システムソフトウェアと
オペレーティング・システム 87－16
（２００１．６．２９）

Webブラウザのための安全なプログラム実行環境の実現

品川 高廣 †　　河野 健二 ††,†††　　益田 隆司 ††

† 東京大学大学院 理学系研究科 情報科学専攻
†† 電気通信大学 情報工学科，††† 科学技術振興事業団さきがけ研究 21
電子メール：shina@is.s.u-tokyo.ac.jp，{kono,masuda}@cs.uec.ac.jp

要旨

最近の Web ブラウザでは，モバイルコードを組み込むことによる機能拡張が一般的に行なわれている．しかし従来の OS では，モバイルコードに対して Web ブラウザ本体と同等のアクセス権限が与えられてしまうため，不正アクセスの危険があった．本論文では，モバイルコードを Web ブラウザ内で安全に実行するために，モバイルコードのアクセス制御を実現する手法を提案する．この手法では，我々が提案・開発中の SeeMoc オペレーティングシステムで提供する細粒度保護ドメインを利用している．実験によって，この手法による保護のオーバーヘッドは 12.9% 程度に抑えられることを確認した．

A Secure Execution Environment for Web-Browsers

Takahiro Shinagawa†　　Kenji Kono††,†††　　Takashi Masuda††

†Department of Information Science, Graduate School of Science, University of Tokyo
††Department of Computer Science, University of Electro-Communications,
†††PREST, Japan Science Technology Corporation
E-mail: shina@is.s.u-tokyo.ac.jp, {kono,masuda}@cs.uec.ac.jp

Abstract

Recent web-browsers can execute mobile code, such as *plug-ins*, which is downloaded from the Internet. However, the mobile code is dangerous because a malicious mobile code may gain unauthorized access to the user's computer. To protect against the malicious mobile code, we have developed the SeeMoc operating system, which supports fine-grained protection domains. This paper presents a secure execution environment for mobile code in the web-browser using fine-grained protection domains. Experimental results show that the average overhead of cross-domain calls is 12.9%.

1 はじめに

Web ブラウザでは，次々と開発される新しい規格の Web コンテンツに対応するために，モバイルコードを利用した機能拡張が一般的に行われている．モバイルコードとは，Web ブラウザに組み込まれて新しい機能を提供するソフトウェアコンポーネントのことである．ユーザは必要に応じてインターネットを用いてモバイルコードを入手することができる．例えば，Netscape Navigator では，プラグインと呼ばれる共有オブジェクト形式のモバイルコードを組み込むことによって，PDF や Shockwave など新しいファイル形式の Web コンテンツをいち早く閲覧することができるようになる．

しかし，モバイルコードはインターネットを経由して入手するので，ユーザのコンピュータが不正アクセスされる危険性がある．インターネットは匿名性が高いので，入手したモバイルコードは必ずしも信頼できるとは限らない．モバイルコードの中にはウィルスのように悪意を持っていて，ユーザのファイルを破壊したり，自身のコピーを電子メールで送りつけるなどの不正アクセスを行なうコードが含まれている可能性がある．

従来のオペレーティングシステムでのプロセスによる保護モデルでは，モバイルコードの不正アクセスを防止することは困難である．まず，保護ドメインの概念がプロセスと一体になっているので，Web ブラウザと同じプロセスで動作するモバイルコードに対しては，Web ブラウザ本体と同等のアクセス権限が与えられてしまう．また，保護ドメインの粒度が粗いので，特定のファイルへのアクセスを制限するなどのように，細かいアクセス制御を行なうことができない．

本論文では，モバイルコードを Web ブラウザ内で安全に実行するために，モバイルコードのアクセス制御を実現する手法を提案する．この手法では，モバイルコードに対して Web ブラウザ本体とは異なる保護ドメインを割り当て，その保護ドメインのアクセス権限を細かく制御することによって，不正なアクセスを防止する．また，既存の Web ブラウザを改変せずに保護を実現するために，モバイルコードの読み込み時に新しい保護ドメインを作成して割り当てたり，保護ドメインのアクセス権限を設定するなどの処理を行なうプラグインを開発した．実際に Netscape 用のプラグインを開発して実験を行った結果，保護によるオーバーヘッドは 12.9 % 程度に抑えられることを確認した．

この保護機構では，我々が提案・開発を行っている SeeMoc オペレーティングシステムの機能を利用している [14, 11]．SeeMoc では，１つのプロセス内に複数の保護ドメインを持つことが可能になっており，このプロセス内の保護ドメインを我々は細粒度保護ドメインと呼んでいる．細粒度保護ドメインのアクセス制御は，ユーザレベルで実装されるポリシーモジュールによって，柔軟に行なうことができる．また，細粒度保護ドメインの切り替えはプロセスよりも高速に行うことが可能であり，保護のオーバーヘッドを低く抑えられる．

以下2章では，SeeMoc でサポートする細粒度保護ドメインの保護モデルについて説明する．3章では，モバイルコードに細粒度保護ドメインを割り当てるプラグインの機構を説明する．4章では，細粒度保護ドメインで保護を行なったことによるオーバーヘッドについての実験結果を示す．5章で関連研究に触れ，6章で本論文をまとめる．

2 保護モデル

SeeMoc オペレーティングシステムでサポートする細粒度保護ドメインは，細粒度のアクセス制御，効率の良い資源共有，高速な保護ドメイン切り替えといった特徴を備えている．本章では，この細粒度保護ドメインによる保護モデルについて説明する．

2.1 細粒度保護ドメイン

細粒度保護ドメインは，一つのプロセスの中に複数個持つことができる保護ドメインである．保護ドメインとはアクセス可能な資源の集合を表す概念で

あり，従来のオペレーティングシステムでは，プロセスが保護ドメインの役目を兼ねている．細粒度保護ドメインはプロセスによる保護ドメインの部分集合として定義される．つまり，細粒度保護ドメインでアクセス可能な資源は，それが定義されたプロセスでアクセス可能な資源の中の一部になっている．

細粒度保護ドメインでは，アクセス制御を細かい単位で行なうことができる．例えば，メモリに対するアクセスは，ページ単位で細粒度保護ドメイン毎に異なる保護モードを設定することができる．また，ファイルやネットワークなどのシステム資源へのアクセスも任意の単位でアクセス権限を設定することができる．システム資源に対するアクセス制御の具体的な方式はオペレーティングシステムでは規定しておらず，プロセス毎にユーザレベルで実装されるポリシーモジュールと呼ぶコードにアクセス制御を委任する．従って，アプリケーションに合わせて様々な保護ポリシーを実現できる．

細粒度保護ドメインはプロセスの一部なので，細粒度保護ドメイン間の資源共有を効率よく行なうことができる．例えば，仮想アドレス空間を共有しているので，ポインタを含む複雑なデータ構造などでもメモリ上で容易に共有することができる．

細粒度保護ドメインはプロセスに割り当てられた資源を共有するので，保護ドメインの切り替えを高速に行なうことができる．例えば，ページテーブルを共有するので，保護ドメイン切り替えの際に TLB フラッシュに伴うコストを避けることができる．また，タイムスライスを共有するので，保護ドメイン切り替えの際のスケジューリングが不要である．

2.2 保護の実現

本説では，SeeMoc カーネルにおいて細粒度保護ドメインを実現するための機構について説明する．なお，具体的な方法については文献 [11, 14] を参照されたい．

2.2.1 細粒度のアクセス制御

システムの資源に対して細かい単位でアクセス制御を行なうために，SeeMoc カーネルではシステムコールを横取りする機構を用意している．細粒度保護ドメインが割り当てられたモバイルコードがシステムコールを発行すると，予め登録されたユーザレベルで動作するポリシーモジュールが呼び出される．ポリシーモジュール自体も細粒度保護ドメインによって保護されており，通常は Web ブラウザ本体の細粒度保護ドメインが割り当てられる．

ポリシーモジュールは自身の保護ポリシーに基づいて，横取りしたシステムコールの扱いを判断する．SeeMoc カーネルはポリシーモジュールを呼び出す際に，呼び出し元のコードに割り当てられた細粒度保護ドメインを識別する ID と，システムコールの種類，及びその引数が渡される．ポリシーモジュールは，この ID とシステムコールの内容をチェックして，そのシステムコールの発行を許可，不許可を決定する．

ポリシーモジュールの呼び出しは，細粒度保護ドメイン間呼び出しで行なわれるので，横取りに伴うオーバーヘッドは低く抑えられる．また，ポリシーモジュールの呼び出し頻度を減らすために，`getpid()` などのシステムコールは無条件で許可するといった設定をすることもできる．

2.2.2 細粒度のメモリ保護

プロセス内でページ単位のメモリ保護を実現するために，SeeMoc カーネルではマルチプロテクションページテーブルという抽象概念を導入している [11, 14]．マルチプロテクションページテーブルとは，従来のページテーブルを拡張して，全てのページの保護モードを一度に切り替えられるようにしたものである．図1 上部に示すように，ページテーブルの各エントリには，仮想アドレスから物理アドレスへのマッピングに加えて，ページ保護モードを複数個設定できるようになっている．ページ保護モードの列がそれぞれ一つの細粒度保護ドメインに対応

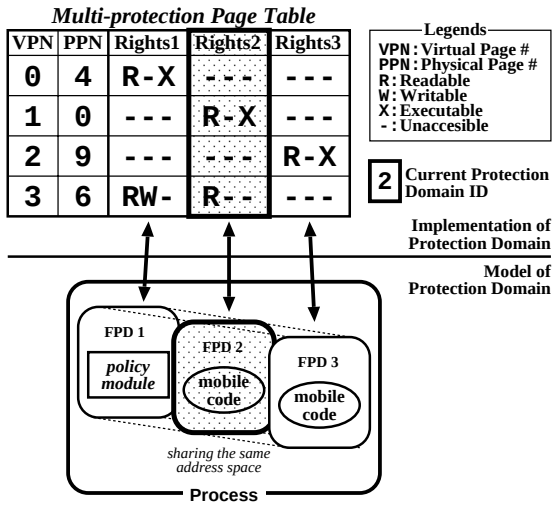

図 1: マルチプロテクションページテーブル

している．ある時点で有効な保護モードの列は一つだけであり，これを変更することによって保護ドメインを切り替えることができる．

マルチプロテクションページテーブルを用いると，細粒度のメモリ保護と同時に，効率の良い資源共有や高速な保護ドメイン切り替えも容易に実現できる．例えば細粒度保護ドメイン毎に異なるページ保護モードを設定しつつもアドレス空間を共有できるので，ポインタを含む複雑なデータ構造などの共有が容易に行なえる．また保護ドメイン切り替えの際にページテーブルを切り替える必要がないので，TLB フラッシュに伴うオーバーヘッドを避けて，高速な保護ドメイン切り替えが実現できる．

2.2.3 高速な保護ドメイン切り替え

高速な保護ドメイン切り替えを実現するために，SeeMoc カーネルは専用のシステムコールを用意している．細粒度保護ドメイン間呼び出しを行なうときには，呼び出し先の細粒度保護ドメインを識別する ID を引数として，このシステムコールを発行する．このシステムコールの中では，システムコールを呼び出したスレッドに割り当てられている細粒度保護ドメインの ID を，引数で指定した細粒度保護ドメインの ID に書き換える．そして，呼び出し元の細粒度保護ドメインの ID を引数として，予め細粒度保護ドメイン毎に登録されているエントリポイントに制御を渡す．

保護ドメイン切り替えに必要な処理は，本質的にはスレッドに割り当てられている細粒度保護ドメインの ID を書き換えるだけであり，高速な保護ドメイン切り替えが実現できる．

3 安全な実行環境の実現

本章では，Netscape のプラグインを利用して，モバイルコードの安全な実行環境を実現する手法について説明する．まず，SeeMoc オペレーティングシステムで細粒度保護ドメインを扱うためのライブラリ関数について説明する．次に，Netscape のプラグインの仕組みについて簡単に説明し，このプラグインを利用してモバイルコードに細粒度保護ドメインを割り当てる機構について説明する．

3.1 細粒度保護ドメインのライブラリ関数

SeeMoc オペレーティングシステムでは，細粒度保護ドメインを容易に扱えるようにするために以下のようなライブラリ関数を用意している．

`domain_create();` 細粒度保護ドメインの作成と初期化を行なう．ELF 形式のモバイルコードをファイルからメモリに読み込み，SeeMoc カーネルが提供するシステムコールを発行して，新しい細粒度保護ドメインを作成して割り当てる．デフォルトの保護ポリシーでは，予め定義された共有領域と自身のメモリ領域以外へのアクセスは禁止される．システムコールの発行も全て禁止される．

`domain_setpolicy();` 細粒度保護ドメインの保護ポリシーを設定する．メモリのページ保護

モードや，システムコール発行の許可・不許可などの設定を行なう．

`domain_call();` 呼び出し先の保護ドメインを指定して，保護ドメイン間呼び出しを行なう．

`domain_destroy();` 細粒度保護ドメインを破棄する．

なお，現在我々が開発中の SeeMoc カーネルは Intel の IA-32 と SPARC 上で動作している．カーネル内の実装の詳細については，文献 [11, 14, 15] を参照されたい．

3.2 Netscape プラグインの仕様

Netscape プラグインはプラットフォーム依存の機械語コードであり，共有ライブラリと同様のオブジェクトファイルで提供される．プラグインは C 言語や C++ 言語などで開発され，オペレーティングシステムの機能を利用して，ウィンドウ内の描画やユーザとの対話的処理などを行なう．

Netscape のプラグインは Web コンテンツの MIME タイプで分類されており，あらかじめ Netscape に登録されている．Netscape は起動時に，環境変数で指定されたディレクトリにあるプラグインを自動的に読み込んでリンクする．Netscape は自分が解釈できない MIME タイプの Web コンテンツを読み込むと，指定されたプラグインを呼び出してウィンドウへの描画やキーイベントの処理などを依頼する．

Netscape とプラグインの間には，いくつかのインターフェイスが定義されている．プラグインが提供する API としては，以下のようなものがある．

`NPP_Initialize();` プラグインの初期化を行なう．Netscape の起動時などに呼び出される．

`NPP_New();` 新しく実体（インスタンス）を作る．Web コンテンツを読み込む度に呼び出される．

`NPP_Destroy();` 実体を削除する．別のページに移動するときなどに呼び出される．

`NPP_Shutdown();` プラグインの終了処理をする．

`NPP_SetWindow();` プラグインが描画を行なうウィンドウについての情報を提供する．ウィンドウの作成，移動，サイズ変更，削除などが行なわれた際に呼び出される．実際の描画などの処理は，プラットフォームに依存した方式で行なわれる．

`NPP_StreamAsFile();` プラグインが表示するべき Web コンテンツをファイルとしてアクセスする． Web コンテンツのダウンロードは Netscape が行ない，それが格納されているテンポラリファイルのパス名が渡される．

3.3 モバイルコードの実行環境の実現

我々が開発したプラグインは，モバイルコードを Web からダウンロードして，ブラウザに組み込んで細粒度保護ドメインを割り当てる処理を行なう．モバイルコードは，ELF 形式のオブジェクトファイルであり，Web 上に置く．モバイルコードの MIME タイプとしては application/x-seemoc を定義した．

ユーザがモバイルコードの URL を指定すると，Netscape はまずモバイルコードの新しい実体を作成するために `NPP_New()` を呼び出す．またプラグインが描画するべきウィンドウの情報を伝えるために `NPP_SetWindow()` を呼び出す．この時プラグインは，このウィンドウに対してイベントハンドラの登録などを行なう．次に Netscape はそのファイルをダウンロードして，プラグインの `NPP_StreamAsFile()` を呼び出す．プラグインはこのファイルのパス名を引数として SeeMoc の `domain_create()` を呼び出して，モバイルコードに細粒度保護ドメインを割り当て，実行できるようにする．

モバイルコードとプラグインは，予め定義したインターフェイスを通じて通信を行なう．モバイルコードとプラグインはそれぞれ異なる細粒度保護ドメインが割り当てられているので，それぞれの API は `domain_call()` を用いて呼び出す．

3.4 インターフェイスの例

モバイルコードの動作実験を行なうために，我々は簡単な実験用のインターフェイスを定義した．プラグインとモバイルコードの間のインターフェイスは，想定するモバイルコードの機能によって様々なものが考えられる．しかし，このインターフェイスは保護ドメインをまたいで呼び出されるため，その設計に当たってはセキュリティ上の欠陥が生じないように行なう必要がある．

今回使用するモバイルコードとしては，実験用として簡単な絵を描くだけのモバイルコードを想定した．ユーザがマウスをクリックすると，モバイルコードを呼び出して絵を描画する処理を行なう．モバイルコードが提供する API としては，`drawWindow()` という関数を定義した．プラグインはマウスがクリックされたというイベントを受け取ると，ウィンドウに絵を描画するために保護ドメイン間呼び出しでこの API を呼び出す．プラグインがモバイルコードに提供する API としては `PutPixel()` という関数を定義した．モバイルコードはウィンドウの座標と色を指定してこの API を呼び出すことによって絵を描画する．

セキュリティポリシーとしては，モバイルコードは `PutPixel()` 以外の API は一切呼び出すことができないようにした．また，システムコールの発行も全て禁止した．

4 性能実験

3章で示した手法で保護を行なった場合のオーバーヘッドを計測する実験を行なった．まず保護ドメイン間呼び出し自体にかかる時間を計測する実験を行なって，細粒度保護ドメインの基本性能を測定した．次に簡単な絵を描くモバイルコードを動作させて，実際のアプリケーションにおける性能を測定した．

実験に使用したマシンは，CPU として PentiumIII 1GHz，メモリを 128M バイト搭載した PC である．使用したオペレーティングシステムは，我々が Linux 2.2.18 を改造して細粒度保護ドメインの機構を組み込んだ SeeMoc カーネルバージョン 0.4 である．

4.1 保護ドメイン間呼び出し

SeeMoc の細粒度保護ドメインにおいて，保護ドメイン間呼び出しにかかるサイクル数を測定する実験を行なった．測定には PentiumIII に搭載されているサイクルカウンタを利用した．実験には保護ドメイン間呼び出しで何もしない手続きを呼び出して，その前後のサイクルカウンタの値の差を求めた．比較実験として，別プロセスの手続きを IPC (パイプ) で呼び出す場合にかかるサイクル数も計測した．表 1に実験結果を示す．

表 1: 保護ドメイン間呼び出しにかかるサイクル数

方式	サイクル数	時間
SeeMoc	192	0.19μs
IPC (パイプ)	4,046	4.05μs

SeeMoc の保護ドメイン間呼び出しは IPC に比べると約 21 倍速く，高速な保護ドメイン間呼び出しが実現されていることがわかる．

4.2 描画モバイルコード

モバイルコードのオーバーヘッドを測定するために，我々は簡単な絵を描くモバイルコードを作成した．このモバイルコードはフラクタル図形の一種であるマンデルブロ集合を描くものである．モバイルコードのインターフェイスには，3.4節で定義したものを使用した．

実験では，正方形のマンデルブロ集合を描画させ，一辺の長さを 1 ピクセルから 150 ピクセルまで変化させながら，それぞれの大きさの図形を描画するのにかかる時間を測定した．

また比較のために，いくつかの対照実験を行なった．まず保護がない場合として，モバイルコードの

呼び出しを通常の関数呼び出しで行なう場合の描画時間を測定した．また，モバイルコードを Netscape とは別のプロセスにいれて，IPC (パイプ) を用いて呼び出した場合の描画時間も測定した．さらに，モバイルコードと同様の絵を描画する Java アプレットを作成して，その描画時間を測定した．使用した JavaVM のバージョンは，Netscape 4.76 組み込みの JavaVM 1.1.5 と，ホットスポット対応の Java VM 1.3 である．

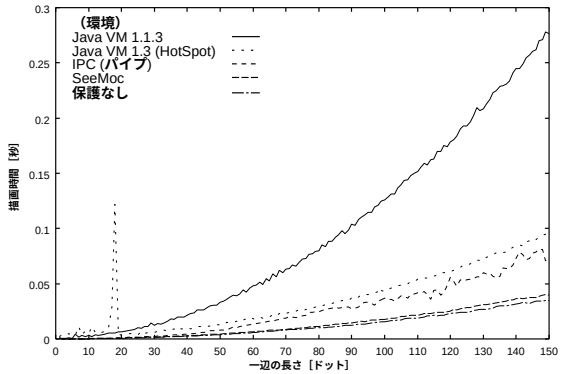

図 2: 保護のオーバーヘッド

図2に，実験結果を示す．細粒度保護ドメインで保護を行なった場合のオーバーヘッドは，保護を行なわない場合に比べて平均すると 12.9% 程度に抑えられている．IPC を用いた場合のオーバーヘッドは平均 102%，Java アプレットは JavaVM 1.3 で 686%，Java VM 1.1.5 では 757% であった．

5 関連研究

Web ブラウザ上でプログラムを安全に動作させる環境としては，Java アプレット [12] による方式が一般的に普及している．Java は安全性を確保するために，型安全な言語の使用や，バイトコードベリファイアによるコードの検証，Java 仮想マシンによる実行時チェックなど，言語処理系としての機能を利用している．しかし Java による方式は，バイトコードと呼ばれる中間コードを Java 仮想マシンで解釈しながら実行するため，ネイティブコードに比べると実行性能が劣る．JIT コンパイラなどの高速化技術を用いても，本論文の実験結果が示すように，ネイティブコードに匹敵するまでにはいたっていない．

ネイティブコードを Web ブラウザ上で実行する仕組みとしては，Microsoft の ActiveX コントロールがある．ActiveX コントロールはネイティブコードで構成されたモバイルコードをインターネットから自動的にダウンロードして Web ブラウザに組み込んで実行する仕組みである．ActiveX コントロールの安全性の確保は，デジタル署名によるモバイルコードの信頼性に頼っており，アクセス制限などの保護機構は用意されていない．

OS に依存しない方式でプロセス内に保護ドメインを形成する仕組みとしては，SFI(Software Fault Isolation) [13] や PCC(Proof Carrying Code) [10, 9] などがある．SFI ではバイナリコードを直接改変してアクセスチェックのための命令を挿入する．また，PCC は，バイナリコードに添付された証明を静的に検証することにより，実行時のチェックを行なわずに安全性を保証する．

OS の機能としてプロセス内に保護ドメインを形成する仕組みとしては，Palladium [3] や PSL (Protected Shared Libraries) [1] などがある．Palladium は Intel CPU のリング保護機構を利用して，プロセス内に 2 段階の保護ドメインを形成する．PSL は POWER プロセッサ上でアドレス空間を部分的に切り替えて，共有ライブラリのデータを安全に共有する．

これらのプロセス内の保護ドメインは，主にメモリ保護のみを目的としたものであり，ファイルなどの資源に対するアクセス制限は想定されていない

従来の保護ドメインであるプロセスの切り替えを高速化する仕組みとしては，さまざまな試みがなされている (LRPC [2], Spring [6], Mach [4], L4 [7, 8]). しかしプロセスによる保護ドメインでは，OS の資源に対して細かなアクセス制限を行なうことはできない．

Janus [5] は，システムコールのトレース機能を利用して，ヘルパアプリケーションのセキュリティホー

ルをついた不正アクセスをを防止している．Janusではシステムコールの監視のために別プロセスを用いており，保護のオーバーヘッドが大きくなる．

6 まとめ

本論文では，Webブラウザ上でモバイルコードを安全に実行できる環境の実現手法を提案した．この手法では，プロセス内のモバイルコードに対して細かなアクセス制御を行なって，モバイルコードによる不正アクセスを防止している．アクセス制御を実現するための機構としては，我々が提案・開発中のSeeMocオペレーティングシステムで提供する細粒度保護ドメインを利用した．また，モバイルコードに細粒度保護ドメインを割り当てるためのプラグインを開発して，既存のWebブラウザを改変せずにモバイルコードを安全に実行する環境を実現した．実験の結果，保護をしたことによるオーバーヘッドは12.9%程度に抑えられることが分かった．

参考文献

[1] Arindam Banerji, John Michael Tracey, and David L. Cohn. Protected Shared Libraries - A New Approach to Modularity and Sharing. In *Proc. of theUSENIX 1997 Annual Technical Conference*, pp. 59–75, October 1997.

[2] Brian N. Bershad, Thomas E. Anderson, Edward D. Lanzowska, and Henry M. Levy. Lightweight Remote Procedure Call. *ACM TOCS*, Vol. 8, No. 1, pp. 37–55, February 1990.

[3] Tzi-cker Chiueh, Ganesh Venkitachalam, and Prashant Pradhan. Integrating Segmentation and Paging Protection for Safe, Efficient and Transparent Software Extensions. In *Proc. of the 17th ACM Symposium on Operating System Principles (SOSP '99)*, pp. 140–153, December 1999.

[4] Bryan Ford and Jay Lepreau. Evolving Mach 3.0 to a Migrating Thread Model. In *Proc. of theUSENIX Winter 1994 Technical Conference*, January 1994.

[5] Ian Goldberg, David Wagner, Randi Thomas, and Eric A. Brewer. A Secure Enviroment for Unstrusted Helper Applications. In *Proc. of the 6th USENIX Security Symposium*, July 1996.

[6] Graham Hamilton, Michael L. Powell, and James G. Mitchell. Subcontract: A flexible base for distributed programming. In *Proc. of the 14th ACM Symposium on Operating System Principles (SOSP '93)*, pp. 69–79, December 1993.

[7] Hermann Härtig, Michael Hohmuth, Jochen Liedtke, Sebastian Schönberg, and Jean Wolter. The Performance of μ-Kernel-Based Systems. In *Proc. of the 16th ACM Symposium on Operating System Principles (SOSP '97)*, pp. 66–77, October 1997.

[8] Jochen Liedtke, Kevin Elphinstone, Sebastian Schönberg, Hermann Härtig, Gernot Heiser, Nayeem Islam, and Trent Jaeger. Achieved IPC Performance. In *Proc. of the 6th Workshop on Hot Topics in Operating Systems (HOTOS '97)*, pp. 28–31, May 1997.

[9] George C. Necula. Proof-Carrying Code. In *Proc. of the 24th ACM Symposium on Principles of Programing Languages (POPL '97)*, pp. 106–119, January 1997.

[10] George C. Necula and Peter Lee. Safe Kernel Extensions without Runtime Checking. In *Proc. of the 2nd Symposium on Operating System Design and Implementation (OSDI '96)*, pp. 229–243, October 1996.

[11] M. Takahashi, K. Kono, and T. Masuda. Efficient Kernel Support of Fine-Grained Protection Domains for Mobile Code. In *Proc. of the 19th IEEE International Conference on Distributed Computing Systems*, pp. 64–73, May 1999.

[12] Java Team, James Gosling, Bill Joy, and Guy Steele. *The Java[tm] Language Spectication*. Addison Wesley Longman, 1996. ISBN 0-201-6345-1.

[13] Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham. Efficient Software-Based Fault Isolation. In *Proc. of the 14th ACM Symposium on Operating System Principles (SOSP '93)*, pp. 203–216, December 1993.

[14] 品川高廣, 河野健二, 高橋雅彦, 益田隆司. 拡張コンポーネントのためのカーネルによる細粒度軽量保護ドメインの実現. 情報処理学会論文誌, Vol. 40, No. 6, pp. 2596–2606, June 1999.

[15] 品川高廣, 河野健二, 益田隆司. セグメント機構を用いた細粒度保護ドメインの性能分析. 情報処理学会研究会報告書,99-OS-82, pp. 17–24, August 1999.