作を可能とする。これにより、オブジェクトの自律性を維持しつつ協調動作することを可能とする。

自律オブジェクトはDCST[3]でのセクレタリオブジェクトの拡張と考えられるので、まず、第2節では、DCSTと、そのセクレタリオブジェクトについて検討する。そして、第3節では、自律オブジェクトモデルについて解説する。

2 DCSTとセクレタリオブジェクト

DCSTはCST[2]をインタバーソナル環境に適するよう拡張した分散システム記述用語である。この言語は、次の3つの特徴を持っている。

- Proxyを用いたリモートオブジェクトアクセス
- 共有を意識したオブジェクトネーミング
- メソッド間の関係を意識した同期

分散オブジェクト指向言語ではリモートサイトに存在するオブジェクトをローカルサイトに存在するオブジェクトとを同じ形で、かつ位置を意識せずアクセスできなければいけない。そのため、いくつかの方法が存在するがDCSTで採用しているProxyは現在のCSTのセマンティクスを変更せずリモートオブジェクトのアクセスを可能とする。また、リモートアクセスをProxyオブジェクトというオブジェクトで実現するため、ユーザはこれを容易に変更できるので非常に柔軟性が高くなっている。

CSTではパーソナル環境で用いるのを前提としているため、グローバルオブジェクトのネーミングはフラットとなっていた。しかし、これをそのままインタバーソナル環境で用いるとユーザ間でオブジェクトの名前がコンフリクトする。そのため、DCSTでは複数のネームスペースを持てるようにする。そして、オブジェクトの共有が簡単にできるよう複数のネームスペースを融合して1つのネームスペースにすることができるようになる。これにより、各ユーザが各自のネームスペースを持つことができ、かつ、それらのオブジェクト間でのオブジェクトの共有を可能としている。

DCSTでの同期メカニズムは、条件同期としては各メソッドの実行する前にチェックされるガードと排他同期としては各メソッド間の排他関係を用いるメソッドリレーションを用いている。これでは、同期情報をメソッドの内部に持たないため同期情報をメソッド定義と別に定義できる。そのため、メソッドを変更せず同期情報だけを変更することが可能となる。しかし、同期メカニズムがオブジェクトのメソッドと分離されて定義されるため、その実現は同期のためのメソッドとオブジェクトが持つメソッドが分離されていなければいけない。そのため、DCSTではセクレタリオブジェクトを導入している。セクレタリオブジェクトはCSTではじめて導入されたものだが、CSTではシステムオブジェクトとして実現されていたため機能が画一的であった。つまり、これでは全てのメソッドが排他的に実行される。しかし、分散システムでは共有オブジェクトをそのよ

うにシングルスレッドで実現するとアクセスのディレイが大きくなりシステム全体のパフォーマンスが悪くなる。そのため、DCSTではオブジェクト内でのメソッドの実行をマルチスレッドで行なえるようにして、メソッド間での排他関係を定義することにより、各オブジェクト毎に適した同期ポリシーを定義できるようにする。これはセクレタリオブジェクトをユーザ定義可能とすることにより実現される。セクレタリオブジェクトはオブジェクトのメソッドの実行状態を監視して、どのメソッドが実行可能かをチェックすることにより同期を実現する。しかし、DCSTでのセクレタリオブジェクトはメソッドの同期のみしか制御できない。また、DCSTでのオブジェクトは自律性を持っていない。そのため、自律分散環境に適合させるためには拡張が必要がある。

3 自律オブジェクトの導入

大規模な分散システムを考える場合さまざまな視点から検討を加える必要がある。しかし、もっと重要な問題は、それらを統括的に考えるためのベースとなるモデルである。そのようなモデルの1つとしてオブジェクトモデルが存在するが、大規模な分散システムに適したオブジェクトモデルに関する検討はまだほとんど行なわれていない。我々は、オブジェクトの自律性と異なったコンテキストで作られたオブジェクト間での協調への考察が大規模なシステムの基盤モデル構築への第一歩になると考えている。ここでは、現在のオブジェクトモデルの問題点に関して述べた後、これら問題点を考慮したオブジェクトモデルである自律オブジェクトモデルを提案する。

3.1 オブジェクトモデルへの考察

オブジェクトモデルでは各オブジェクトはある入力に対してある出力を返すブラックボックスであると考えられる。このブラックボックスは機能と構造の両方を持っていて外部からは機能のみがわかっている。そこでは、オブジェクトは階層状に構造化され、上位のオブジェクトから見ると、下位のオブジェクトは機能のみがわかっているブラックボックスであると考えられる。ここでの計算は通常、次の4つの性質を持っている。

- 階層型コントロール
- Request/Reply型メッセージ転送
- 相手を選定した通信
- オブジェクトの動作が静的に決定される

1番目の性質の階層型コントロールは、上位のオブジェクトが下位のオブジェクトに対する制御権を持っていることを意味する。つまり、上位のオブジェクトが下位のオブジェクトにメッセージを送ること

により計算は進む。この階層型コントロールの問題点は、コントロールが集中型で行なわれることである。これは、一般に信頼性や、効率の面から好ましくない。また、全体の動作が1つのオブジェクトにより管理されるため動作が一般に画一的になりがちになるため柔軟性が乏しくなる。1人の人が全体を構築する場合はこれでもよいが、多くの人が作ったプログラムを同時に協調させて動かす場合には問題がある。従って、自律的なオブジェクトはこのような階層型コントロールではコントロールできない。そのため、非階層型のコントロールを用いる必要がある。非階層モデルは現在多く分けて2つに分類することができる。1番目はクライアント/サーバモデル、2番目は会話型モデルである。クライアント/サーバモデルでは複数のオブジェクトが1つのオブジェクトと通信する。会話型モデルでは複数のオブジェクトが複数のオブジェクトと通信する。実際、分散システムでの共有サービスはクライアント/サーバモデルが必要であるし、複数ユーザでのリアルタイムメッセージ通信などでは会話型モデルを必要とする。そのため、非階層型コントロールのサポートは自律的オブジェクトモデルには必要不可欠である。これは、すべてのオブジェクトが並行動作できるようにすることで実現できる。

2番目の性質であるRequest/Reply型メッセージ転送はオブジェクト間の通信モデルとして最も多く用いられている。これは、センダオブジェクトがターゲットオブジェクトに要求を出すと、ターゲットオブジェクトはその要求に答えて返答を返すというものである。Request/Replyメッセージ転送には2つの問題がある。1つ目の問題はセンダ/レシーバ2つのオブジェクト間で常に同期がとられることである。つまり、他のオブジェクトに対して依頼を出す場合、常に返答を待たなければいけない。自律オブジェクトでは必ずしもリプライが返ってくるとは限らないのでこのような通信は好ましくない。2つ目の問題はターゲットオブジェクトが常にメッセージを同じ形で受け取る ことである。つまり、オブジェクトはいつもメッセージを受け取るとすぐ処理する。オブジェクトが自律的に実行されるためには、メッセージを受け取って実行するかどうかは各オブジェクトの自由意思としなければいけない。

3番目の性質であるメッセージを送る相手を常に特定しなければいけないということは、次の2つの問題を持っている。1つ目の問題は自分の要求をどのオブジェクトが処理できるかわからない時生じる。このようなとき、この要求を処理できそうな複数のオブジェクトに同時に要求を送り返答を待つことができればいけない。2つ目の問題は自分の状態をほかの複数のオブジェクトに同時に送りたい時である。このとき、自分の状態を必要とするオブジェクトを特定することができないので、同時に複数のオブジェクトに送れることが好ましい。このためのメカニズムとしてオブジェクトのグルーピングとマルチキャストメッセージが必要である。自律システ

ムでは相手との関係が疎に結び付いているため。常にメッセージを送りたい相手を特定できるとは限らないのでこのようなシステムは必要不可欠である。

4番目のオブジェクトの動作が静的に決定されていることである。そのため、予想外のメッセージを処理することができない。別々に作られたオブジェクトが協調するためには同じコンテキストを持たなければいけない。つまり、センダは相手に送ったメッセージがどのように処理されるか知らなければいけない。しかし、自律的な環境では勝手にオブジェクトが作られるためこれを前提とすることができない。従って、オブジェクトを協調させる場合はメッセージの処理、要求に対する動作、相手の状態の通知の受け取りなどは全てのオブジェクトで同じように扱えなければいけない。そのため、オブジェクトのポリシのコントロールをメタレベルで定義できるようにし、協調動作が必要な時は同じポリシを各メンバーのオブジェクトに与えることにより、それらを同一のコンテキストで用いることを可能とする。

本論文で提案する自律オブジェクトモデルはこれらの要求をすべて満足している。

3.2 自律オブジェクトモデル

自律オブジェクトモデルは各オブジェクトの自律性を保つとともに、異なったコンテキストで用いる場合でもポリシをコントロールすることによりそのオブジェクトが使われるコンテキストで動作することを可能とする。この節では、自律オブジェクトモデルについてオブジェクトの構造、コミュニケーション、オブジェクトグルーピング、メタレベルの利用について述べる。そして、最後にいくつかの具体例について解説する。

3.2.1 オブジェクトの構造

自律オブジェクトモデルは各オブジェクトが並行に実行される並行オブジェクトモデルをベースとしている。それにより、実行のコントロールとして階層型コントロールだけではなく非階層型のコントロールもサポートできる。自律オブジェクトは次の3つのオブジェクトより構成される。1つ目はベースオブジェクト、2つ目はセクレタリオブジェクト、3つ目はアクティビティオブジェクトである。ベースオブジェクトは自律オブジェクトのバッキングな部分である。ベースオブジェクトのステートの変更はセクレタリオブジェクトに伝えられる。アクティビティオブジェクトはセクレタリオブジェクトがメッセージを受け取る毎に作られる。これは、実際のメソッドを実行するために必要なコンテキストの管理を行なう。セクレタリオブジェクトはメソッドの実行、メッセージキューの管理、ステートの変化の管理をおこなう。

セクレタリオブジェクトは次の7つのメソッドを持っている。

1. stateLookup: 'stateName'
2. stateChange: 'stateName'
3. stateChange: 'stateName' value: 'object'
4. incomingMessage: 'message'
5. processMessage: 'message'
6. startMethod: 'message'
7. endMethod: 'message'

1 番目のメソッドはセクレタリオブジェクトからオブジェクト状態を参照したいとき用いる。これは、例えば条件同期を実現する時オブジェクトの状態を見たい時に用いる。2 番目のメソッドはオブジェクトで状態が変更された時呼ばれるセクレタリオブジェクトで定義される。これを定義することによりセクレタリオブジェクトはオブジェクトの状態の変化を知ることが可能となる。3 番目のメソッドはセクレタリオブジェクトがオブジェクトの状態を変更したい時に用いる。これは、セクレタリオブジェクトが他のセクレタリオブジェクトからの要求を受けそのオブジェクトの状態を変更したい時に用いる。4 番目のメソッドは新しいメッセージを受け取った時に実行されるメソッドである。これは、メッセージキューのスケジューリングのために用いる。このメソッドは内部で processMessage メソッドを呼び出す。5 番目のメソッドはメソッドを実行する前の前処理をおこなうためのメソッドである。これにより、メソッドを実行する前に必要な処理を行なう。6 番目のメソッドは新しいアクティビティオブジェクトを作って実際にメソッドを実行させるメソッドである。7 番目のメソッドはアクティビティオブジェクトがメソッドの実行を終了した時に呼ばれるメソッドである。これを受け取ることでセクレタリオブジェクトは他のメソッドの実行を行なうことを可能とする。

セクレタリオブジェクトではこれらのメソッドを定義することによりそのオブジェクトを管理するためのポリシーを決定することができる。

また、これらのメソッドはメタセクレタリオブジェクトから与えることも可能である。メタセクレタリオブジェクトはセクレタリオブジェクトのポリシーを決定する。これは、複数のオブジェクトで共有することが可能である。このオブジェクトはセクレタリオブジェクトのポリシーを入れ換えることができ、ポリシーの変更によりセクレタリオブジェクトが他のセクレタリオブジェクトと通信したり、各オブジェクトのセクレタリオブジェクトのローカルなポリシーをリプレースすることにより、複数のオブジェクトが共通なポリシーを持つことを可能とする。詳しくは 3.2.4 で解説する。

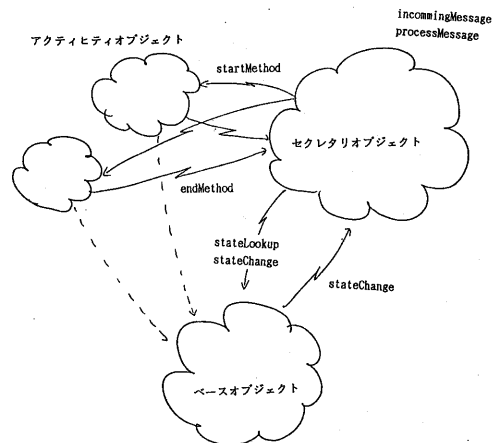


図1：自律オブジェクト

自律オブジェクトはこれら3つのオブジェクト：ベースオブジェクト、セクレタリオブジェクト、アクティビティオブジェクトのコンポジットオブジェクトとして実現される。

3.2.2 コミュニケーション

自律オブジェクトにおける通信は他の並行オブジェクトモデル同様、同期メッセージ転送と非同期メッセージ転送の2種類を提供している。非同期通信は他のオブジェクトへの非同期な処理の依頼を可能とする。

しかし、オブジェクトを自律的に実行するためには2つの問題を解決する必要がある。1 番目の問題は相手を選定しないメッセージ送信である。このため、まず、オブジェクトをグループのためのメカニズムを提供する。そして、新しい通信の手段としてグループの全メンバーへのマルチキャスト通信を提供する。これにより、グループの名前を指定するだけでそのグループの全メンバーにメッセージを送ることが可能となる。そのため、センドからは相手がオブジェクトかオブジェクトグループかは透過となる。オブジェクトのグルーピングに関しては 3.2.3 で詳しく解説する。

2 番目の問題はメッセージ処理の自律性である。これを実現するため自律オブジェクトモデルでは Action/Reaction に基づく通信モデルを提案する。Action/Reaction モデルではメッセージはすべて Future メッセージとして送られる。Action メッセージが受け取られると、オブジェクトはそれを実行するかどうかが決定する。もし実行可能ならそれに対応するメソッドを実行した後 Reaction メッセージを返す。センドオブジェクトが Reaction メッセージを受け取るか否かはセンドオブジェクトの自由である。また、レシーバオブジェクトが Reaction メッセージを返さないことがあるかもしれないし、オブジェクトグルー

ブへ Action メッセージを送った時は複数のオブジェクトから Reaction メッセージが返ってくるかも知れない。そのため、センダオブジェクトはその Action により生じた Reaction メッセージに含まれるリブライオブジェクトを Future オブジェクトに蓄える。そして、必要なリブライオブジェクトを Future オブジェクトから取り出す。しかし、Reaction メッセージは返ってこないときもあるし、適当な数の Reaction メッセージを待ちたいときもある。そのため、Future オブジェクトへのアクセスは必要な数の Reaction メッセージが返ってくるまでブロックされる。これは、Future オブジェクトにタイムアウト時間とブロックを解除するための条件式を送ることにより実現される。これがタイムアウトによる解除の時は Future オブジェクトは incomplete の状態でリターンし、条件式が満足した時は complete 状態でリターンする。そして、センダはブロックが解除された時それが complete か incomplete か調べて次に動作を決定する。

マルチキャストの導入及び、Action/Reaction モデルの導入によりオブジェクトの自律性を高めることが可能となる。

3.2.3 オブジェクトのグルーピング

オブジェクトのグルーピングは複数のオブジェクトを1つのオブジェクトとして扱うための手段として非常に有効である。これにより、複数のオブジェクトを一括管理したり、グループの不定メンバーにメッセージを送ったりすることが可能となる。グループはほとんどの場合 (Part-Whole Relation とメタセクレタリオブジェクトによる Object Cooperation を除いて) エクスプリシットに作られる。つまり、グループに対しての join メソッド、leave メソッドによりグループはダイナミックに形成される。

オブジェクトのグルーピングとして次の5つの場合が考えられる。

1. Part-Whole Relation
2. Function Replication
3. Structure Relication
4. Information Grouping
5. Object Cooperation

1 番目の Part-Whole Relation では Part オブジェクトは Whole オブジェクトの部品であると考えられる。Part オブジェクトはグループ化され Whole オブジェクトにより管理される。Whole オブジェクトは実体を持たず Part オブジェクト間のコンストレーションをメタセクレタリオブジェクトを用いて各 Part オブジェクトの状態の変化を他のオブジェクトにセクレタリオブジェクト間の通信を用いて実現することにより仮想的に存在させることも可能である。

2 番目の Function Replication はサーバグループ

を意味する。サーバグループでは同じ機能のサーバをグループ化し、そのサーバグループへのメッセージは全メンバーにマルチキャストされる。例えば、プロセスサーバを考えた場合、各サーバは自分の負荷を他のサーバにマルチキャストする。そして、プロセスサーバがプロセスクリエーションメッセージを受け取った時、マルチキャストによって受け取った各サーバの状態から一番軽い負荷を持った計算機に処理を依頼する。このように同じ機能を持つサーバ間でロードバランシングを行なう場合サーバグループは非常に有効であると考えられる。

3 番目の Structure Replication は信頼性を向上させるためオブジェクト自体を多重化することを意味する。グループへ送られたメッセージはメンバーの全オブジェクトに同時に送られる。オブジェクト間の一貫性はセクレタリオブジェクトにより管理される。つまり、全 Replication は共通のポリシーを持ったセクレタリオブジェクトを持っていて、この共通ポリシーにより一貫性を保つ。

4 番目の Information Grouping では関係のある情報を1つのものとしてグループ化する。例えば、ある家族の個人情報などは1つのものとして管理される。これは、従来の Unix などで用いられているディレクトリと同様の機能を持つものである。

5 番目の Cooperation は異なった機能/構造を持ったオブジェクトが全体で1つの仕事を実行するものである。これは、次の3つの場合が存在する。

- 全体を管理するグループオブジェクトが存在し階層型に管理される場合
- 全体を分散管理された1つのグループとし、その間でステートの交換や要求のマルチキャストを用いて協調する場合
- 共通のメタセクレタリオブジェクトを持たせ、ポリシー間で通信させることにより協調を行なう場合

1 番目ではグループを管理するオブジェクトがメッセージをすべて受け取りそれをすべてのメンバーに送る。これは、すべてのメッセージがグループオブジェクトを介して行なわれるためこれがボトルネックになる可能性がある。また、コントロールが階層化されるため自律分散システムには適さない。2 番目では集中管理するオブジェクトは存在せずメッセージはマルチキャストを用いて全体に転送される。集中管理が必要ない代わりシステムがマルチキャストをサポートする必要がある。3 番目では協調はセクレタリオブジェクトでインプリシットに行なわれる。この方法は、共通ポリシーを提供するメタセクレタリオブジェクトの設計が困難であると考えられる。

自律オブジェクトでは自律性の面から2番目と3番目の方法を提供する。

3.2.4 メタレベルの利用

この節では、セクレタリオブジェクトとメタセクレタリオブジェクトの関係について説明する。セクレタリオブジェクトはオブジェクトの実行のポリシーを決定する。しかし、そのポリシーは自律的に決定されるためオブジェクトが協調して動く時は共通のポリシーが必要となる。また、各オブジェクトの協調のためセクレタリの持つポリシー同士が通信し合うことも必要である。そのような共通のポリシーを提供するメカニズムが、メタセクレタリオブジェクトである。メタセクレタリオブジェクトは協調するオブジェクト間で共有される。あるオブジェクトは複数の協調グループに属することもあるので、1つのオブジェクトが複数のメタセクレタリオブジェクトを持つこともある。その場合、必要に応じてメタセクレタリを選びダイナミックにポリシーをリブレースしていくことも必要である。ここでは、メタセクレタリオブジェクトの性質を次の7つの項目について議論する。

1. Default Action
2. Dynamic Policy Replacement
3. セクレタリオブジェクトのグルーピング
4. メタセクレタリオブジェクトの共有の方法
5. ポリシの競合
6. ポリシの共有
7. メタセクレタリオブジェクトによるコンストレイントの処理

1 番目の Default Action は、本来そのオブジェクトで受け取れないメッセージを処理するためにメタセクレタリオブジェクトで定義されたメソッドである。これは、グループのメンバ間で協調動作させる時、オブジェクトがグループとして構成されていればマルチキャストにより適当なオブジェクトがそれを受け取れる。しかし、エクスプリシットにそれが使えないのでこの Default Action によりあるオブジェクトがそれを受け取って適当なオブジェクトに転送する。また、オブジェクトを拡張やカスタマイズしたい時にも用いることが可能である。

2 番目の Dynamic Policy Replacement はセクレタリオブジェクトの持つメソッドをメタセクレタリオブジェクトが取り換えることである。これには、2つの方法が存在し、1番目はセクレタリオブジェクトが持つ別のポリシーに変更する場合で、2番目はメタセクレタリオブジェクトの持つポリシーを継承する場合である。1番目はオブジェクトスベシフィックなトレーニングなどの場合に用いる。2番目はセクレタリオブジェクトが持つメソッドをメタセクレタリオブジェクトと取り換える場合である。これは、オブジェクトが持つポリシーを自分のコンテキストで

用いたい場合に用いる。つまり、メッセージの扱いや、DefaultAction を自分向けにカスタマイズしたい時や拡張したいときに用いる。また、オブジェクト間で協調動作させるため、セクレタリオブジェクトをグルーピングしてインプリシットにグループを構成したいときもこれを用いる。

3 番目のセクレタリオブジェクトのグルーピングは協調動作させたいセクレタリオブジェクトを1つのグループとして構成することである。これは、オブジェクトグループによるエクスプリシットなグルーピングではなくインプリシットなグルーピングである。この利点は、グループへの Join/Leave をオブジェクトがエクスプリシットに定義する必要がないことである。つまり、メタセクレタリオブジェクトを用いてグループを形成させる。つまり、これを実現するため、協調動作させたいオブジェクト内で1つのメタセクレタリオブジェクトを共有させる。共有するメタセクレタリオブジェクトは processMessage メソッドと stateChange メソッドの2つをリブレースする。これにより、あるオブジェクトへの要求をメンバ全体に伝達したり、自分のステートの変化をメンバ全員に伝達することが可能となる。そのため、本来このオブジェクトでは定義されていないメッセージも受け取れるようにする必要がある。そのために Default Action を用いる。また、ステートの変化やメッセージの到着を他のセクレタリオブジェクトに知らせる時、それを受け取るためのメソッドも必要である。そのメソッドもメタセクレタリオブジェクトで定義される。

4 番目のメタセクレタリオブジェクトの共有によりオブジェクト間での協調動作を可能にする。ここでは、メタセクレタリオブジェクトが実際どのようにして共有されるかについて述べる。はじめオブジェクトはメタセクレタリオブジェクトを持っていない。これは、ユーザによりエクスプリシットにセクレタリオブジェクトに付加される。そして、メタセクレタリオブジェクトの共有は、メタセクレタリオブジェクトを持つオブジェクトにメッセージを送ることによりおこなわれる。しかし、そのメタセクレタリオブジェクトがメッセージ転送による共有が不可と定義されている場合は共有されない。そして、メタセクレタリオブジェクトのデタッチはメタセクレタリオブジェクトから継承されたポリシーでエクスプリシットに行なわれる。あるオブジェクトが複数のメタセクレタリオブジェクトを持つオブジェクトにメッセージを送るとそのオブジェクトは複数のメタセクレタリオブジェクトを持つことになる。それらが持つポリシーのうちどれを選ぶかについて次に議論する。

5 番目のポリシーの競合は、次の2つの場合に生じる。1番目はセクレタリオブジェクトが持つポリシーとの競合、2番目は複数のメタセクレタリオブジェクトを持つことにより生じるポリシーの競合である。1番目のポリシーの競合はオブジェクトの自律性を保つためセクレタリオブジェクトの責任である。つま

り、セクレタリオブジェクトでは PolicyReplacement を許すか、ローカルのポリシーとメタセクレタリオブジェクトのポリシーの両方を実行するかについて記述してある。2 番目のポリシーの競合では、メタセクレタリオブジェクトのポリシーがグループを意味するため、複数のメタセクレタリオブジェクトを持つことは複数のグループに属することを意味する。そのため、それらのポリシーは全部実行されなければいけない。つまり、複数のメタセクレタリオブジェクトを持つことは複数のポリシーを持つことになる。そして、このポリシーが実行される時はそのすべてのポリシーが実行される。しかし、この実行はセクレタリオブジェクトによっても制御可能で、これにより特定のメタセクレタリオブジェクトのポリシーを実行することも可能である。

6 番目のポリシーの共有はメタセクレタリオブジェクトを共有することによりおこなわれる。ここでは、特に、共有ポリシーの構造について議論する。ポリシーは次の 2 つが存在する。1 つ目はローカルポリシー、2 つ目は協調ポリシーである。ローカルポリシーは他と通信することがないポリシーである。これは、オブジェクトの動作を自分用にカスタマイズするため、オブジェクトを変更せずに拡張するために用いる。協調ポリシーはオブジェクトをインプリメントに協調させたい場合に用いる。つまり、あるオブジェクトのポリシーが他のオブジェクトのポリシーと協調させたい場合に用いる。共通のメタセクレタリオブジェクトを持っている時、ポリシーの継承によりセクレタリオブジェクトはグループを形成する。その v のため、ポリシーは他のオブジェクトのセクレタリオブジェクトに対してマルチキャストメッセージを送ることが出来る。これを用いて、自分の状態を送ったり、他のセクレタリオブジェクトに対して要求を送ったりすることが可能である。これによりオブジェクトの協調が実現される。

7 番目はメタポリシーを用いたコンストレインツの実現である。2 つのオブジェクト間にコンストレインツをかけたい場合 stateChange メソッドをメタセクレタリで定義する。このメタセクレタリオブジェクトはメッセージ伝搬により他のオブジェクトからの共有の対象とならない。そして、この 2 つの間でセクレタリオブジェクトはグループを形成する。このオブジェクトのセクレタリオブジェクトが持つ stateChange メソッドが起動された時、このメソッドはそれをコンストレインツがかけられた他のオブジェクトのセクレタリオブジェクトに伝搬する。そのセクレタリオブジェクトはそれにより自分のオブジェクトのステートを変更する。

このように、メタセクレタリオブジェクトの導入によりオブジェクトの自律性、及び、柔軟性を高めることが可能となる。

3.2.5 メタポリシーの具体例

この節ではメタポリシーの具体例としてラウンドロビンスケジューラ (RRS) とデッドラインスケジュー

ラ (DLS) が同一計算機内で協調してそれぞれのタスクを実行する場合を考える。通常は RRS がすべてのタスクをラウンドロビンポリシーでスケジューリングしている。そこに、タイムクリティカルなタスクがスタートされた場合を考える。これは、DLS により管理される。DLS はこのタスクが必要な時間を見積り全体の CPU の何パーセント消費するかを計算する。そして、DLS のセクレタリオブジェクトは RRS のセクレタリオブジェクトにこれを知らせる。そうすると RRS のセクレタリは自分のタスクを見て特に対話性を必要とするタスクを調べてどの程度の CPU を必要とするかを計算する。そして、DLS の要求が受け入れられるかどうかを見てその結果を DLS のセクレタリオブジェクトに送る。それが、可能ならこのタスクを実行し、不可能ならこのタスク実行を拒否する。このような操作をする incomingMessage メソッド及び、他のスケジューラからのステート通知を受けるためのメソッドである stateInformed をメタセクレタリオブジェクトで定義することによりそれぞれのスケジューラオブジェクトを変更することなく、オブジェクトを拡張して新しい環境で使うことを可能とする。

3.2.6 メタポリシーの応用

実際のセクレタリオブジェクトで実現されるポリシーとして、例えば同期を行なうことを考えた時オブジェクトのリード/ライトの構文的なパターンで行なう場合とオブジェクトの持つ意味を用いて同期を行なう場合がある。これは、各オブジェクトにローカルなポリシーとなるので問題はない。また、楽観的なポリシーと悲観的なポリシーの使用も同様である。ここでは、メタポリシーの実際の応用をいくつか検討してみる。ここでは、次の 5 つの応用を考える。

- ガード及びメソッドリレーションの実現
- クライアント/サーバでの機能の分離
- マシン独立/非独立部分の分離
- ブラオリティインヘリタンス
- アトミシティインヘリタンス

1 番目のガードとメソッドリレーションの実現は processMessage、endMethod、stateLookup を用いて行なわれる。つまり、メッセージが到着してメソッドを起動する前に processMessage によりガードのチェックを行なう。これは、stateLookup を用いることによりオブジェクトのバンプステートをアクセスし、それとアーギュメントから実行できるかどうか調べる。そして、この後これから実行しようとするメソッドと排他関係にあるメソッドが現在実行されていないかどうかチェックする。そして、実行可能ならこれを実行する。

2 番目のクライアント/サーバでの機能の分離はサーバが同時に多くのクライアントからアクセスさ

れる時サーバのボトルネックを解消するため重要である。はじめ、サーバはクライアントにサービスを提供するためのいくつかのメソッドを持っている。メタセクレタリオブジェクトのポリシはメソッドの実行時間をモニタしている。そして、実行時間が長いことがわかるとそのメソッドをリデュースする。そして、その結果リデュースしたメソッドをメタセクレタリオブジェクトの Default Action として登録する。そして、その他の部分をクライアント側のメタセクレタリオブジェクトの Default Action として登録する。これにより、サーバの負荷をダイナミックに減らすことが可能となる。

3 番目のマシン独立/非独立の分離では、まずオブジェクトはマシンに依存しないオブジェクトと、マシンに依存するオブジェクトに分離される。マシンに依存するオブジェクトをネットワークを介して転送する場合、マシンタイプがことなる時マシン依存オブジェクトのメタセクレタリオブジェクトはデータを転送先のマシンに合わせて変換する。

4 番目のプライオリティインヘリタンス [1] ではまず、1つのスレッドを管理するメタセクレタリオブジェクトが共有されるこれはそのスレッドのプライオリティを管理してそれをメッセージが伝搬する毎にそのメタセクレタリオブジェクトも伝搬して共有されていく。しかし、問題なのは、共有オブジェクトをアクセスした時である。ロープライオリティのスレッドがオブジェクトをアクセスしている間に、ハイプライオリティのスレッドが送ったメッセージが到着したとする。このとき、ロープライオリティでそのオブジェクトが実行されているためハイプライオリティのメッセージは実行できないのでブロックする。これでは、システム全体のパフォーマンスを悪くするため、この共有オブジェクトをハイプライオリティで実行しなければいけない。これをプライオリティインヘリタンスという。これは、メッセージを受け取るとすぐセンダのメタセクレタリオブジェクトを共有する。共有オブジェクトのセクレタリオブジェクトは2つのメタセクレタリオブジェクトを持つ。しかし、セクレタリオブジェクトは2つのメタセクレタリオブジェクトの持つポリシのうちハイプライオリティのものを継承する。

4 番目のアトミシティもプライオリティ同様メッセージ転送により伝搬して共有されていく。同期は各オブジェクトのセクレタリオブジェクトのローカルポリシで管理されるが、リカバリの管理、コミットメントコントロールはこれで管理される。そして、コミット/アポートメッセージを各セクレタリが受け取るとロックの解除、ログの処理を行なった後、継承したポリシをデタッチする。

このように、メタセクレタリオブジェクトを導入することにより様々なことをシステム自体を変更せずセクレタリオブジェクトとメタセクレタリオブジェクトを用いることにより実現することができる。

4 結論

自律オブジェクトモデルは自律性が重要な問題となる大規模分散システムの基盤モデルとして適当なものであると考えられる。とくに、メタセクレタリの導入によりオブジェクトのポリシをダイナミックに変更して自分のコンテキストに合わせてカスタマイズや拡張を行なうことが可能となった。また、各オブジェクトのポリシ間でコミュニケーションさせることにより、オブジェクト間でメッセージ交換を行なうことなしに協調することを可能とした。現在、DCST 上に自律オブジェクトモデルを実現することを検討中である。そして、実際にメタセクレタリオブジェクトを用いてアトミックトランザクション [4] を実現する予定である。

参考文献

- [1] [Sha 87] L. Sha, R. Rajkumar and J. Lehoczky, *Priority Inheritance Protocols: An Approach to Real-Time Synchronization* CMU Tech. Report, CMU-CS-87-181 1987
- [2] [Yokote 87] Y. Yokote, M. Tokoro, *Experience and Evolution of ConcurrentSmalltalk* In proc. OOPSLA'87
- [3] [中島 88-1] 中島達夫、所真理雄、*DistributedConcurrentSmalltalk* におけるスレッドと共有オブジェクト第4回ソフトウェア科学会全国大会 1988
- [4] [中島 88-2] 中島達夫、所真理雄、オブジェクトモデルのためのアトミックトランザクション第37回マルチメディアと分散処理研究会 1988