社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

# 分散システムにおけるCapabilityを用いた資源アクセス制御

橋本　正樹† 　藤澤　一樹† 　宮本久仁男† 　金　美羅† 　辻　秀典†,††
田中　英彦†

† 情報セキュリティ大学院大学 神奈川県横浜市神奈川区鶴屋町 2-14-1
†† 株式会社情報技研 東京都中央区日本橋小舟町 3-1
E-mail: †dgs074105@iisec.ac.jp

**あらまし**　分散システムは、単一システムと比較して経済性、速度、冗長性、拡張性、柔軟性といった面で優れているため、その実現に向けて従来より多くの研究が行われてきた。しかしながら、それらの諸研究における実装の多くはミドルウェアやアプリケーションのような上位層で実現されているために権限管理の粒度が粗くなり、システム全体に対する適切な安全性確保を困難にしている。このため本研究では、ディペンダブルな分散システムの構築を目的としたシステムソフトウェアによる細粒度の権限管理方式を検討する。また、その実装としてCapability を利用した手法について検討する。
**キーワード**　アクセス制御、分散システム、ケーパビリティ

# Capability-Based System for Distributed Environment

Masaki HASHIMOTO†, Kazuki FUJISAWA†, Kunio MIYAMOTO†, Mira KIM†, Hidenori TSUJI†,††, and Hidehiko TANAKA†

† Institute of Information Security
2-14-1, Tsuruya-cho Kanagawa-ku, Yokohama
221-0835, Japan
†† Advanced Institute of Information Technology, Inc.
3-1 Nihonbashi-Kobunacho, Chuo-ku, Tokyo
113-0024, Japan
E-mail: †dgs074105@iisec.ac.jp

**Abstract**　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 fine-grained protection and the principle of least privilege. These are compared each other in terms of the features they offer in the context of secure computing: Reference monitor concept, secure channel, authorization and naming. Finally, we suggest a prototype of our system and the future plan.
**Key words**　Access Control, Distributed System, Capability

## 1. はじめに

分散システムは、単一システムによる情報システムと比較して経済性、速度、冗長性、拡張性、柔軟性といった面で優れている。そこで、その実現に向けた研究が 1960 年代より盛んに行われ、大きな成果をあげてきた。それらの諸研究はいずれも性能が重大なテーマのひとつであったが、この問題は昨今の基盤技術の進歩と共に大きく改善されようとしている。しかし一方で、いくつかのテーマに関する研究は未だ発展途上にあり、中でもセキュリティに関しては最小特権のセキュリティ原則 [1](PLP: Principle of Least Privilege) の適用が課題となっている。PLP は、システムをセキュアに構成するための設計指針となる概念であり、タスクの実行に最小限の資源のみを関与させることによって、余分な特権によるシステムへの影響を防ぐものである。PLP を実現するためには、タスクに関与するアクセス主体／アクセス対象／実行内容を細かく制御できる必要があるため、この機能は細粒度アクセス制御と呼ばれ、PLP を支える要素技術となっている。

このような立場から、筆者らは分散システムにおける PLP の実現に向けた研究を行っている。本稿では、分散システムにおける細粒度アクセス制御に関して、設計要件の定義とそれに基づいたアクセス制御モデルの提案を行う。PLP の要素技術は、ポリシー記述法、細粒度アクセス制御、アクセス制御対象となる実体の識別／認証であるが、特に細粒度アクセス制御に関しては、細粒度のアクセス制御を強制的に OS 層から実施するセキュア OS が大きな成果をあげて、一般にも普及し始めている。しかし一方で、セキュア OS のアクセス制御機構は単一システムでの利用を想定しており、分散システムへの拡張には問題がある。

そこで本研究では、分散システムにおいて細粒度アクセス制御を実現するための要件を定義し、これに基づいたアクセス制御モデルを提案する。実応用での利用を考える時には、アクセス対象の位置を特定する参照情報の扱いや制御情報の改竄、Confused Deputy Problem [2], [3] などに耐え得る細粒度アクセス制御を行う必要がある。ここで Confused Deputy Problem とは、アクセス制御を行うプログラムが不適切な制御によって本来意図された情報保護を行えない問題である。本稿では上記の問題を解決するために Capability [4], [5] を用いた仕組みについて検討する。

## 2. 関 連 研 究

本章では、はじめに PLP についてその概要と効果を説明する。その後、この領域に関する近年の状況を説明し、本研究との関連を述べる。

### 2.1 最小特権のセキュリティ原則

PLP は、システムをセキュアに構成するための設計指針となる概念であり、タスクの実行に最小限の資源のみを関与させることによって、余分な特権によるシステムへの影響を防ぐ。PLP は、要素技術としてポリシー記述法や細粒度アクセス制御、アクセス制御対象となる実体の識別／認証を必要とし、中でも細粒度アクセス制御は、あらかじめ制御対象となる実体に対してポリシーに基づいた細粒度の属性を設定し、それを評価することによって行われる。ACL [6](ACL)、Capability、Flask [7] といったものが細粒度アクセス制御機構の代表的なものであり、属性の設定方法や評価の仕組みがそれぞれに異なっているが、そのアクセス制御に利用するコードとデータを不正な改竄から防ぐために特別な領域に保護している点で一致している。

例えば、UNIX は ACL によるアクセス制御を行っているシステムであるが、あるプロセスがシステム内のファイルをオープンしようとするときには、システムがまずアクセス対象となるファイルに設定されている保護ビットをアクセス主体に設定されている実効 UID、実効 GID と照らし合わせ、アクセスが許可されているかどうかを評価する。また、Flask の一実装である Security-Enhanced Linux [8] においては、あるプロセスがシステム内のファイルをオープンしようとするとき、システムはまず通常の UNIX システム同様のアクセス権限の評価を行う。その後、システムコールがフックされ、Flask で設定されている SID とドメインによる追加の評価が行われる。一方で、UNIX と Flask は共にアクセス制御に利用するコードとデータをシステム内の特別な保護領域に置くことによって、その安全性を担保している。

PLP の実現を目指したこれらのアクセス制御機構は、セキュリティを考慮した情報システムにおいて利用されている。しかし、その要素技術である細粒度アクセス制御は、システムの規模が大きくなるに従ってアクセス主体と資源に付与する属性の管理が煩雑になるため、拡張性に問題がある。加えて分散システムへの適用を考えた時には、ネットワークを介するためにアクセス主体／資源の認証や重要な情報の漏洩を防ぐような対策を行う必要がある。

### 2.2 近年の研究

PLP の実現は、セキュリティに関連する活発な研究テーマのひとつである。近年の研究では、アクセス主体や資源を表現する専用のポリシー言語に関するもの [9]〜[11] や細粒度の強制アクセス制御機構に関するもの [8], [12], [13]、複数システム間のセキュアな認証技術に関するもの [14], [15] に焦点があてられている。これらの諸研究は、PLP の実現に寄与するものであり、実運用されているシステムで実装することによってセキュアなシステム設計に有効であることを証明している。特に、細粒度のアクセス制御機構に関してはセキュア OS と呼ばれる OS での実装が行われており、一般的なものになりつつある。しかし、これらのアクセス制御機構は単一システムでの利用を想定しているものであり、分散システムでの利用に関してはシステムの特性が大きく異なるため、問題がある。例えば、分散システムは情報の伝達経路にネットワークを利用するが、これは飽和する可能性があるので他の様々な問題を引き起こし得る。また、ネットワークを利用した通信は単一システム内でのデータ交換に比べて秘密を守るのが難しい。そのため、現状の分散システムにおいては粒度の粗いアクセス制御が行われている。

そこで、本研究では、必要十分なアクセス権限付与を行う新しいカーネルと、それに基づいた分散システムにおけるアクセス制御モデルを提案する。提案モデルは、システムの各構成要素に関するセキュリティ耐性を向上するものではなく、システムを権限に関して適切に区画化することにより全体としての強度向上を期待するものである。必要最低限のアクセス権限を付与するためには、細かい粒度のアクセス制御が必要であるが、分散システムでは制御すべき情報が膨大であるため、それら全て正しく把握し、制御するのが困難である。そのため従来の仕組みでは、アクセス制御情報の大雑把なグループ化とそれに基づいた大雑把なアクセス制御を実施している。そこで本研究においては、アクセス主体を全て把握することの困難性に着目し、Capability と呼ばれるチケットを利用することによって、分散システムにおける細粒度アクセス制御を目指す。

## 3. 設 計 要 件

本研究においては、分散システムにおける細粒度アクセス制御の実現を第一の目標とする。これは PLP に則した情報システムを構築するためであり、そのために提案モデルは一般的な

認可システムとしての応用層からの利用も考慮する。この目標のために、本稿ではまず想定する分散システムの特徴を明確にした上で、その特徴から分散システムにおける細粒度アクセス制御システムの機能要件を定義する。

分散システムにおいて現在一般的に利用されているアクセス制御モデルは、アクセス主体と資源の関係性に関して、その認可情報をあらかじめ静的に定義できることを期待しているため柔軟性に乏しく粒度も粗い。しかし、分散システムにおいては潜在的に無制限で流動的なアクセス主体、資源、その利用方法が存在し得るため、認可情報を動的に扱える方が望ましい。また、アクセス制御の粒度に関しては、現状のような計算機毎やプロセス毎といった単位では十分ではなく、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を管理できる必要がある [16]。そこで本研究において想定する分散システムは、柔軟性の高い認可情報の扱いが可能であり、アクセス制御の粒度を細かく保つためにプロセス内で無数に生成／削除され得る実体を管理できるものとする。そのため、認可情報を上位の認める範囲内で制限可能なものとし、子プロセスや生成／削除する実体に委譲できるような仕組みを用いるものとする。

これらの要求に応えるために、提案モデルにおいては以下を設計要件とする。

**参照情報と認可情報の不可分：** 参照情報と認可情報は不可分に扱う。ACL のように、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 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、アクセス主体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。

**動的なアクセス主体の生成：** 認可情報をアクセス主体に関連づける。ACL においては、アクセス主体に関する情報を別に扱うのでアクセス主体が頻繁に変化する場合は整合性を保つのが難しい。アクセス主体が変化する頻度は粒度に依存することになるので、ACL ではプロセスをユーザアカウントなどの粗い実体にマップし、通常のオペレーションではあまり変化しないことを想定している。認可情報をアクセス主体毎に関連付ければ、アクセス主体が頻繁に変化しても問題にならず、プロセス自身やプロセスの生成するより細かな実体を扱うことができる。

**アクセス主体の認可情報委譲：** 新しく生成／削除される実体に対して、アクセス主体が認可情報を委譲できるようにする。ACL においては、認可情報に対する編集能力が資源毎に関連付けられており、ひとつのアクセス主体に対する認可情報の編集能力は通常 ACL 全体の編集能力となる。アクセス制御の柔軟性を保つためにはアクセス主体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。

**動的な資源の生成：** 動的に生成／削除される資源に対してアクセス制御を行う。あらゆる情報システムは、資源の生成と削除を繰り返しながら運用されるので動的な資源に対してのアクセスを制限できる必要がある。

## 4. 提案システム概要

### 4.1 想定システム構成

提案システムの実装方式に関しては、既存の OS に対する拡張を想定している。通常 OS には、アクセス制御以外にも多くの機能があるので、その全てをはじめから実装していくのは現実的ではない。

本提案が想定する分散システムは、構造として同一の資源アクセス制御機能を持つ OS の集合として構成され、それぞれの個別システムは分散システムにおける資源アクセス制御に必要な情報を保持するための保護された記憶領域を持つものとする。保護された記憶領域は TCB Boundary を形成するものであり、提案する資源アクセス制御に必要なコードとデータ、加えてこれを実装する OS はこの中で保護される。

アクセス主体による対象資源へのアクセスは、全て各個別システム内の OS によって提供されるシステムコールを利用して行われるものとし、アクセス主体はあらかじめ OS からシステムコールとして提供される手続きに引数のみを渡すことが可能である。結果としてあらゆる資源アクセスは OS によって捕捉され、提案する資源アクセス制御機能によって検査される。

### 4.2 提案する資源アクセス制御システムの構造

提案システムは分散システムにおいて細粒度アクセス制御を行うものであり、アクセス制御表を Service をアクセス制御対象とした Capability によって記述する点に特徴がある。

Service は実際にアクセスされる資源を機能毎にグループ化した Object であり、資源の詳細情報を隠蔽する。分散システムにおいてはアクセス制御の対象となる資源数が構成システム数に応じて増加するが、Service Object としてまとめることにより制御表を簡単にできる。

Capability は Service Object の Location と Access Mode を記述したデータ構造であり、システムの保護領域においてプロセス毎に関連付けられる。Capability をローカルシステムに保持することにより、分散システム内のあらゆる Object に対する認可問い合わせをローカルシステム内で完結できる。

提案システムは ISO10181-3 をベースに拡張されており、コンポーネントの種別は可能な限り踏襲されている。それぞれ、Initiator は資源へのアクセス要求を最初に発行するシステム内の実体を示し、Target はそのアクセス対象となる実体を示す。また、Access Enforcement Function(AEF) は、Initiator から Target へのアクセス要求 (Access Request) を仲介することによってアクセス制御を強制するためのコンポーネント、Access Decision Function(ADF) は AEF からの認可決定要求 (Decision Request) を受け取るコンポーネントを示す。図 1 において、ISO10181-3 アーキテクチャの概要を示す。

## 5. 提案システムの構成

### 5.1 Architectural Overview

提案システムのアーキテクチャを図 2 に示す。本アーキテクチャは分散システムを構成しており、ISO10181-3 に Environment と Agent をコンポーネントとして加えたものである。

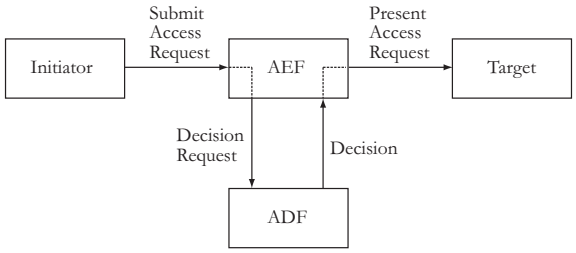


図 1　ISO10181-3 Access Control Framework

Initiator、Target、AEF、ADF の基本的な役割は ISO10181-3 と同等であり、Environment は Initiator の Capability 情報と Target の Location 情報を保持する。また、Agent はリモートシステムの Agent に対する安全な情報伝達を担うものであり、またリモートシステムにおいては Target に対して Initiator の代理アクセスを行う。

図 2 は $System_A$ の Initiator によって生成された Access Request が $System_B$ の Target に到達するまでの手順を含んでおり、以下にその概要を示す。

**Step 1** Initiator は $AEF_A$ に Access Request を行う。

**Step 2** $AEF_A$ は Initiator から Access Request を受け取り、$ADF_A$ に Decision Request を行う。

**Step 3** $ADF_A$ は $AEF_A$ から Decision Request を受け取り、$Env._A$ に保持されている情報を元に Decision を生成して $AEF_A$ に返す。

**Step 4** $AEF_A$ は $ADF_A$ から Decision を受け取り、$Env._A$ に保持されている情報を元に Service の Location を解決する。

**Step 5** $AEF_A$ は $Agent_A$ に Remote Access Request を行う。

**Step 6** $Agent_A$ は $AEF_A$ から Remote Access Request を受け取り、$Agent_B$ に Proxy Access Request を行う。

**Step 7** $Agent_B$ は $Agent_A$ から Proxy Access Request を受け取り、$AEF_B$ に Access Request を行う。

**Step 8** $AEF_B$ は $Agent_B$ から Access Request を受け取り、$ADF_B$ に Decision Request を行う。

**Step 9** $ADF_B$ は $AEF_B$ から Decision Request を受け取り、$Env._B$ に保持されている情報を元に Decision を生成して $AEF_B$ に返す。

**Step 10** $AEF_B$ は $ADF_B$ から Decision を受け取り、$Env._B$ に保持されている情報を元に Service の Location を解決する。

**Step 11** $AEF_B$ は Target に Access Request を行う。

### 5.2　Initiator

Initiator は Access Request を発行するプロセスであり、希望する Service Name と Operation Name を知っている。Initiator は AEF に Service Name と Operation Name を含む Access Request を行い、Location の解決と認可決定を委ねる。Access Request が認可された場合は、AEF から Access Responce を受け取る。

### 5.3　Target

提案システムの Target は、Service として機能毎に資源をグループ化した Object である。Target は AEF から Access Request を受け取り、Responce を返す。Target は複数の資源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他システムからの利用が可能となる。Service Object の詳細な構成と受け入れ可能な Operation は、Service 毎に規定される。

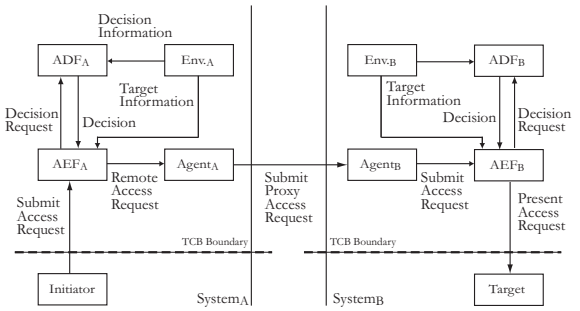


図 2　Architectural Overview

### 5.4　Operation

Operation は Target に対して要求する操作種別である。Operation は Service Object 毎にインターフェースとしてあらかじめ提供されるものであり、Access Request に対して Responce を返す。

### 5.5　Access Enforcement Function

AEF はアクセス制御対象となる資源へのアクセスを仲介するコンポーネントであり、ADF による認可決定を Initiator に強制する。

AEF は TCB Boundary の内側にあり、外側に対して定型化された手続きを提供する。外側のアクセス主体は、AEF から提供される手続きに変数を渡すことによってのみ資源にアクセス可能であり、AEF は資源への実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 Capability を元に Target の Location 解決を行い、Location がリモートシステムであった場合は Agent に代理アクセスを依頼する。Location がローカルシステムであった場合は、システムファンクションを利用して実際に資源へのアクセスを行う。

以下に AEF の動作手順を示す。

**Step 1** AEF は Initiator から Access Request [Initiator Name : Service Name : Operation Name] を受け取る。

**Step 2** AEF は ADF に Decision Request [Initiator Name : Service Name : Operation Name] を行う。

**Step 3** AEF は ADF から Decision [Capability or Reject] を受け取る。

**Step 4** AEF は Capability の Directory Index と Service Table Directory から Service の Location を特定する。

**Step 5** AEF は Capability と Location から [Location : Table Index : Operation Name] を生成する。

1　[Location → Local]

（図 3 参照）

**Step 6** AEF は Service Table と Table Index から Service Object を特定する 。

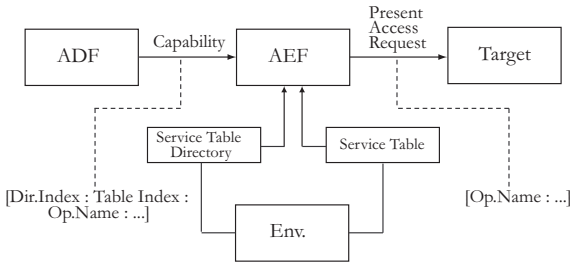


図 3　AEF Local Access

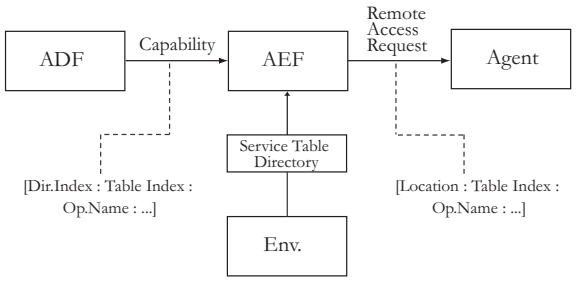


図 4　AEF Remote Access

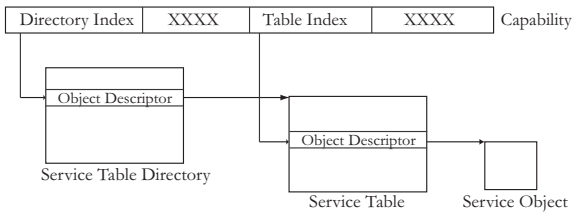


図 5　Service Location Transition

**Step 7** AEF は Service Object に対して Operation を実行し、Responce を Initiator に返す。

2　[Location → Remote]

（図 4 参照）

**Step 6** AEF は Agent に [Location : Table Index : Operation] を渡して Remote Access を依頼する。

**Step 7** AEF は Agent から Responce を受け取り、Initiator に返す。

図 5 に Capability から Service Object を特定する様子を示す。AEF は以下の手順に従って、Service Object を特定する。

**Step 1** AEF は ADF から Capability [Directory Index : Table Index : Operation Name : ...] を受け取る。

**Step 2** AEF は Directory Index を元に、Environment に保持されている Service Table Directory から特定の Service Table Object に関する Object Descriptor を抽出する。

**Step 3** AEF は Object Descriptor を元に、Service Table を特定する。

**Step 4** AEF は Table Index を元に、Service Table から特定の Service Object に関する Object Descriptor を抽出する。

**Step 5** AEF は Object Descriptor を元に、Service Object を特定する。

### 5.6　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

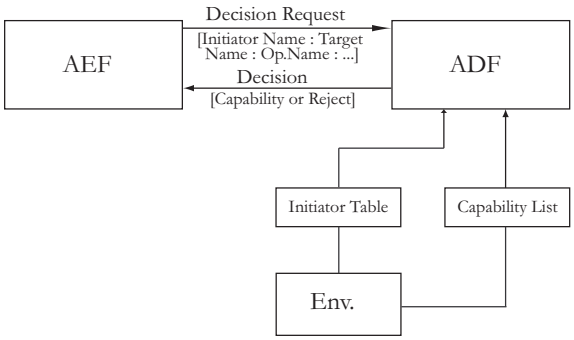


図 6　Inputs and Outputs to ADF

の Decision Request に対して Decision を返す。ADF は TCB Boundary の内側にあり、AEF と Environment から認可決定に必要な情報を収集する。

以下に ADF の動作手順を示し、図 6 に ADF に入出力される情報の概略を示す。

**Step 1** ADF は AEF から Decision Request [Initiator Name : Service Name : Operation Name] を受け取る。

**Step 2** ADF は Initiator Name を元に、Environment に保持されている Initiator Table から特定の Initiator Descriptor を抽出する。

**Step 3** ADF は Initiator Descriptor から Initiator Name に関連付けられた Capability List(C-List) を特定する。

**Step 4** ADF は C-List の各 Capability を検索し、Service Name と Operation Name を含む Capability を特定する。

**Step 5** ADF は Capability を特定した後に Decision として [Capability] を返すが、必要が Capability が見つからなかった場合は Decision として [Reject] を返す。

### 5.7　Agent

Agent は分散システムを構成する各システムに存在し、自システム以外のシステムに存在する Agent と通信を行う。各 Agent はあらかじめ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 は自システムの AEF から外部システムに存在する Service Object へのアクセス要求を受け取り、他システムの 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、あらためて Initiator として自システムの AEF に Service Name と Operation Name によるアクセス要求を行い、Responce を元の Initiator に返す。

以下に Agent の特徴を示し、図 7 にその概略を示す。

- Agent は各システムに存在する外部アクセスを行うシステムプロセスであり、各 Agent 間はあらかじめ互いを識別することができる。
- Agent は Local AEF から Remote AEF まで安全な情報伝達経路を保証する。
- Local Agent は Local AEF から Remote Access Request を受け取り、Remote Agent に中継する。
- Local Agent から Remote Access Request を受け取った Remote Agent は、Remote Host で Target への Proxy Access を行う。

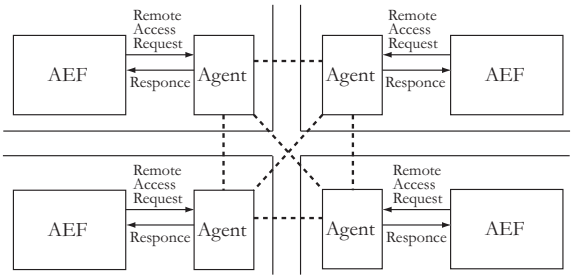


図 7　Communication between Agents

- Remote Host で AEF から Responce を受け取った Remote Agent は、Local Agent へ中継する。
- Remote Agent から Responce を受け取った Local Agent は、Local AEF へ Responce を中継する。

### 5.8　Environment

Environment は TCB Boundary の内側で保持されるシステムに関する属性情報の集合である。Environment は分散システムを構成する各ホスト毎に構成され、アクセス制御を行う際に各コンポーネントによって利用される。

#### 5.8.1　Initiator Table

Initiator Table は各 Initiator 対応にあるエントリ (Initiator Descriptor) を管理する表である。各エントリは Initiator Name、Initiator ID、Initiator Group 等の Initiator に関する属性情報を記述したデータ構造であり、アクセス制御の際には ADF によって Initiator Name を Key に検索される。Initiator Descriptor には C-List が格納された記憶領域へのポインタが含まれており、ADF による Capability の特定に利用される。

以下に Initiator Table の特徴を示す。

- Initiator Table はシステム内の各 Initiator の属性情報を管理する表である。
- Initiator Table は分散システムを構成する各ホスト毎に管理される。
- Initiator Table は ADF によって参照される。
- Initiator Table は C-List へのポインタを含んでいる。

#### 5.8.2　Service Table Directory

Service Table Directory は、各 Service Table Object 対応にあるエントリ (Object Descriptor) を管理する。各エントリは、Length や IP Address 等の Service Table Object に関する属性情報を記述したデータ構造であり、アクセス制御の際には AEF によって利用される。Service Table Directory は、あらかじめ Environment に保持された Base Address によって AEF に指定され、Capability による Service Table Object の特定に利用される。

以下に Service Table Directory の特徴を示す。

- Service Table Directory は Environment 内に保持され、あらかじめ設定された Base Address を元に指定される。
- Service Table Directory は Service Table Object に関する属性情報の集合である。
- Service Table Directory は AEF によって Service Table の Location 解決に利用される。
- Service Table Directory はひとつだけ存在し、分散システムを構成する各ホスト間で共有される。

#### 5.8.3　Service Table

Service Table は、各 Service Object 対応にあるエントリ (Object Descriptor) を管理する。各エントリは、Length や IP Address 等の Service Object に関する属性情報を記述したデータ構造であり、アクセス制御の際には AEF によって利用される。Service Table は、Object Descriptor に記述された Base Address によって AEF に指定され、Capability による Service Object の特定される。

以下に Service Table の特徴を示す。

- Service Table は Service Table Directory の Object Desciptor に記述された Base Address によって指定される。
- Service Table は Service Object に関する属性情報の集合である。
- Service Table は AEF によって Service Object の Location 解決に利用される。
- Service Table は分散システムを構成する各ホスト毎に管理される。

#### 5.8.4　Capability

Capability は、Service Object の Location と Operation を指定するものである。Initiator は、Capability を通じて Service Object の Location と Operation を指定するため、ある Initiator に関連付けられた Capability の集合はその Initiator が利用可能な Service Set を規定するものである。Capability は Service Name、Operation Name、Directory Index、Table Index を含むものであり、Service Object の認可決定の際は ADF に参照される。また、Service Object の Location 解決の際には AEF に参照される。

以下に Capability の特徴を示す。

- Capability は Environment 内に保持される Initiator に関する属性情報のひとつである。
- Capability は Initiator Table と C-List によって各 Initiator に関連付けられて保持される。
- Capability は Service Name、Operation Name、Directory Index、Table Index を含む。
- Capability は ADF によって Initiator の認可決定に利用される。
- Capability は AEF によって Service Object の Location 解決に利用される。

#### 5.8.5　Capability List

C-List は Environment に保持され、Initiator Table において Initiator 毎に関連付けられる。C-List は Initiator が保持する Capability の集合であり、各 Initiator の Initiator Descriptor からポイントされる。

以下に C-List の特徴を示す。

- C-List は Environment 内に保持される。
- C-List は Initiator Table において各 Initiator に関連付けられる。
- C-List は Initiator Descriptor からポイントされ、AEF

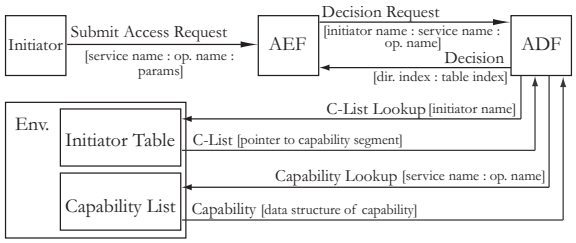


図 8　Decision Making Procedure

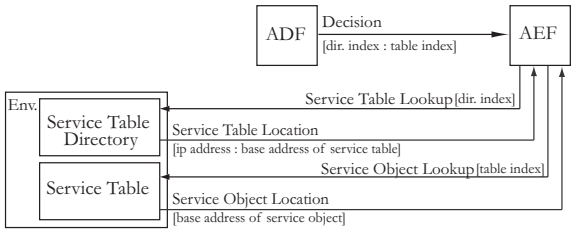


図 9　Location Looking up Procedure

と ADF によって参照される。

- C-List はある Initiator が保持する Capability の集合である。

### 5.9　処理の流れ

提案システムにおいては、分散システム上で細粒度アクセス制御を実現するために ADF において 5.6 に示した Capability チェックが実施される。この Capability チェックは、すべて ADF が提供するインターフェースを利用して行われるものであり、Initiator が直接このインターフェースを呼び出すことはできない。

図 8 に Capability チェック時における各コンポーネント間の連携の様子を示す。

Service の Location 解決は Capability から AEF が行い、Location が Local であった場合は、システムファンクションを利用して AEF が Service Object へのアクセスを行う。Location が Remote であった場合は、Agent が提供するインターフェースを呼び出し Remote Access を依頼する。

図 9 に Location 解決時における各コンポーネント間の連携の様子を示す。

## 6.　提案システムの評価

**資源情報の保持：** 提案システムにおいては、資源情報を全体として保持する共通領域に一括して置くのではなく、Capability に含んで各個別システムに分配する。Capability は各個別システム毎に OS の一部として保持されるので、不正な改竄を防止することが可能である。また、各個別システム内で保持される Capability は、そのシステム内のアクセス主体が分散システムのポリシーによってアクセスを許可されたものだけであるので、結果として各個別システムはシステム内の全アクセス主体がアクセス可能な資源情報のみを保持することとなった。

**資源の参照方法：** 提案システムにおいては、Service Table Directory と Service Table を利用することにより、分散システム全体の資源に対して一貫した資源参照を行うことが可能となった。この 2 つのテーブルは、OS の一部として保持されるので不正な改竄を防止することが可能であり、同じく保護された Capability に含まれるインデックスを経由して利用される。

**アクセス制御の粒度：** 提案システムにおいては、アクセス主体に提供する機能を基本単位とした Service に対象資源のグループ化を行い、その粒度においてアクセス制御を行うことが可能である。また、アクセス主体に関しては、それぞれのアクセス主体毎に Capability の有無が制御情報となるので、結果としてグループ化は行わないものとなった。提案システムは通常の UNIX より粒度の細かい資源アクセス制御が可能であり、Security-Enhanced Linux よりは粒度の粗い資源アクセス制御を行うものである。

## 7.　今後の課題

本研究では、分散システムにおける細粒度アクセス制御に関して、Capability-Based System の手法を取り入れた仕組みを検討した。分散システムも Capability-Based System も共に古くから研究されてきたテーマであり、既に多くが確立されている分野であるが、昨今の技術革新とセキュリティへの要望の高まりを背景に融合・再考を試みた。

提案システムでは、Capability を利用した分散システムのアクセス制御メカニズムを示したが、今後はポリシーから各種制御表を生成する仕組みと 5. で示した Agent に関して、Agent 間の安全な通信手法の詳細化が問題である。今後はこれらを課題とするが、特に重要な点を以下に示す。

**表とリストの初期化／変更方法：** Service Table Directory、Service Table、C-List の正しい初期化は、提案システムが期待通りに動作するための前提となるものである。提案システムにおいては認可決定をローカルシステムに保持された Capability を参照をすることにより判断するものであるが、これには C-List があらかじめ正しく初期化されている必要がある。資源参照の際の Location 解決においても、Service Table Directory と Service Table があらかじめ正しく初期化されている必要があるので、これらの各データを初期化する仕組みは非常に重要なものである。

また、提案システムにおいては、Service Table Directory、Service Table、C-List によって認可情報と Location 情報が静的に各システムにおいて保持されるが、これらの情報が一定に固定され続けるような情報システムは考えにくいので、必然的に変更方法も検討する必要がある。

**Agent 間通信の動作メカニズムとプロトコル：** Agent 間通信は、提案システムにおいて各構成システム間の相互識別と安全な情報伝達を担うものであり、Capability による認可決定と Location 解決を確実に実施するための鍵となる。提案システムにおいては、あるシステムの ADF による認可決定が Agent を通じて他システム内で実施されるので、Agent 間通信が信頼できない限りは成り立たないものである。

また、以下にその他の検討項目を列挙する。

- Initiator Table と Initiator Descriptor の詳細な内容
- Service Table Directory と Service Table、Object De-

scriptor の詳細な内容

- Capability と C-List の詳細な内容

## 文　献


[1] Peter J. Denning, “Fault Tolerant Operating Systems”, ACM Computing Surveys (CSUR), v.8 n.4, p.359-389, December 1976.

[2] N. Hardy. The KeyKOS Architecture. ACM Operating Systems Review, September 1985, p. 8 - 25. http://www.agorics.com/Library/KeyKos/architecture.html

[3] N. Hardy. “The Confused Deputy (or why capabilities might have been invented)”. Operating Systems Review 22(4), October 1988, pp. 36 - 38.

[4] 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.

[5] J. B. Dennis and E. C. Van Horn, “Programming Semantics for Multiprogrammed Computations”, Comm. ACM, 1966.

[6] Jerome H. Saltzer, “Protection and the control of information sharing in Multics”, Comm. ACM, 1974.

[7] Ray Spencer, Stephen Smalley, Peter Loscocco, Mike Hibler, David Andersen, Jay Lepreau, “The Flask Security Architecture: System Support for Diverse Security Policies,” USENIX, August 1999.

[8] “Security-Enhanced Linux”, NSA. http://www.nsa.gov/selinux/

[9] A. Keromytis, S. Ioannidis, M. Greenwald, and J. Smith, “The strongman architecture,” In Third DARPA Information Survivability Conference and Exposition (DISCEX III), April 2003.

[10] Matt Blaze, Joan Feigenbaum, Martin Strauss, “Compliance Checking in the PolicyMaker Trust Management System,” Financial Cryptography 1998, Anguila, 1998.

[11] Matt Blaze, Joan Feigenbaum, Angelos D. Keromytis, “KeyNote: Trust Management for Public-Key Infrastructures,” In Proceedings of the 1998 Security Protocols International Workshop, Springer LNCS vol. 1550, pp. 59 - 63. April 1998, Cambridge, England. Also AT&T Technical Report 98.11.1.

[12] “AppArmor”, Novell, Inc. http://en.opensuse.org/Apparmor

[13] ”TrustedBSD home page,“TrustedBSD Project. http://www.TrustedBSD.org/

[14] M. Y. Becker and P. Sewell. “Cassandra: Distributed access control policies with tunable expressiveness”. In 5th IEEE International Workshop on Policies for Distributed Systems and Networks, 2004.

[15] E. Bertino, E. Ferrari, and A. C. Squicciarini. “Trust-X: A peer-to-peer framework for trust establishment”. IEEE Transactions on Knowledge and Data Engineering, 16(7):827 - 842, Jul. 2004.

[16] 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.