システムソフトウェアと オペレーティング・システム 77－16
マルチメディア通信と分散処理 87－16
（1998. 2. 26）

保護ドメイン簡約による分散実行環境の保護

古川 陽　　柴山悦哉
東京工業大学 大学院 情報理工学研究科 数理・計算科学専攻

{furukawa,etsuya}@is.titech.ac.jp

インターネット経由でコンポーネントを動的に組み込む開放的な分散システムでは，従来よりも細かな保護ドメインの管理が必要とされる．コンポーネント間に信頼関係がない場合，複数の保護ドメインに分割し，コンポーネントの意図しない動作を防止しなければならない．しかし，組み込まれるコンポーネントはあらかじめ予測できず，また変更されることもあるため，保護ドメインの管理をプログラマがあらかじめ記述するのは困難である．本稿では，動的コンパイルの技法を用いて保護ドメインを自動生成する手法を提案する．保護ドメイン簡約による最適化と，軽量保護機構をターゲットとする出力コードの生成により，細粒度の保護が少ないオーバーヘッドで実現されることを示す．

Protection Domain Reduction as a Local Security Support for Open Distributed Systems

Yo FURUKAWA　　Etsuya SHIBAYAMA
Department of Mathematical and Computing Sciences
Tokyo Institute of Technology

{furukawa,etsuya}@is.titech.ac.jp

Distributed systems consisting of various sorts of network loaded components potentially require fine-grained protection domains. To protect the local system from malicious codes, it should locate each component to the individual protection domains. However, fine-grained protection is complicated for both the programmers and the runtime systems. We propose a compiler-based technique which automatically generates optimized protection domains from a simple description of protection policies. Our domain reduction and dynamic code generation techniques provide cost-effective protection for distributed systems.

1 ほじめに

近年，Java[5] をはじめとしてインターネットの開放的な環境を前提としたさまざまな分散システムや言語が提案されている．そのようなシステムでは信頼のないコードを実行することを想定して，ローカルシステムのセキュリティを確保する機構が組み込まれている [9]．例えば，Java ではクラス検証器 (verifier) でメモリの不正参照を防止し，セキュリティマネージャでアクセスできる API を制限している．しかし，多くはあらかじめ定義された保護ポリシーの中で実行を制御するものであり，個別の要求に応じるための拡張性に乏しい．信頼のあるコンポーネントに対して個別にセキュリティポリシーを緩めたり，コンポーネント間の保護を支援する機構はない[1]．

一方で，セキュリティ拡張機構が用意されている場合でも，開放的な分散システムでは必要となるコンポーネントは必ずしもプログラミング時には予測できず，またコンポーネントも変化しうるため，プログラマがコンポーネントの依存関係に注意しながらあらかじめ保護を管理するコードを記述することは困難である．従来のシステムでは保護ドメインの概念はシステムに組み込みのものあり，保護ドメイン自体をプログラミングの対象とすることは少なかった．プログラマに保護機構を開放することにより，かえって負担が増えてしまわないように支援機構が必要である．

この問題に対応するため，本研究では保護コードを自動生成することを試みる．Java のクラスファイルとクラスの利用に関するポリシーから保護ドメインを生成する．生成される保護ドメインは細粒度のものとなるため，そのまま実行時の保護の単位とするとオーバーヘッドが大きくなる場合がある．そこで，開発側，実行側双方でそれぞれのポリシーのもとで静的に解消できる保護ドメインを簡約する特化 (specialization) を行う．保護コードに展開することで実行時の保護ドメイン管理を最小限のものとする．

本稿では，Java のためのプロトタイプ実装をもとに，保護ドメイン簡約の技法，型安全な言語と組み合わせた軽量護機構，実行時保護コード出力の概要を述べる．また，本技法によるオーバーヘッドの見積りと関連研究との比較を示す．

[1]これは最近公開された JDK1.2 のセキュリティ API を用いて実現できる．本研究との関係については関連研究に述べる．

2 保護ドメイン簡約

この節では，ポリシーから保護ドメインを生成する技法を説明する．

2.1 保護の表現と実現

保護を実現するためには，システムが保護モデルを提供し，プログラマ，システム管理者，もしくはユーザのいずれかが保護すべき対象についてのポリシーを記述する必要がある．従来のシステムでは，実行時の保護モデルもしくはメカニズムをそのまま説明し，ユーザにそれにそったプログラミングもしくは記述をさせるものが多い．しかし，数十あるいはそれ以上のの異なる保護ドメインに属するオブジェクトが互いに関連しかつ保護されなければならない状況では，ユーザがそれを把握し適切なプログラミングをすることは困難である．

本研究では，ユーザには誰が何をしてよい何をしてはいけないなどの宣言的で局所的な記述のみをさせ，システムが自動的に保護ドメインを組み立て，最適化し，実行時コードに変換するアプローチをとる．ここで，保護ドメインは説明のためのモデルであり，実装では保護ドメインの中間表現としてアクセスコントロールリスト (ACL) を用い，実行時には型安全な言語を前提とした軽量ケイパビリティを用いる．このように記述のためのモデルと実行時モデルの対応にこだわらないことにより，最適化の機会が広がると考えられる．

2.2 静的な保護と動的な保護

保護には，実行中に権限が変化せず動的なチェックを必要としない静的な保護と，常に動的なチェックを必要とする動的な保護が存在する．例えば，ブラウザに組み込まれた JavaVM のアプレットはダウンロード元のホスト以外に接続できないという制約は，静的な保護で実現できる．一方，ファイルを開くには動的な保護が必要である．静的な保護は実行前にポリシーと照らしあわせることにより解消することができる．

2.3 ポリシー記述と保護ドメインの生成

プロトタイプでは，簡単なポリシー記述と Java のクラスファイルから保護ドメインを生成する．ポリシーは保護される Java のメソッドと必要な権限の関係を記述する．図 1はポリシー記述の例であ

```
# guard
system.net_conn()          needs   NETCONN
system.net_conn_safe()     gains   NETCONN
# inborn capability
allow system                       NETCONN
allow NetSrv                       NETCONN
```

図 1 ポリシー記述の例（部分）

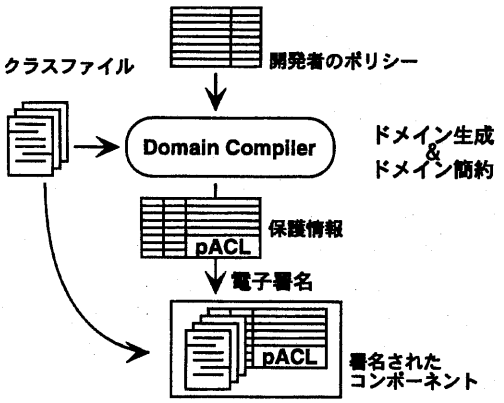

図 2 保護ドメインの自動生成

る．ポリシー記述は 2 つの部分からなる．前半は，各メソッドを呼ぶ際に必要な権限を示している．例えば，無制限のネットワーク接続が可能なメソッド system.net_conn() を呼ぶためには NETCONN が必要である．また，ダウンロード元にのみ接続できる system.net_conn_safe() では，NETCONN 権限が得られることを示している．

後半は，生来的に持っている権限を示している．図では，system 組み込みのクラスライブラリと NetSrv 社のクラスは NETCONN 権限を持ち，ホストの制限なくネットワークに接続できることを示している．この認証には電子署名を用いる．

保護ドメインの生成には，ドメインコンパイラと呼ぶプログラムを用いる．コンパイラはポリシーファイルとのクラスファイルから，pACL(partial ACL) と呼ぶ保護ドメインの中間表現を生成し，元のクラスファイルとともに電子署名する (図 2)．

ドメインコンパイラは，クラスファイルの相互参照情報から提供しているメソッドを調べ，ポリシーファイルを参照してそのメソッドを呼ぶのに必要な権限や得られる権限を書き出す．図 3は生成された

type,	*method,*	*resolve,*	*capability*
	net		
exp,	net_conn(),	static,	-NETCONN
exp,	net_conn_safe(),	static,	+NETCONN
	URL		
exp,	URL_read(),		
imp,	net_conn(),		
imp,	file_open(),		
imp,	file_read(),		
	net+URL		
exp,	net_conn(),	static,	-NETCONN
exp,	net_conn_safe(),	static,	+NETCONN
exp,	URL_read(),	static,	?NETCONN
imp,	file_open(),		
imp,	file_read(),		

図 3 生成された pACL

pACL の例である．type フィールドの exp はクラスが提供しているメソッドを示す．resolve フィールドの static は権限が静的に解消できる場合があることを示している．

2.4 保護ドメイン簡約

ドメインコンパイラはまず 1 つのクラスファイルについて 1 つのドメイン（1 つの pACL）を生成する．しかし，これを実行時保護ドメインとすると，膨大な保護ドメイン間通信を生じるためオーバーヘッドが大きくなる．そこで，実際には，パッケージ単位，JavaBeans 単位など，より粒度の大きなの保護ドメインを作る必要がある．

ドメインコンパイラは，ドメイン簡約と呼ぶ操作により 2 つの保護ドメインをマージし，より大きな保護ドメインを生成する．簡約は主に次の 2 つの解消を行う．

- 権限が必要ない呼出しは静的に解消する
- 権限が必要である場合，記述されたポリシーのもとで権限のチェックを行い，静的に解消できるものは省略する

コンポーネントを構成するクラスの間でインクリメンタルにマージすることにより，大きな保護ドメインを生成する．

図 3に簡単なマージの例がある．URL クラスはネットワーク接続とファイルの読みだしを必要と

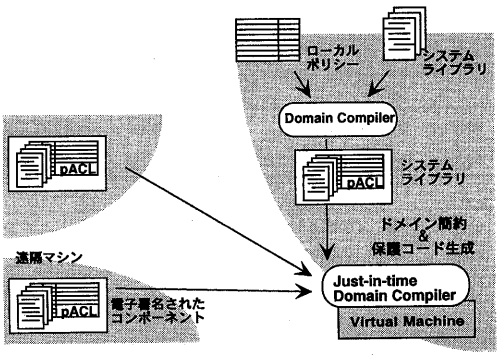

図 4 システムの概要

するクラスであり，system が提供する．system のクラスは NETCONN 権限を持つので，URL_read() は net_conn() を呼び出すことができ，このチェックは静的に解消される．ここで，URL_read() を呼び出すクラスは NETCONN 権限を持つとは限らないが，net_conn() を間接的に呼び出しているので NETCONN 権限は必要である．図の中で?NETCONN は，URL_read() 内では権限がチェックされないが，のちのち必要であることを示している．

2.5 分散実行環境の保護

生成された保護ドメインを含むコンポーネントは図 4に示すようにネットワーク経由でダウンロードされ実行される．クラスローダに組み込まれた JIT ドメインコンパイラが，他のクラスとリンクする際に最終的な簡約を行う．簡約には実行マシンのポリシーが用いられる．ここで，静的に簡約可能な保護はすべて解消され，解消されなかった保護は次節に述べる方法で動的なチェックが行われる．

3 スタックケイパビリティ

解消されなかった保護は JIT ドメインコンパイラにより，保護機構を実現するプログラムコードに変換される．保護は局所的に解決されるため，実行時に明示的な保護ドメインの管理を行う必要がない．

3.1 スタックケイパビリティ機構

JIT ドメインコンパイラはスタックケイパビリティと呼ぶ技法を用いた保護コードを生成する．

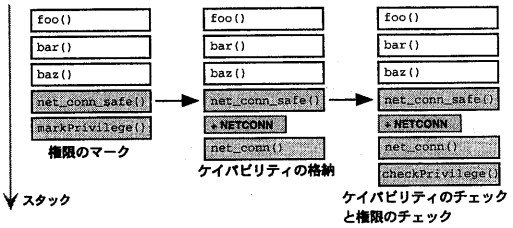

図 5 スタックケイパビリティ

スタックケイパビリティでは，その場で獲得したケイパビリティをコールスタックにプッシュする (markPrivilege)．アクセス権限をチェックする際にはスタックを逆にたどり，権限をあらわすケイパビリティをサーチする (checkPrivilege)．ここで，Java は型安全な言語であり，ユーザプログラムからスタックに格納されたケイパビリティに不法にアクセスすることはできない．そのため，ケイパビリティを守るために暗号化したり，特権領域に置く必要がなく，コストがかからない．また，この技法は次の特質を持つ．

- コンテキストとケイパビリティを同時に管理することができ，保護を局所的に解決できる
- ケイパビリティをキャンセルするコストが不要であるため効率がよい

スタックをケイパビリティの格納場所として用いる技法は他にも提案されているが [11]，本研究の技法は保護コードを自動生成する点と保護が遅延チェックにより行われる点が特異である．

3.2 コード生成

JIT ドメインコンパイラは解消できなかったすべての保護ドメイン間コールについて保護コードを生成する．保護ドメイン情報から権限が必要な呼出しの前に markPrivilege を挿入し，権限のチェックが必要なメソッドの最初に checkPrivilege を挿入する．図 6は URL_read() の呼出しにおける保護コード自動挿入の例である．ここで，markPrivilege では署名のＩＤと権限をスタックにプッシュするだけであり，実際にそのＩＤと権限の対が適切であるかのチェックが行われるのは checkPrivilege である (遅延チェック)．これは，markPrivilege は機械的に挿入されるため，実行のトレースによっては権限が

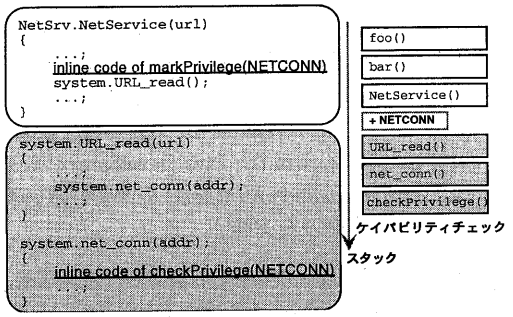

図 6 保護コードの自動挿入

利用されないこともあり，無駄なチェックを生じる可能性があるためである．チェックが成功する場合（正しい権限を持ったコード）は最小のコストで実行されるが，チェックが失敗する場合は markPrivilege から checkPrivilege の実行が無駄になる．

4 プロトタイプの実装と実験

プロトタイプシステムを JavaVM の実装の 1 つである Kaffe に実装し，本研究の方式のオーバーヘッドの見積りを行った．実験には，Pentium 150MHz, Linux 2.0.30, kaffe 0.9.2, gcc 2.7.2 を用いた．

HORB で小さなメッセージを 100,000 回ピンポンする単純なアプリケーションを作成した．このアプリケーションは，Sun のライブラリ，HORB ORB，ユーザプログラムからなる．3 者を違う保護ドメインで実行した場合 (*vender*) と Sun のライブラリと HORB が互いに信頼できるとして保護ドメインを作った場合 (*trust*) で比較し，以下のような削減率が得られた．ただし，保護ドメイン間コールの数や削減率は，アプリケーションの持つ保護ドメインの数や信頼ポリシーの設定によって大きく異なる．

	保護ドメイン間コール		
	vendor	trust	削減率
server	2,000,544	600,004	70.0%
client	1,300,470	500,015	61.6%

次に，軽量保護機構の評価のために，スタックケイパビリティを Kaffe のインタプリタ版と JIT 版に実装した．空の関数コールと，markPrivilege と checkPrivilege の対を追加した関数コールの実行速度を以下に示す．また，比較のために，C で同様の処理をするコードを作成した．同じマシンでもっとも軽量なシステムコールが 6μsec かかることを考慮に入れれば，妥当なオーバーヘッドで実現されていると考えられる．

(μsec)	null	mark + check	overhead
Intrp	13.3	14.3	1.00
JIT	0.137	0.614	0.530
C	0.101	0.603	0.502

最後に，上記アプリケーションで単位時間あたりの保護ドメイン間コールの数を数え，保護をしない場合と較べた実行時のオーバーヘッドを見積もった．ごく少ないオーバーヘッドが実現されており，より細粒度の保護ドメインを用いるアプリケーションにも対応できると考えられる．

	time	trust		
client	(sec)	cross	cross/sec	overhead
Intrp	354.3	500,015	1,411	0.141%
JIT	169.0	500,015	2,959	0.157%

5 関連研究

保護ドメインを自動生成する点では直接対応する研究ではないが，インタフェースを自動生成する点では RPC スタブジェネレータなどに近い．また，ロード時に保護コードを挿入する点では Software Fault Isolation [10] などの技法と同類と考えられる．

SPIN

SPIN はカーネルに組み込むカーネル拡張モジュールを型安全な Modula-3 で記述し，コンパイル時にメモリ参照の安全性を確認する．また，SPIN の動的リンカ [7] はモジュールがアクセスするインタフェースを実行前に（静的に）チェックし，実行時のチェックを極力省くことができる．さらに，アクセスしてよいインタフェースの集合を保護ドメインとして定義したり，保護ドメインを簡単に作成するジェネレータが用意されている．

これらの機能は本研究の保護ドメイン自動生成に相当すると思われるが，モジュールからカーネルを保護する目的で開発されているため，複数のユーザモジュールの間で保護を簡約する機構は用意されていない．JavaBeans のように数多くのモジュールが連係するアプリケーションでは，ユーザが保護ドメインを管理するのは困難であると思われる．

JDK1.2

最近公開されたJDK1.2のセキュリティ拡張[3, 4]では，Javaのプログラムの中で保護ドメインを定義しプログラミングすることにより，従来のSandboxの制限を越えてセキュリティを拡張できるようになっている．保護は，クラスと保護ドメインと権限，それぞれのオブジェクトを対応づけすることにより定義する．また，あらかじめ用意されたものについては，テキストファイルでポリシーを記述することができる．

しかし，このモデルで意図した保護を得るためには，作者らも認めているように注意深いプログラミングが必要となる．多数のクラスについて細かい保護ドメインを記述するのは繁雑である．本研究では，細かな関係の解析と保護を自動化することにより，プログラマの負担を少なくする．

6 まとめと将来課題

本稿では，分散環境の細粒度保護のために保護を最適化する方法とJavaVMの中で実現する方法を紹介した．生成や簡約を自動化することによりプログラマの負担を増やすことなく細かな保護を効率的に実現することができる．また，プログラマが保護コードを書かなくてもよいので不注意によるセキュリティホールを少なくすることができると期待できる．

現在，部分評価[1]の手法を用いて動的なケイパビリティの簡約を実現する研究を進めている．実用的な面では，JDK1.2のセキュリティモデルと互換性を持たせることが重要であると考え，本研究の手法を応用する方法を検討している．具体的には，ポリシー記述とコード解析からJDK1.2の保護ドメインを自動生成する方法，JDK1.2の保護ドメインをロード時簡約することにより実行を効率化する方法を検討している．

参考文献

[1] Charles Consel and Olivier Danvy. Tutoirial Notes on Partial Evaluation. In *Symposium on Principles of Programming Languages*. ACM, 1993.

[2] F. Yellin and Sun Microsystems, Inc. Low Level Security in Java, 1996. http://java.sun.com/sfaq/verifier.html.

[3] Li Gong, Marianne Mueller, Hemma Prafullchandra, Roland Schemers, and Inc. JavaSoft, Sun Microsystems. Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2. In *Symposium on Internet Technology and Systems*. USENIX, 1997.

[4] Li Gong, Roland Schemers, and Inc. JavaSoft, Sun Microsystems. Implementing Protection Domains in the Java Development Kit 1.2. In *Symposium on Network and Distributed System Security*. Internet Society, 1997.

[5] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[6] Przemyslaw Pardyak and Brian Bershad. Dynamic Binding for an Extensible System. In *Second USENIX Symposium on Operating Systems Design and Implementation*, p. 1996. ACM, October 1996.

[7] Emin Gun Sirer, Marc Fiuczynski, Przemyslaw Pardyak, and Brian Bershad. Safe Dynamic Linking in an Extensible Operating Systems. In *Workshop on Compiler Support for System Software*. ACM, 1996.

[8] Kenjiro Taura, Satoshi Matsuoka, and Akinori Yonezawa. StackThreads: An Abstract Machine for Scheduling Fine-Grain Threads on Stock CPUs. In *Joint Symposium on Parallel Processing*, 1994.

[9] Tommy Thorn. Programming Languages for Mobile Code. *ACM Computing Surveys*, Vol. 29, No. 3, pp. 213 – 239, September 1997.

[10] Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham. Efficient Software-Based Fault Isolation. In *the 14th ACM Symposium on Operating Systems Principles*, pp. 203 – 216. ACM, 1993.

[11] D. Wallach, D. Balfanz, D. Dean, and E. Felten. Extensible Security Architectures for Java. In *the 16th Symposium on Operating System Principles*. ACM, 1997.