システムソフトウェアと
オペレーティング・システム 81－10
（１９９９．５．６）

組み込みOSにおけるシステムオブジェクトのダウンロードを用いたアプリケーションオブジェクトの実行最適化

岡村英明
ソニーコンピュータサイエンス研究所

横手靖彦
ソニー株式会社
インフォメーションネットワーク研究所

アプリケーションの特性に応じたシステムオブジェクトをダウンロードし，アプリケーションオブジェクトの動作を最適化する方法を提案する．システムオブジェクトを安全に置換するためにダウンロード実行権を定義するレベル（ダウンロード許可レベル/受諾レベル）と，アプリケーションの実行を妨害しないようにシステムオブジェクトのサービスを管理する構造体(システム依存リスト，サービスリスト)を導入している．また，組み込みOSに適するようにメモリ消費量の少ない方法を採用している．本方法をオブジェクト指向組み込みOS Aperios上に実装した．性能評価の結果，本方法でアプリケーションオブジェクトの実行性能が改善されることを示す．

Optimization of Application Object Execution by System Object Downloading on Embedded Operating Systems

Hideaki Okamura
Sony Computer Science Laboratories Inc.

Yasuhiko Yokote
Information and Network Technologies Lab.
Sony Corporation

We have developed a method for optimizing application object execution by system object downloading. To support safe downloading of system objects, downloading permission levels called downloaded levels and downloadable levels are introduced. Primitive structures, a system dependency list and a service list are also introduced to ensure that application object execution does not fail. To maintain minimal memory usage for downloading, which is important in embedded OSs, the code for downloading system objects can be shared with that used for downloading application object. An implementation of our method on Aperios, an object-oriented OS, is presented and our method is shown to improve the execution performance.

1 はじめに

ネットワークやディスクからソフトウェアモジュールをダウンロードし実行する機能は，ワークステーションやパーソナルコンピュータの標準機能となりつつある．家電上の組み込みOSも，家電の情報家電への進化(ネットワークやCD, MDなどの着脱可能メディアの装備)に伴い同等の機能を持ちつつある．ダウンロード機能がない従来の組み込みOSでは，全てのアプリケーションプログラム（以下，アプリケーション）の性質を解析後，動作を最適化可能だったが，ダウンロード機能を持つ情報家電の場合，ダウンロードされ実行される全てのアプリケーションの性質を予測して，そこで動作する組み込みOSを設計することは不可能である．この場合，OSは全てのアプリケーションに対して最適なサービスを提供できない．例えば，メッセージ通信のOSサービスを考える．AppA, AppBは，それぞれが複数のプログラミングモジュールからなる2種類のアプリケーションとする．AppAは1つのデータベースサーバと，2つのクライアントから成る．これらのクライアントは対等な優先度でサーバと通信する．一方，AppBは，Ethernetからのデータを処理するクライアント，Sereal Lineからのデータを処理するクライアントとクライアントからの要求に応じてデータ処理をするサーバから成る．Sereal LineとEthernetの動作には優先順位があり，Serial Lineはより頻繁にデータを処理する．ただし，時々Ethernetからの制御パケットの優先的処理が必要である．AppAに対しては，なるべく単純な通信処理が最適な実行性能をもたらす．一方，AppBではメッセージ通信機構にメッセージ優先度の処理が必要である．優先度付きメッセージ通信機構を用いると，AppAには優先度チェックという不必要な処理が行われ，性能が劣化する．一方，優先度なしメッセージ機構を用いると，AppBは要求に応じた動作が不可能である．

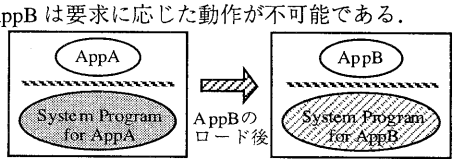

図1. アプリケーションに応じたシステム変更

この時，AppA，AppBをダウンロードし，それぞれを最適実行することを考えると，図1のようにメッセージ通信サービスを提供するシステムプ

ログラムを，各アプリケーションオブジェクトの性質に応じて変更する必要がある．

想定しうる複数の機能をあらかじめOSに装備し，アプリケーションから選択する方法をとれば，いくつかのアプリケーションに適応可能である．しかし，全てのアプリケーションを満足させる機能を提供することは不可能である．特に，組み込みOSはハードウェア上の搭載メモリ量に制約があるため，システムプログラムのサイズにも制約がある．よって，あらかじめ複数機能を用意することは避けるべきである．

以上の問題点を解決するには，アプリケーションの性質に合わせて，システムプログラムを柔軟に変更する必要がある．最近は，Unix系のシステムでアプリケーションの性質に応じたシステムの変更，拡張機構に関する研究[1][2]が活発だが，組み込みOSにも同様な機能が求められる．

本論文では，システムオブジェクトのダウンロードによるアプリケーションオブジェクトの動作の最適化方法を提案する．すなわち，オブジェクト指向技術を導入し，OSを構成するシステムプログラムをシステムオブジェクトとして定義し，オブジェクトのダウンロードの機能をシステムオブジェクトにも適用する．Open Implementationの技術[3]を用いて，アプリケーションの性質を熟知しているアプリケーションプログラマにシステムオブジェクトをプログラミング可能にする．アプリケーションプログラマは，アプリケーションの性質に応じてシステムオブジェクトをカスタマイズする．そして，アプリケーションオブジェクトのダウンロード時に，システムオブジェクトを同時にダウンロードし，既存のシステムオブジェクトを置換して，アプリケーションプログラムの実行最適化を行う．ここでオブジェクト指向技術の適用は，次の意味でも組み込みOSに都合がよい．第一に，プログラムをダウンロードする場合に，様々な情報をオブジェクト内部に隠蔽して複雑なオペレーションを単純化できる．第二に，システムオブジェクトとアプリケーションオブジェクトの基本的実行機構を共有しやすい．提案された方法の実装には，オブジェクト指向OS Aperios（以前はApertos[12]）を組み込みOSとして用いる．

本論文では，第2章でシステムオブジェクトダウンロードに対する要求事項と，その解決方法を述べる．第3章で実行最適化の適用例，第4章でAperios上での実現，第5章で本方法の評価，第6章で本論文のまとめと関連研究を述べる．

2 システムオブジェクトダウンロード

本章では，システムオブジェクトダウンロードに対する要求事項と，解決方法について述べる．

2.1 本研究でのシステムの前提

「オブジェクト」はプログラマが変更，置換可能なソフトウェアモジュールで，相互にメッセージ通信可能である．オブジェクトにはOSのサービスを提供する「システムオブジェクト」と，アプリケーションプログラムを構成する「アプリケーションオブジェクト」がある．OSは「カーネル」(アプリケーションプログラマにより不可変で，コンテキスト切替などの最低限の機能を装備）と，複数のシステムオブジェクトから成る．システムオブジェクトはメッセージ送信要求，メモリ割当要求などアプリケーションオブジェクトの要求に応じたサービスを提供する．システムオブジェクトは，実行スタック，メモリセグメントなどの計算資源を直接アクセス可能である．アプリケーションオブジェクトからのサービス要求はカーネルに処理され，システムオブジェクトが起動される．サービス提供が終了すると，制御はアプリケーションオブジェクトに戻る．ダウンロードのサービスは「ダウンローダ」と呼ばれるシステムオブジェクトが提供する．

2.2 要求事項

システムオブジェクトはアプリケーションオブジェクトにサービスを提供するため，以下のことを考慮する必要がある．

- ダウンロード要求権

  システムオブジェクトのダウンロードを全てのオブジェクトへ許可することは不可能である．不適当なタイミングでのシステムオブジェクトの置換や悪意のある置換により，システムのサービス提供が不可能になる．最悪の場合，システムが動作不能になる可能性もある．このような事態を避けるため，ダウンロード要求をするオブジェクトに対して「ダウンロード要求権」を与えることで，システムオブジェクトのダウンロードを要求可能なオブジェクトを制限する方法が必要である．

- システムサービスの管理

  アプリケーションにサービス提供中のシステムオブジェクトの置換は注意が必要である．例えば，メッセージ送信サービス提供オブジェクトの置換中に，メッセージ送信要求が発生すると誤動作する可能性がある．また，

メッセージ送信サービス提供オブジェクトを置換中に，メッセージ送信処理中のオブジェクトがある場合，置換中のオブジェクトは，メッセージ送信サービス用の中間データ(メッセージキュー中のメッセージなど)を持つが，このデータを置換後に新しいシステムオブジェクトへ渡す必要がある．これらの問題の解決には，「システムサービスの管理」を行い，アプリケーションの実行を妨害せずシステムオブジェクトの置換を行う必要がある．

2.3 解決方法

2.3.1 ダウンロード受諾レベル / 許可レベル

「ダウンロード要求権」を定義するために「ダウンロード受諾レベル」と「ダウンロード許可レベル」を導入する．ダウンロード受諾レベルはダウンロードされるオブジェクトに設定される整数値である．この値が大きいほど，ダウンロードは困難になる．システム設計者はシステムオブジェクトに適当なダウンロード受諾レベルを設定する．基本的にアプリケーションオブジェクトのダウンロード受諾レベルはシステムオブジェクトのものより小さく設定される．一方，ダウンロードを要求するオブジェクトには，ダウンロード許可レベルが設定される．ダウンロード許可レベルはオブジェクトをダウンロードする能力を示す．このレベルもシステム設計者により設定される．

オブジェクト A のダウンロード許可レベルが Da_A でオブジェクト B のダウンロード受諾レベルが Dp_B の時，条件 1 を満たす時，オブジェクトAがオブジェクトBをダウンロード可能である．

条件 1

Da_A > Dp_B

例えば，AppA のダウンロード許可レベルを 10 とする．AppB, AppC のダウンロード受諾レベルがそれぞれ 5，20 の時，AppA は AppB を新しいオブジェクトをダウンロードして置換可能だが，AppC を置換できない．

2.3.2 システム依存リストとサービスリスト

「システムサービスの管理」のため，「システム依存リスト」と「サービスリスト」の両構造体を導入する．サービス依頼者であるアプリケーションオブジェクトの動作は，システムオブジェクトで提供されるサービスの集合としても定義される．ここで，アプリケーションオブジェクトがどのシステムオブジェクトからサービス提供されるかを示すリストがシステム依存リストである．システム依存リスト Dlの定義は，定義1の通りである．

定義 1: システム依存リスト

Dl = {<Index_1, Lst_1>, <Index_2, Lst_2>,..., <Index_n, Lst_n>}
Index_n: OS サービスのインデックス
Lst_n: サービスを提供するシステムオブジェクトの ID のリスト

Index_n は OS サービスのインデックス，Lst_n はサービスを提供するシステムオブジェクトの ID のリストである．システムオブジェクトがダウンロードにより置換される場合，システム依存リストは，利用されるだけでなく更新される必要がある．なぜなら，そこに含まれるシステムオブジェクト ID が変更される可能性があるからである．

一方，サービス提供者のシステムオブジェクトは，定義2で定義されるサービスリスト Sl を持つ．

定義 2: サービスリスト

Sl = {<ID_1, Dl_1>, <ID_2, Dl_2>,... , <ID_n, Dl_n>}
ID_n: アプリケーションオブジェクトの ID
Dl_n: アプリケーションオブジェクトのシステム依存リスト

サービスリストは，システムオブジェクトをダウンロードし，既存のシステムオブジェクトを置換する場合，そのシステムオブジェクトのサービスが続行中であることと，アプリケーションオブジェクトがそのシステムオブジェクトへサービス提供要求中であることの確認に用いられる．ダウンローダオブジェクトは，置換されるシステムオブジェクトのサービスリストを調査し，そのシステムオブジェクトがサービスを提供するアプリケーションオブジェクトの存在を調べる．アプリケーションオブジェクトが存在した場合は，例えば，置換を中止するか延期する処理を行う．サービスリスト内のシステム依存リストは，各アプリケーションオブジェクトの持つシステム依存リストの正しさを確認するのに用いられる．なお，システムオブジェクトがどのアプリケーションオブジェクトにもサービスを未提供ならば，サービスリストは null である．

システム依存リストとサービスリストは，各オブジェクトの初期化時に作成される．システム依存リストとサービスリストの例は第 3 章で示す．

2.4 置換アルゴリズムの概要

既存のシステムオブジェクトをダウンロードを用いて置換するアルゴリズムの概要を述べる．詳細なアルゴリズムは文献 [10] を参照されたい．

置換過程はアンロードフェーズとロードフェーズからなる．アンロードフェーズは，既存のシステムオブジェクトの停止，削除 (アンロード) を行う．

1. 条件 1 を確認し，アンロード要求をしたオブジェクトのダウンロード許可レベルが，削除されるシステムオブジェクトのダウンロード受諾レベルより大きいかを調査する．
2. 削除されるシステムオブジェクトのサービスリストを調査し，各アプリケーションオブジェクトが，サービスリストから参照されるシステム依存リストを正しく保持しているか確認する．アプリケーションオブジェクトの動作状態が動作中なら，削除を延期する．
3. 動作中のオブジェクトがなければ，削除要求をされたオブジェクトの動作を停止する．
4. 削除されるオブジェクトの ID, 実行スレッド，スタック，使用メモリ領域を解放する．

ロードフェーズは新しいオブジェクトに必要メモリ領域を確保し，オブジェクトをロードする．

1. 条件 1 を確認する．満たさない場合は，エラーを返し，ロードフェーズを中止する．
2. 新しいオブジェクトの ID，実行スレッド，スタック，使用メモリ領域を割り当て，ダウンロード元からその内容をコピーする．
3. サービスリストから，新しいシステムオブジェクトにサービスを提供されるアプリケーションオブジェクトを探し，それぞれのシステム依存リスト中の削除されたシステムオブジェクトの ID を新しいものに更新する．

3 実行最適化の適用例

本章では，ダウンロードを用いた実行最適化の適用例として，メッセージ通信方式の最適化の例を示す．AppA は 3 つのアプリケーションオブジェクト AO-a1, AO-a2, AO-a3 からなり，各オブジェクト間では対等の優先度，頻度のメッセージ通信を行う．実行性能の観点から，単純なメッセージ通信機構が必要である．AppB は AO-b1, AO-b2, AO-b3 からなる．通常，AO-b1, AO-b2 間のメッセージ通信は AO-b1, AO-b3 間のメッセージ通信より優先的に処理される．時々，AO-b1, AO-b3 間には高優先度のメッセージ通信が行われる．AppB には，優先メッセージを処理するメッセージ通信機構が必要である．

単純なメッセージ通信機構を持つシステムオブジェクトを MO1, 優先メッセージ処理機構を持つシステムオブジェクトを MO2 とする．AppA, AppB は同時に動作せず，それぞれ，アプリケーションオブジェクト AppC にダウンロードされる．AppC は，AO-a1, AO-a2, AO-a3 のロード前に，MO1 をロードする．この時，AO-b1, AO-b2, AO-b3, MO2 をアンロードする．AppB をロードする時，図 2 のように，AO-a1, AO-a2, AO-a3, MO1 をアンロードし，MO2, AO-b1, AO-b2, AO-b3 をロードする．

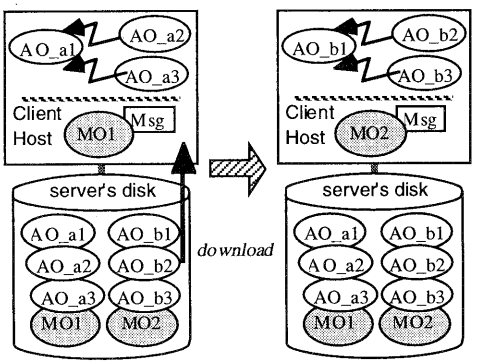

図 2. 最適なメッセージ通信機構のサポート

こうして，AppA, AppB それぞれの場合に最適なメッセージ通信機構を利用できる．

```
AppC::LoadAppB ()
{
    Unload ("AO-a1");
    Unload ("AO-a2");
    Unload ("AO-a3");
    Unload ("MO1");

    Load ("MO2");
    Load ("AO-b1");
    Load ("AO-b2");
    Load ("AO-b3");
}
```

図 3.AppC の LoadAppB() メソッド

図 3 は AppB をダウンロードする AppC のメソッド LoadAppB() のプログラム例である．Load() は引数で示されるオブジェクトをロードする API である．置換アルゴリズム中のロードフェーズが，Load() の呼出しにより実行される．Unload() は引数で示されるオブジェクトをアンロードする API である．置換アルゴリズム中のアンロードフェーズが，Unload() の呼出しにより実行される．

この時，AppC のダウンロード許可レベル Da_appc と MO1, MO2 のダウンロード受諾レベル Dp_mo1, Dp_mo2 の間に，条件 1 を満たすように，Da_appc > Dp_mo1, Da_appc > Dp_mo2 の関係が成り立っている．また，MO1 のサービスリスト Sl_mo1，MO2 のサービスリスト Sl_mo2，AO-a1, AO-a2, AO-a3 のシステム依存リストをそれぞれ Dl_AO-a1, Dl_AO-a2, Dl_AO-a3，AO-b1, AO-b2, AO-b3 のシステム依存リストをそれぞれ Dl_AO-

b1, Dl_AO-b2, Dl_AO-b3 とすると，これらは図 4 の関係を満たす．ここで，ID_obj は obj の ID を示し，0x10 はメッセージ送信サービスのインデックスである．なお，他の最適化（カスタマイズ）の例は文献 [10] を参照されたい．

```
Sl_mo1 = {<ID_AO-a1, Dl_AO-a1>, <ID_AO-a2,
      Dl_AO-a2>, <ID_AO-a3, Dl_AO-a3>}
   Dl_AO-a1 = {<0x10, {ID_MO1}>...}
   Dl_AO-a2 = {<0x10, {ID_MO1}>...}
   Dl_AO-a3 = {<0x10, {ID_MO1}>...}
Sl_mo2 = {<ID_AO-b1, Dl_AO-b1>, <ID_AO-b2,
      Dl_AO-b2>, <ID_AO-b3, Dl_AO-b3>}
   Dl_AO-b1 = {<0x10, {ID_MO2}>...}
   Dl_AO-b2 = {<0x10, {ID_MO2}>...}
   Dl_AO-b3 = {<0x10, {ID_MO2}>...}
```

図 4. 第 3 章の例題に関連するリスト

4 Aperios での実装

Aperios は動的置換が不可能な部分を最小限にし，OS のサービス提供部分を並行オブジェクト [13] で実現した OS である．置換不可能な部分 (2.1 節における「カーネル」) を MetaCore と呼ぶ．システムオブジェクトは，メッセージ通信，スケジュリング，メモリ管理，ダウンロードの機構を提供する．システムオブジェクト間通信はメッセージ通信で行われる．基本的に全てのシステムオブジェクトはダウンロード可能である．よって，Aperiosは2.1節のシステムの前提を満たしている．

ダウンロード実行では，ダウンローダオブジェクトが中心的な役割をする．Load(), Unload() の API に対するサービスは，ダウンローダが提供する．Load() が用いられると，ダウンローダが起動され，ダウンロード可能モジュールと呼ぶオブジェクトのイメージ（コード領域，データ領域）と，スタックサイズなどのオブジェクト情報を含むバイナリデータを RAM 上にコピーする．

本研究では，アプリケーションオブジェクトだけでなくシステムオブジェクトも扱えるようにダウンローダを拡張した．拡張コード量は 2,440 バイトで，ダウンロード機能の全体 (72,016 バイト) の約 3% である．よって，システムオブジェクト用の追加部分は，メモリ量としてそれほどオーバーヘッドにならず，したがって，本論文で提案したダウンロード方法は組み込み OS に適していると言える．なお，コンパイルには Green Hills Software 社製 C++ コンパイラを用いており，単純な比較のため，コード量最適化は行っていない．

本章では Aperios の実現について述べたが，本論文で提案した方式は Aperios 以外のシステムでも実現可能である．例えば，Java OS [7] でも OS 提供部分がダウンロード可能なソフトウェアモジュールになっており，クラスローダのようなダウンロード機構を変更すれば，第 2 章で述べた実行最適化方法を適用可能であると考えられる．

5 性能評価

本論文で提案したアプリケーションオブジェクトの動作の最適化方法の評価をするため，第 4 章で議論した Aperios 上での実装での実行性能評価を行った．測定は，CPU に MIPS R4300 (50MHz) を持つ評価ボード (16M バイト RAM 搭載) 上で行った．キャッシュヒットのばらつきを防ぐため，測定ではキャッシュ機構をオフの状態にした (つまり，実際の実行性能は 5-10 倍程度高速になる)．本評価での測定は比較が重要なので，コンパイラによる実行最適化は行っていない．

ここでは，第 3 章で例示したアプリケーション AppA において，メッセージ通信方法を変更した時の，最適化前後のメッセージ送信コストを示す．本章ではメッセージ通信サービスを提供するシステムオブジェクトを「メイラ」と呼ぶ．評価プログラム AppA は AO-a1, AO-a2, AO-a3 から成る．評価は AO-a1 と AO-a3 の間のメッセージ通信の往復時間を測定した．表 1 に結果を示す．

表 1. メッセージ通信時間

繰り返し回数	1000	3000	10000
ロード前 (μs)	4424	4404	4390
ロード後 (μs)	4320	4359	4357
割合 (後 / 前)	0.976	0.990	0.990

表 1 において，「ロード前」は，優先度付きメッセージ通信機構を持つメイラ (優先メイラ) での測定結果である．AppA にとっては，優先メッセージ処理機構は不必要なので，最適なメイラではない．「ロード後」は，単純なメッセージ通信機構に変更したメイラ (単純メイラ) をダウンロードした後に測定した通信コストである．それぞれの場合をメッセージ通信を繰り返し (1000, 3000, 10000 回)，メッセージの往復時間を測定し，平均値を結果とした．結果として，メイラオブジェクトをダウンロードして置換した場合，1-2% 程度の実行性能の改善がある．今回の変更は，メイラのコード量で約 340 byte だけの単純な変更だったので効果が小さいが，最適化の方法により，より多くの性能改善が期待できる．

表2にメイラをダウンロードする時のコストを示す．測定では，メイラをフラッシュ ROM から

RAM 上にダウンロードした．表 2 の「データ転送量」は，メイラのコード領域，データ領域およびオブジェクト情報を全て含んだ転送データ量である．「ロード時間」は，アプリケーションオブジェクトでLoad() APIを発行してからその実行終了までの時間である．アプリケーションオブジェクトの単純なダウンロード，実行に比較して，システムオブジェクトのダウンロードの分だけアプリケーション起動が遅延する．ここでは，AppA 全体の動作時間の割合がダウンロード時間に比べて大きいため，システムオブジェクトのダウンロード時間の操作上の影響は小さい．

表2. メイラのダウンロードコスト

	データ転送量 (byte)	ロード時間 (ms)
単純メイラ	40107	116.5
優先メイラ	40451	116.7

6 おわりに

本論文では，システムオブジェクトのダウンロードによるアプリケーションオブジェクトの動作の最適化方法について述べた．本方法では，システムオブジェクトを置換の安全性に対する考慮をしている．ダウンロード受諾レベル / 許可レベルの導入により，不適当なダウンロード要求を却下可能になった．システム依存リスト，サービスリストの導入により，実行中のアプリケーションの実行を妨害せずにシステムオブジェクトを置換可能になった．オブジェクト指向 OS Aperios 上に本方法を実装し，アプリケーションの性質に応じてメッセージ通信サービス提供オブジェクトを置換した時に，メッセージ通信の性能改善が得られることを示した．

関連研究には，まず OS が Open Implementation されていて，OS を提供するモジュールを動的に変更可能な Unix 系の拡張 OS (例えば，Exokernel [2]．SPIN[1]) がある．これらのシステムにも，本論文の方式を適応可能である．また，最適化のためのプログラミングは，一種の自己反映的プログラミング [8][9] である．メタオブジェクトプロトコル [4] (本研究では，システムオブジェクト定義用プロトコル) の導入で，より安全な変更方法が期待できる．Java 言語系のシステムは，Java Core Reflection [11] のようにシステムの挙動を変更可能なものもある．これらの研究では，著者の知る限りでは，メッセージ通信メカニズムのような OS のサービス提供部分の変更機構は未提供であり，ダウンロードを用いた最適化も行っていない．しかし，提案した最適化方法をクラスローダに組み込み，同様な最適化を行うことも可能だと考えられる．

謝辞

Aperios を共に開発し，有益な議論をしてくれたソニー株式会社スープラストラクチャセンター組み込みソフトウェア開発部の諸氏に感謝する．

参考文献

[1] B.N. Bershad, et al. Extensibility, safety and performance in the SPIN operating system. In *Proceedings of the 15th ACM Symposium on Operating Systems*, pages 267-284, 1995.

[2] M. Frans Kaashoek, et al. Application Performance and Flexibility on Exokernel Systems. In *Proceeding of the 16h ACM Symposium on Operationg Systems Principles*, 1997.

[3] G. Kiczales. Beyond the black box: Open Implementation. IEEE Software, 13(1): 8,10-11, 1996.

[4] G. Kiczales, J. des Rivieres and D. G. Bobrow, The Art of the Metaobject Protocol. The MIT Press, 1991.

[5] Rodger Lea, Yasuhiko Yokote, and Jun-ichiro Itoh. Adaptive Operating System Design using Reflection. In *Proceedings of USENIX Conference on Object Oriented Technologies*, June 1995.

[6] Sheng Liang and Gilad Brancha. Dynamic Class Loading in the Java Virtual Machine. In *Proceedings of ACM Object-Oriented Programming Systems, Languages and Application*, October 1998.

[7] Peter Madany. JavaOS: A Standalone Java Environment. *http://java.sun.com/products/javaos/javaos.white.html*, 1998.

[8] P. Maes. Concepts and Experiments in Computational Reflection. In *Proceedings of ACM Object-Oriented Programming Systems, Languages and Application*, pp. 147-155, 1987.

[9] Hideaki Okamura and Yutaka Ishikawa. Object Location Contros Using Meta-level Programming. In *Proceedings of the 8th European Conference of Object-Oriented Programming*, LNCS 821, pp. 299-319, Springer-Verlag, 1994.

[10] Hideaki Okamura and Yasuhiko Yokote. Cutomizing Application Object Execution by System Object Downloading in Embedded Operating Systems. In *Technical Report* , Sony CSL, 1999.

[11] Sun Software. Java Core Reflection Specification. *http://java.sun.com/products/jdk/1.2/docs/guide/reflection/index.html*.

[12] Yasuhiko Yokote. The Apertos Reflective Operating System: The Concept and Its Implementation. In *Proceedings of Object-Oriented Programming Systems, Languages and Application*, 1992.

[13] A. Yonezawa and M. Tokoro. Concurrent Object-Oriented Systems. The MIT Press, 1987.