ポリシーに基づく分散処理向けアクセス制御モデルの初期的検討

橋本正樹[†] 金美羅[†]
辻秀典[†,††] 田中英彦[†]

情報システムにおけるセキュリティ確保は必須である．近年の情報システムは，ネットワークを介して複数の処理要素が連携処理を行う分散システムが基本である．従って本稿では，分散システムにおけるセキュリティ確保基盤をアプリケーションプラットフォームとしての OS 内で行なうことを目的とし，そのために必要となる OS の機能を検討した上で，中核機能に位置づけられるアクセス制御モデルを提案し検討する．特に本稿においては，ポリシーの記述法とそれを正しく強制するためのアクセス制御機構に焦点をあてて検討する．

Preliminary Studies of Policy-Based Access Control Model for Distributed Environment

Masaki HASHIMOTO,[†] Mira KIM,[†] Hidenori TSUJI[†,††] and Hidehiko TANAKA[†]

This paper describes the use of operating system for the realization of distributed secure computing infrastructure. In particular, it describes a few resource management schemes for distributed environment, addressing the policy expression and the access control mechanism based on the policy. The access control is fine grained so as to realize the appropriate access granularity at distributed environment, and forced mandatory through mechanism such as reference monitor, secure channel, authorization and naming. Finally, we suggest a prototype of our system and the future plan.

1. はじめに

情報システムにおけるセキュリティ確保は必須である．特に，近年の情報システムは，ネットワークを介して複数の処理要素が連携処理を行う分散システムが基本である．しかしながら，現状の分散処理におけるセキュリティ確保は個々のアプリケーションの実装方式，もしくは一部のミドルウェア機能に依存しており，共通のセキュリティプラットフォームが確立されているとは言いがたい．そのため，アプリケーション開発者に対してセキュリティ確保のための負担を強いるとともに，情報システムのセキュリティレベルが個々のシステムによって異なる要因となる．本来であれば，情報システムのセキュリティ確保は共通化されたプラットフォームで行うべきであるので，これを個々のアプリケーションレベルではなく OS レベルで行うことにより，情報システム一般のセキュリティレベルを標準化することが必要不可欠と考える．

本稿では，分散システムにおけるセキュリティ確保基盤をアプリケーションプラットフォームたる OS で行なうことを目的とし，そのために必要となる OS の機能を検討した上で，中核機能に位置づけられるアクセス制御モデルを提案し検討する．

2. 背景と関連研究

本節では，分散システムにおけるアクセス制御の問題を簡単に整理した後，セキュア OS では志向の違いにより解決が難しいことを述べる．そして分散システムに適合したプラットフォームの必要性と提案する方法について述べる．

2.1 分散システムにおける問題認識

OS は計算機においてアプリケーションを動作させるための資源管理を行うプラットフォームであり，下位層のハードウェアを管理し，上位層のアプリケーションに対して，資源を利用するための機能であるシステムコールを提供する．しかしながら，TCP/IP ネット

† 情報セキュリティ大学院大学
Institute of Information Security
†† 株式会社情報技研
Institute of Information Technology, Inc.

ワークの普及により，むしろアーキテクチャやプラットフォームの差異をアプリケーションが通信を介することにより吸収している．そのため，分散システムのセキュリティ確保は，プラットフォームより上位層のソフトウェア階層で行われることが一般的となっている．セキュリティに限らず，SOA（service-oriented architecture）に代表されるように，アプリケーション階層で分散システムの標準化を行なう志向が近年強い．

アプリケーション階層でアクセス制御を行なおうとした場合には，アクセス制御に関する情報はTCP/IP等による通信のペイロードの中に，アプリケーション階層の情報（トークン等）として埋め込まれている．そのため，アプリケーションより下位のソフトウェア層において，処理内容に応じた権限の制御を行うことはできず，その判断はすべてアプリケーションに委ねるしかない．しかしながらアプリケーションそのものは，OS レベルではプロセス等の非常に粗い粒度でしか管理されておらず，アプリケーションレベルにおけるセキュリティホールが OS レベルにおいて大きな問題に拡大してしまう原因となっている．

2.2 セキュア OS と分散システムの適合性

最小特権のセキュリティ原則[1]は，情報システム内の実体が必要以上の権限を有することによって起きる問題を防ぐために考案された．これは，セキュアなシステムを構成するための指針となる概念であり，要素技術としてポリシーの記述法や細粒度の強制アクセス制御[2]（Mandatory Access Control : MAC）機構，アクセス制御対象となる実体の識別／認証を必要とする．ここで MAC とは，任意アクセス制御（Discretionary Access Control : DAC）と対をなす概念であり，DAC がリソースの所有者にアクセス権限の管理を一任するのに対して，MAC ではポリシーに基づいてシステムがアクセス権限を一元的に管理する．

プラットフォームにおける粗粒度アクセスが，分散システムのセキュリティ確保において問題であるとするならば，それが細粒度アクセス制御が可能なセキュア OS にて解決されるのかを考えたとき，以下の理由により解決は難しいと考える．まず，現在のセキュア OS が単一システム志向であることがあげられる．MAC による細粒度アクセス制御はあくまでセキュア OS が稼動しているシステム内にしか及ばず，分散システムとの親和性が低い．さらに，アクセス制御が細粒度すぎることも問題である．細粒度すぎるがゆえ，これを単純に分散システム拡張した場合，複雑性が爆発してしまい非現実的となる．

2.3 分散システムに適合したプラットフォーム要件

分散システムに適合したプラットフォームの要件をひとことでいうと，必要十分な粒度によってシステム全体で透過的なアクセス制御が可能となることである．そのために必要となる要件は以下のとおりとなる[3]．

- ローカルノードにおいても，ネットワーク経由の別ノードにおいても，OS がアクセス主体を透過的に扱える必要がある．
- 透過的扱いとは，分散システム内のアクセス主体の共通化もしくは，各ノードにおけるアクセス主体同士の権限の継承をさす．
- アクセス主体は分散システム内において，許可されたオブジェクトにしかアクセスできないこと．
- オブジェクトに対するアクセス主体のアクセス制御情報は，分散システム内で共有化されていること．
- OS が識別できるアクセス主体およびオブジェクトに対するアクセス制御情報を，アプリケーションに対しても提供できること．

2.4 分散システムに適合した提案モデル

先に述べたプラットフォーム要件のうち，分散システム内で透過的なアクセス制御を行うためのモデルを本稿にて提案する．分散処理に対する共通の枠組みを実現するためには，分散処理が複数ノード／複数アプリケーションの連携によって仕事を行うことを考慮すると，この枠組みはそれらすべてから等しく利用可能である必要がある．加えて，アクセス制御を行うコンポーネントとしての観点からすると改ざんに強く，さらにアクセス制御の対象を確実に捕捉できることが期待される[4]．そのため，提案モデルでは OS で分散処理のアクセス制御を行う．複数ノード／複数応用から利用可能であると言う点では，ミドルウェアやライブラリによってアクセス制御を実現する方法もあり，この場合は OS 層以下の差異を吸収できる利点があるが，一方でセキュリティの観点からすると改竄防止や確実な実施という点において，より下位の OS で実現した方が有利である．このため，近年における情報システムのセキュリティ強化に対する要請を考慮すると，応用の実行環境としての OS がより安全な分散処理志向のアクセス制御機能を備えるべきである．

次に，セキュリティ管理の方法を述べる．分散処理は単一システムでの処理と比較して複雑であり，潜在的に多数のアクセス制御対象を含む可能性があるため，アクセス制御情報を規則として記述する負担を軽減し，全体を把握しやすい仕組みが必要となる．そのため，提案モデルでは分散システム全体にまたがった

ポリシーによるアクセス制御情報を記述する．リソースの所有者がアクセス制御情報を個別に設定していくような方法は，非常に単純で管理者の負担を軽減できるが，全体として要求するアクセス制御を処理毎に強制できないため，統一的なセキュリティ管理が難しい．一方で，ポリシーによるアクセス制御情報の記述は，複雑な処理を必要とする上，管理者にはポリシー記述の負担をかけるが，システム全体としてのセキュリティレベルの平均化と向上が期待できる．

3. 提案モデルの概要

本節では，分散処理のアクセス制御を共通の枠組みで実現するモデルの基本構成について述べる．提案するモデルはポリシー記述とアクセス制御機構が中核であり，アクセス制御機構は Capability を用いてアクセス制御を行なう．次にこれらの 3 点について述べる．

3.1 ポリシー記述

ポリシーは個々の分散処理に関して，必要十分な権限付与を表現できる必要がある．一方で，分散処理の複雑さに対応するために，ポリシー記述の負担を軽減し，ポリシー全体を把握しやすくするような仕組みが求められる．そのためポリシー記述に関しては，以下のような機能を必要とする．

細粒度の記述と粒度の制御 個々の分散処理を正しく記述するためには，プロセスやノードといった粗い粒度によるものでは不十分であり，トランザクション下で実際にアクティビティを行う実体（インスタンス）を記述できる必要がある．一方で，ポリシー記述による管理者の負担を軽減するためには，任意のインスタンスを含むグループを構成し，それに対するアクセス制御を記述できる必要がある．インスタンス毎にアクセス制御を記述できると同時にグループ化をサポートすることによって，必要に応じた粒度の制御が可能となる．

権限遷移の記述 分散処理においては，ひとつのインスタンスがトランザクション内で複数の仕事を行う場合があるので，仕事毎にアクセス権限を付与できるのが望ましい．しかし，トランザクションで必要とされるすべてのアクセス権限をインスタンスに付与するのは過剰であり，仕事毎に付与するアクセス権限を切り替えるような仕組みが必要となる．そのため，提案モデルではアクセス権限の遷移を適切に記述できる必要がある．

サブタスクの記述 分散処理においては，あるタスクが別のサブタスクを生成することによって仕事を処理する場合があるが，仕事に必要十分なアクセス権限はタスク毎に異なる可能性がある．一般的な手法では，サブタスクにも親となるタスクからアクセス権限が引き継がれるが，この方法ではサブタスク毎に異なるアクセス権限を付与することができない．そのため，提案モデルにおいてはサブタスク毎に独立したアクセス権限を付与するような記述ができる必要がある．

3.2 アクセス制御機構

ポリシーに基づいたアクセス権限管理を OS で実施するためには，ポリシーで表現されたアクセス制御情報を実際のアクセス制御機構が実施できる必要がある．そのためには，ポリシーで表現されたアクセス制御情報を過不足なく柔軟に強制できる仕組みが必要となるため，提案モデルではアクセス制御機構に関して，以下のような機能を必要とする．

認可情報の関連付け 提案モデルのアクセス制御機構は認可情報を *Initiator* に関連付ける．分散処理ではアクセス主体となるインスタンス (*Initiator*) の頻繁な生成／削除を考慮する必要があるが，認可情報をアクセス対象のインスタンス (*Target*) に関連付ける仕組み[5] では *Initiator* の情報が別に管理されるので，認可情報と *Initiator* の関連付けに関する整合性を保つのが難しい．

参照情報と認可情報の不可分 *Target* に関する参照情報と認可情報は不可分に扱う．参照情報と認可情報を別々に扱ってアクセス制御の際に統合するような仕組みでは Confused Deputy Problem[6] を防ぐのが難しい．参照情報と認可情報が不可分であれば，*Initiator* は常に特定の目的の基に参照を行うため，そのような問題が起こりにくくなる．

認可情報の委譲 新しく生成／削除されるインスタンスに対して，*Initiator* が認可情報を委譲できるようにする．これは，分散処理における権限遷移やサブタスクへの認可情報設定のためであり，またアクセス制御の柔軟性を保つためには *Initiator* に関する動作をあらかじめ固定できないので，認可情報をアクセス主体毎に委譲できるようにする．

アプリケーションとの連携 アクセス制御機構がポリシーに基づいた制御を行うためには，制御対象のインスタンスを適切に扱う必要がある．そのため，アクセス制御機構はノードやアプリケーション内で生成されるソフトウェアコンポーネントを確実に捕捉してポリシーに従ったアクセス権限の付与を行い，強制できなければならない．

3.3 Capability

Capability[7),8)] は，オブジェクトの識別子とそれに

対する許可された操作を表す偽装不可能な対として構成される．UNIX システムのファイル記述子[9] は，*Capability* と類似したものである．*Capability* によるシステムは，各々のプロセスに *Capability* が関連付けられ，*Capability* に定義された操作を実行することができる．*Capability* をアクセス制御の基礎とするシステムのセキュリティは，以下の 3 つによって担保される．

1. *Capability* には耐タンパ性があり，偽装不可能である．
2. プロセスは，認められた特定の手続きによってのみ *Capability* を獲得する．
3. *Capability* は，認められたプロセスにのみ与えられる．

保護ドメインは，アクセス可能なサブシステムを定義した *Capability* のセットであり，*Capability* システムの基本的な発想は，アプリケーションと OS がそれぞれの保護ドメインに明確に分離されていることによって可能となる．

4. 提案モデルの構成

本節では，提案モデルを構成する各コンポーネントについての概要を示す．提案モデルは，分散処理に関わるアクセス制御情報を記述したポリシー部とそれに基づいて実際のアクセス制御を実施するアクセス制御機構部から構成される．

4.1 ポリシー

ポリシーは，あらかじめ定義された構文に基づいてアクセス制御情報が記述され，所定の処理を経てアクセス制御機構に権限付与に関する規則を与える．アクセス制御情報は，{*Initiator : Target : Operation*} として記述されるが，提案モデルではアクセス制御情報を AHMED らによる CSCW システム向けの手法[10] によってグループ化し，動的な権限の付与を実現する．また，AHMED らの手法ではユーザに対して *Role* を割り当てるが，提案モデルにおいてはあらかじめ粒度が制御されたセキュリティドメインに対して *Role* の割り当てを行う．ポリシーのグループ化と動的な権限付与は以下の情報を利用して行われる．

アクティビティの雛形　提案モデルにおいては，アクティビティに応じた雛形を定義し，それに基づいたアクセス制御情報の記述を行う．雛形には，分散処理における *Role* を基礎とした規則の記述がなされており，個々のインスタンスに対しては動的に *Role* を割り当てることによって，雛形に対するインスタンスの割り当てを柔軟に行うことができる[11]．雛形は，以下のような項目を含む．

- *Role* の設定
- *Role* 毎に関連付けた処理
- 生成／アクセスされるオブジェクトタイプ
- 生成される子アクティビティの雛形
- ライフサイクル内で生成される動的なイベントの設定

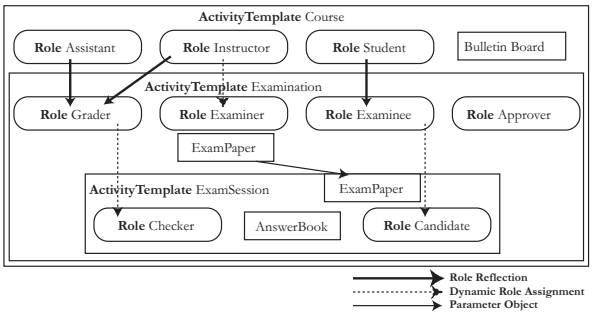

図 1　階層化された処理の記述
Fig. 1　Activity Template

Role　*Role* は許可された処理を関連付けられた保護ドメインであり，アクティビティの雛形で定義されたオブジェクトに対して所定の処理を行う．*Role* の定義は，当該 *Role* に対するセキュリティドメインの参加可否／ *Role* に許可された処理／処理が実行されるために必要となる *Role* の状態に関する三つの定義を含む．

処理の定義　*Role* 毎に関連付けられる処理の定義には，事前状態と実際の処理内容が記述される．事前状態の定義は，当該処理を行うために必ず満たしていなければならない状態を示し，アクティビティの雛形で定義されるイベントを用いて記述する．実際の処理内容は，オブジェクト毎に提供される操作や新しいオブジェクトの生成，子アクティビティの生成を記述する．

セキュリティドメイン　提案モデルにおいては，セキュリティドメインを定義し，同様のアクセス権限を必要とするようなインスタンスのグループ化を行う[12]．*Role* の割り当ては，セキュリティドメインに対して行われる．

イベント　分散処理の動的な特性を扱うために，イベントとイベントカウンター[13] を利用する．イベントは，処理の定義内で記述されることによって *Role* の動作を規定するものであるが，具体的には *Role* によって実行される処理名や子アクティビティの生成／削除によって記述される．また，*Role* によって実行される各処理には少なくとも *start*

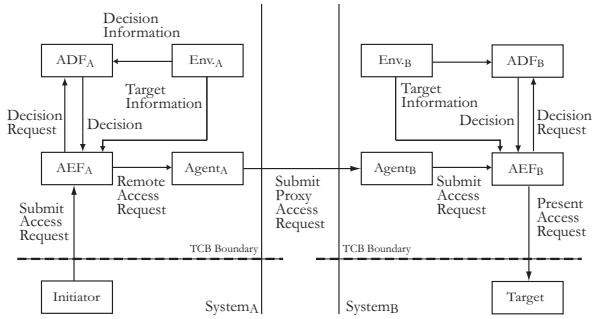

図 2 アーキテクチャ概要
Fig. 2 Architectural Overview

と *finish* の二つのイベントが関連付けられる．

子アクティビティ アクティビティは，あるサブタスクを行うために子アクティビティを利用することができる．子アクティビティは，親アクティビティの制限下で定義され，独立した雛形によって記述される．子アクティビティによって，階層化した処理を記述することが可能であり，子アクティビティは親アクティビティの制限下でオブジェクトにアクセスする．

図 1 では，アクティビティと *Role* による処理の記述例を説明する．この例では，アクティビティ *course* に *instructor*，*assistant*，*student* の 3 つの *Role* と子アクティビティ *examination* が定義されており，これには *grader*，*examiner*，*examinee*，*approver* が *Role* として設定されている．また，*course* は，化学や物理などの科目毎に生成することが可能であり，それぞれで任意の数の *examination* を生成することができる．さらに *examination* では，イベント *ExamSession* をトリガーに子アクティビティ *ExamSession* が生成され，*candidate* と *checker* の 2 つ *Role* が設定される．メンバーの設定方法は 2 種類あり，この例では，*role reflection* によって *instructor* と *assistant* のメンバーがそのまま *grader* に，*student* のメンバーがそのまま *examinee* にそれぞれ設定されているが，*dynamic role assignment* によって変則的にメンバーの設定を行うことできる．

4.2 アクセス制御機構

アクセス制御機構のアーキテクチャを図 2 に示す．以下に，各コンポーネントについて詳細を述べる．

Initiator *Initiator* はアクセス要求を発行するインスタンスであり，その際はサービス提供を希望する *Service* とそれに対する操作の名前を指定する．また，アクセス要求が許可された場合は *AEF* を仲介したアクセスレスポンスを受け取り，拒否された場合は *Access Reject* を受け取る．

Service *Service* は，提供する機能単位にオブジェクトをグループ化したインスタンスである．*Service* は *AEF* からアクセス要求を受け取り，レスポンスを返す．*Service* は複数のオブジェクトが連携することによって機能するものであり，*Service Table Directory* に登録されることによってリモートノードからの利用が可能となる．*Service* が受け入れ可能な操作は，ポリシーに基づいて *Service* 毎に設定される．

Access Enforcement Function *Access Enforcement Function*(*AEF*) は，アクセス制御対象へのアクセスを仲介するコンポーネントであり，アクセス制御機構による認可決定を *Initiator* に強制する．*AEF* は OS のコンポーネントとして実装され，アプリケーション層に定型化された手続きを提供する．*Initiator* は，*AEF* から提供される手続きに *Service* と操作の名前を含む変数を渡すことによってのみ *Service* にアクセス可能であり，*AEF* は *Service* へのアクセスに関わる詳細な手続きとデータを隠蔽する役割を果たす．また，*AEF* は *ADF* から受け取った情報を元に *Service* の場所を特定し，それがリモートノードであった場合は *Agent* に代理アクセスを依頼する．ローカルノードであった場合は，*Initiator* に代わって *Service* へのアクセスを行う．

Access Decision Function *Access Decision Function*(*ADF*) は認可決定を行うコンポーネントであり，*AEF* からの認可決定要求に対して認可決定と，アクセス許可の場合は *Service* の場所を示す情報を返す．*ADF* は OS のコンポーネントとして実装され，*AEF* に対するインターフェースのみを提供する．

Agent *Agent* は分散処理に関わる各ノードに存在し，相互にネットワーク通信を行うコンポーネントである．各 *Agent* は互いを識別することができ，各 *Agent* 間は安全な通信を行うことができる．*Agent* はローカルノードの *AEF* からリモートノードに存在する *Service* へのアクセス要求を受け取り，リモートノードの *Agent* に代理アクセスを依頼する．代理アクセスを依頼された *Agent* は，サブタスクを生成することによって，あらためて *Initiator* として当該オブジェクト対するアクセス要求を発行し，レスポンスを元の *Agent* に返す．レスポンスを受け取った元 *Agent* は，ローカルノードの *AEF* にそれを中継する．

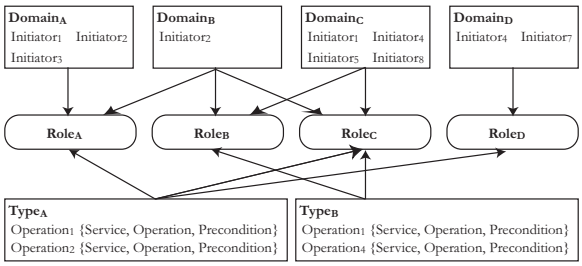

図 3 アクセス制御粒度の制御
Fig. 3 Granularity Control

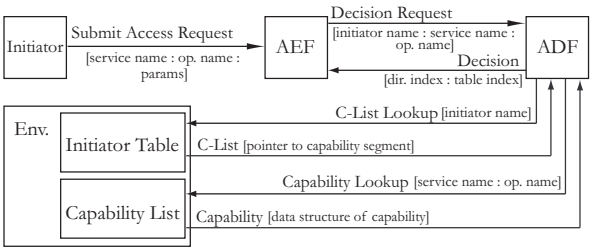

図 4 認可決定の手順
Fig. 4 Decision Making Procedure

Environment *Environment* は OS 内部で保持される属性情報の集合であり，インスタンス毎に関連付けられた *Instance Table* と *Capability* のリスト *(C-List)* を含む．*Instance Table* は，*C-List* と共にサブタスクの情報を記述するために利用され階層構造をとることが可能である．*Capability* は，アクセス権限を含む *Service* の参照情報であり，アクセス制御機構は *Initiator* に割り当てられた *Capability* を検査することによってアクセス制御を行う．*Environment* は分散処理に関わる各ノード毎に設定され，アクセス制御の際に各コンポーネントから利用される．

Capability *Capability* は，*Service* の参照情報とそれに対する操作を対で記述することによって，*Service* の参照先を指定するものである．図 2 では，*Decision Information* と *Target Information* が *Capability* から提供される．提案モデルでは，ポリシーに基づいて *Capability* が生成され，それを *Initiator* 毎に関連づける．*Initiator* は，関連付けられた *Capability* に基づいてアクセス対象を参照するので，ある *Initiator* に関連付けられた *Capability* の集合は，その *Initiator* がアクセス可能な参照先と操作の対の集合である．提案モデルでは，この集合を *Capability List(C-List)* と呼び，認可決定の際は *ADF* によって該当アクセスに必要な *Capability* の存在を検査される．また，実際に *Service* のアクセスを行う際には *AEF* によって参照先の指定に利用される．

4.3 アクセス制御粒度の制御

提案モデルでは，*Role* とセキュリティドメインによってアクセス制御粒度の制御を行う．図 3 にその様子を示す．提案モデルにおけるアクセス制御粒度の制御は，分散処理を構成する最も小さなアクセス制御単位であるインスタンスを段階的にグループ化することによって実現される．

はじめに，同様のアクセス権限付与が期待されるインスタンスはセキュリティドメインにグループ化することが可能である．ひとつのインスタンスは複数のセキュリティドメインに所属することができ，ひとつのセキュリティドメインには複数のインスタンスが所属することができる．また，ひとつのセキュリティドメインにひとつのインスタンスのみを所属させることによって，細かい粒度のアクセス制御単位を構成することもできる．また，同様の権限によってアクセスされることが期待される処理はセキュリティタイプにグループ化することが可能である．セキュリティドメインとセキュリティタイプは，グループ化する対象によって名前が異なるが同様の性質を有する．セキュリティタイプは *Service* への参照情報と操作種別，実行に必要な状態の組み合わせの集合として構成される．

提案モデルでは，セキュリティドメインとセキュリティタイプを *Role* として統合することによって，任意の粒度でアクセス制御を行うことが可能となる．*Role* は，セキュリティドメインとセキュリティタイプをさらに上位レベルで抽象的にグループ化するものであり，分散処理でのアクセス権限付与やアクセス権限の遷移は *Role* の付与／変更によって行われる．実際にアクセス制御機構に制御情報を設定する際は，*Role* によってグループ化されたインスタンスを再び展開し，個々の処理を *Service* と操作の対によって表現した *Capability* を生成した上で，*Initiator* 毎に関連づける．*Initiator* 毎の *C-List* は，*Initiator* に割り当てる可能性がある *Role* 単位に生成され，あらかじめアクティビティの雛形で定義されたポリシーに基づき，処理の過程で切り替えられていく．

5. プロトコル概要

本節では，提案モデルが動作する仕組みについて，各コンポーネントが連携する手順を示す．特に，アクセス制御の粒度を制御する手順とアクセス権限の遷移を実現する手順，認可決定とそれに基づいて対象リソースにアクセスする手順に焦点を当てて説明する．

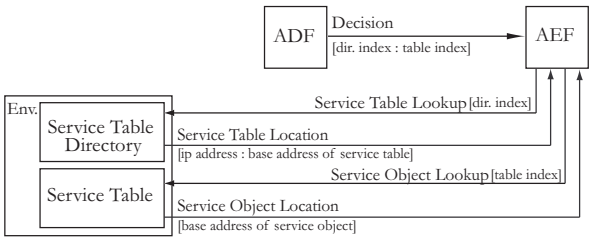

図 5 参照先特定の手順
Fig. 5 Location Looking up Procedure

5.1 認可決定

提案モデルでは，*ADF* で *Capability* の検査を行い，それに基づいた認可決定が実施される．図 4 に認可決定時に各コンポーネントが連携する様子を示す．

はじめに，*Initiator* が *AEF* によって提供される手続きに則って *Service* の名前と操作の名前，パラメータによるアクセス要求を行う．*AEF* は，アクセス要求を発行した *Initiator* の名前を特定した上で，受け取った *Service* の名前と操作の名前と共に *ADF* に渡し，認可可否の判定を求める．

Initiator，*Service*，操作の名前を受け取った *ADF* は，まず *Initiator* の名前を基に *Environment* 内の *Initiator Table* を検索し，該当 *Initiator* の記述子から *C-List* を見つける．その後，*Service* と操作の名前を基に *C-List* から，該当 *Service* 名と操作名を含む記述子によって構成される *Capability* を検索し，認可可否の判定を行う．

ADF による認可可否は *Capability* の有無によって検討され，*Capability* がある場合はアクセス許可であり，ない場合はアクセス拒否となる．アクセス許可であった場合は，同時に該当 *Capability* から *Service* への参照情報を抽出し，それを *AEF* に渡すことによってアクセス許可判定とする．アクセス拒否の場合は，*Service* への参照情報の代わりに *Access Reject* が返される．

5.2 Service の参照

提案モデルでは，*ADF* による認可決定と同時に *Service* への参照情報が *AEF* に提供される．*AEF* は，これに基づいて *Service* の参照先を特定し，それがローカルノードであった場合は指定された操作によるアクセスを行う．また，*Service* の参照先がリモートノードであった場合は，*Agent* に代理アクセスを要求する．図 5 に参照先特定時に各コンポーネントが連携する様子を示す．

はじめに，*ADF* から *Service* への参照情報を受け取った *AEF* は，参照情報として渡された *Directory Index* を基に *Service Table Directory* から特定の記述子を見つける．この記述子には *Service* が存在するノードの識別子とそのノードの主記憶領域内で *Service Table* が存在するベースアドレスが記述されており，*AEF* はまずノードの識別子から *Service* がローカルノードで提供されているのか否かを判定する．

Service の参照先がローカルノードであった場合，*AEF* は *Service Table Directory* 内の記述子から読み込んだ *Service Table* のベースアドレスを利用して *Service Table* の主記憶内でのベースアドレスを特定し，*ADF* から参照情報として渡された *Table Index* を基に特定の記述子を見つける．この記述子には *Service* の最終的な参照先が記述されており，これに基づいて *AEF* が *Service* へのアクセスを行った後，アクセスレスポンスを受け取って *Initiator* へ渡す．

Service の参照先がリモートノードであった場合，*AEF* は *Agent* にリモートノードへのアクセスを依頼し，これを受け取った *Agent* は，該当ノードの *Agent* に代理アクセス要求を発行する．代理アクセス要求を受け取った該当ノードの *Agent* は，自身を *Initiator* として，通常のアクセス要求手続きを開始し，アクセスレスポンスを受け取った上でそれを代理アクセス要求を行った *Agent* に返す．アクセスレスポンスを受け取ったローカルノードの *Agent* は，それを *AEF* に中継し，*AEF* が *Initiator* にアクセスレスポンスを返す．

6. 提案システム評価

本節では，提案モデルのセキュリティに関する特性について考察する．はじめに，第 3 節で述べたような設計要求を満たすことによって，提案モデルが分散処理におけるアクセス権限の管理を適切に扱えているのかどうかを評価する．その後，いくつかの問題点に対する提案モデルの有効性を検証する．

6.1 設計要求評価

第 3 節において，分散処理のアクセス権限管理を共通の枠組みで行うために必要となる特性について検討し，7 つの機能（細粒度の記述と粒度の制御，権限遷移の記述，サブタスクの記述，認可情報の関連付け，認可情報と参照情報の不可分，認可情報の委譲，アプリケーションとの連携）を導き出した．提案モデルとプロトコルは，これらの要求に基づいて設計されている．

細粒度の記述と粒度の制御は，任意の粒度で記述したインスタンスを *Role* とセキュリティドメインを利用してグループ化することにより実現される．最上位レベルで抽象化されたアクセス制御情報は，セキュリ

ティドメインとセキュリティタイプを *Role* として統合することによって表現される．これによって提案モデルは，粒度を適切に調整した上でアクセス制御を行うことができる．

次に，権限遷移の記述とサブタスクの記述であるが，これはイベントと子アクティビティによって実現され，特定の処理をトリガーにした権限の遷移やサブタスクの生成による新たなアクティビティの初期化を制御することが可能になる．これによって提案モデルは，分散処理における特定の状況化でインスタンスに与えるアクセス権限を必要十分なものに変更していくことができる．

また，認可情報の関連付け，認可情報と参照情報の不可分，認可情報の委譲については *Capability* を利用することによって実現しており，提案モデルのアクセス制御機構は *Role* 単位に *C-List* を設定した上で *Initiator* に関連付けてアクセス権限を制御する．これによって提案モデルは，ポリシーで記述されたアクセス権限付与に関する規則に基づいたアクセス制御を適切に強制することが可能になる．

最後にアプリケーションとの連携であるが，これはアクセス制御機構がアプリケーション向けにいくつかの手続きを提供し，それをアプリケーションが利用することによって実現される．

6.2 セキュリティ評価

提案モデルは，第 3 節に挙げたような 7 つの機能的な要求に焦点を当てて設計されているが，本節では分散処理におけるセキュリティ管理に関して特に考慮すべき点について検証する．

6.2.1 情報フロー制御と Reference Monitor

情報フローの制御は，ポリシーに基づいたアクセス制御を強制するために必須であり，これを行うためには *Reference Monitor*[14] がアクセス制御対象となるインスタンスをすべて捕捉できなければならない．逆にいうと，*Reference Monitor* が捕捉できないインスタンスにアクセス制御を強制するのは不可能であると言える．一般的な単一システムにおいては，管理する資源へのすべてのアクセスに対してシステムコールの利用を強制することによって，OS カーネルが *Reference Monitor* の役割を果たす．これによって，OS カーネルは，あらかじめ決められたアクセス制御規則を適用することが可能になる．

提案モデルは分散処理に対してアクセス制御を行うものであるが，複数ノード間で流動的に移動し，OS カーネルから捕捉不可能なプロセス内のインスタンスをアクセス制御対象としている．提案モデルでは，*Capability* によるアクセス制御情報を受け渡すことによってノード間を移動するインスタンスに対するアクセス制御を可能としているが，一方でプロセス内のインスタンスの捕捉に関しては，当該プロセスが OS カーネルから提供される手続きを適切に利用することを期待している．これはプロセスに *Reference Monitor* の役割を委譲しているものであり，提案モデルは階層的な *Reference Monitor* 構造をとっているとみることができる．それぞれの *Reference Monitor* は上位の *Reference Monitor* の制限内で委譲された範囲のアクセス権限管理を行うため，必ずしも最上位の *Reference Monitor* である OS カーネルとポリシーがすべての処理のアクセス制御ができるとはいえない．

しかし，そもそもひとつの *Reference Monitor* が複数ノード間のすべてのアクセスを捕捉するためには，ローカル／リモートによらず，すべてのアクセスが迂回不可能なポイントに *Reference Monitor* を置く必要があるが，一方で，そのようなポイントを設けるのは困難である．そのため，いずれにせよ *Reference Monitor* の階層化を避けることはできない．そのため，ここで重要になるのは上位階層の *Reference Monitor* が下位階層を可能な限り制限して制御することであり，提案モデルにおいてはそれが可能である．

6.2.2 Confused Deputy Problem

deputy とは，複数のインスタンスから来るアクセス制御要求の認可判断を管理するプログラムである．*confused deputy* は，不適切な認可判断をしてしまう *deputy* であり，コンピュータセキュリティにおけるひとつの大きな課題はこれを防ぐことにある．*confused deputy problem* は，World-Wide Web を含む多くのシステムにおいて一般的なセキュリティインシデントを起こす原因となっている．

一般的な *confused deputy problem* について，コンパイラを例に説明する．ここに，課金情報を蓄積するための BILL というファイルに書き込み許可されたコンパイラを仮定する．このコンパイラを実行するとき，ユーザはデバッグ情報を出力するファイルを指定できる．この時，仮にユーザが BILL をデバッグファイル名として指定すると，コンパイラは誤って課金情報をデバッグ情報で上書きしてしまう．このような動作は，コンパイラに与えられた書き込み権限が誤った目的に利用されたことによって起こる．仮に，コンパイラが課金情報を書き込む権限とデバッグ情報を書き込む権限を別々に保有することができれば，BILL へのアクセス権限はあいまいでなくなる．その上で，コンパイラが意図した目的の元に権限を使い分ければ，

confused deputy problem が起こることはない．つまり，アクセス主体が与えられた権限を認識し，それを使い分ければ *confused deputy problem* を防ぐことが可能になる．

このような目的のために，提案モデルではアクセス権限を *capability* として *Initiator* に関連付けているが，一方でこれによって完全に *confused deputy* を防げるものではない．*confused deputy problem* を完全に防止するためには，*Initiator* が目的に基づいて適切な *capability* を選択し，利用することが期待される．提案モデルにおいては，ポリシーに基づいて *Initiator* の *C-List* を切り替えるので，ポリシーが正しく記述されていればこれが可能となる．

7. 結論と今後の課題

本稿においては，分散システムにおける権限管理を適切に行うためのモデルを提案した．提案モデルは，分散処理の権限管理を記述するポリシーの記述法と，これを強制するためのアクセス制御機構から構成される．提案モデルにおいては，アクティビティに応じた雛形を定義し，それに基づいた分散処理の記述を行う．雛形には，*Role* を基礎とした規則の記述がなされており，個々のインスタンスに対してはこの規則に基づいて動的に *Role* を割り当てるので，アクセス権限に対するインスタンスの設定を柔軟に行うことができる．また，記述されたポリシーを適切に強制するために，提案モデルにおいては *Capability* を利用したアクセス制御を行う．アクセス制御の際には，各々のインスタンスに設定された *Capability* の集合である *C-List* を検査することによって，認可決定が行われる．また，動的な権限の変更は，*C-List* をポリシーに基づいて適切に変更することによって実現される．

今後は，実際のシステムに対する提案モデルの実装を検討すると同時に，第 6 節で示した階層的な *Reference Monitor* について，下位の *Reference Monitor* をより細かく制限する手法について検討する．また，分散処理を行う各ノードの *Agent* 間で安全に情報を交換する手法についても検討していく．加えて，ポリシーを各ノードで管理されるテーブルに適切に設定していく手順についても今後の課題とする．

参 考 文 献

1) Peter J. Denning, “Fault Tolerant Operating Systems”, ACM Computing Surveys (CSUR), v.8 n.4, p.359-389, December 1976.
2) Jonathan M. McCune, Stefan Berger, Ramon Caceres, Trent Jaeger, Reiner Sailer, ”Bridging Mandatory Access Control Across Machines”, November 4, 2005.
3) 辻 秀典, 橋本 正樹, 金 美羅, 田中 英彦. “アプリケーション・プラットフォームとしてのセキュア OS に関する初期的検討”. 情報処理学会 第 38 回コンピュータセキュリティ研究会, 2007 年 7 月.
4) U. S. Dept. of Defence. ”Department of Defence trusted computer system evaluation criteria,” Dept. of Defence, December 1985.
5) Jerome H. Saltzer, “Protection and the control of information sharing in Multics”, Comm. ACM, 1974.
6) N. Hardy. “The Confused Deputy (or why capabilities might have been invented)”. Operating Systems Review 22(4), October 1988, pp. 36 - 38.
7) Jerome H. Saltzer, and Michael D. Schroeder. “The Protection of Information in Computer Systems”. In Proceedings of the IEEE 63, 9 pp. 1278 - 1308. September 1975.
8) J. B. Dennis and E. C. Van Horn, “Programming Semantics for Multiprogrammed Computations”, Comm. ACM, 1966.
9) K.Tompson, “UNIX implementation”, The Bell System Technical Journal, 57(6):1931-1946, July 1978.
10) Ahmed, T. and Tripathi, “Specification and verification of security requirements in a programming model for decentralized CSCW systems”, ACM Trans. Inf. Syst. Secur. 10, 2 (May. 2007), 7.
11) Barkley. J, “Implementing role-based access control using object technology”, In Proceedings of the First ACM Workshop on Role-Based Access Control (Gaithersburg, Maryland, United States, November 30 - December 02, 1995). C. E. Youman, R. S. Sandhu, and E. J. Coyne, Eds. RBAC ’95. ACM Press, New York, NY, 20.
12) P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell, “The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments”, In Proceedings of the 21st National Information Systems Security Conference, pages 303-314, Oct. 1998.
13) Roberts, P. and Verjus, J.-P, “Toward autonomous descriptions of synchronization modules”, In Proceedings of IFIP Congress, North-Holland, Amsterdam, 981-986, 1977.
14) J. Anderson, ‘Computer Security Technology Planning Study” , Air Force Elect. Systems Div., ESD-TR-73-51, October 1972.