MobileStart：アプリケーションのシームレスな実行を支援するシステム

福田浩章[†] 山本喜一[†]

現在，情報システムの普及および，高性能な計算機の低価格化によって，我々は複数の計算機を使用して作業を行う機会が増加している．作業の効率化はそれに適したアプリケーションによって実現されるが，使用するアプリケーションの数が増加するほどそれぞれのインストールやアップデートといった作業は煩わしくなる．また，近年のアプリケーションは多機能化し，ユーザが実際に使用するのは一部の機能だけであることも少なくない．そこで本論文では，アプリケーションの中でユーザが実際に使う機能だけをネットワークを介して取得しながら実行するシステム，MobileStart の提案および実装を行う．そして，実際にアプリケーションを構築し，MobileStart を用いて実行することによってその動作と有効性を示す．MobileStart では，ユーザが実際に使用する機能だけでアプリケーションの実行を可能とし，プログラムのダウンロード，インストール，実行をシームレスに行う．そのため，計算機のリソースを有効に活用すること，また複数の計算機に同じアプリケーションをインストールする手間を軽減することが可能になる．

MobileStart: A System for Supporting Seamless Loading of an Application

Hiroaki Fukuda[†] and Yoshikazu Yamamoto[†]

As a result of the popularization of information system and the appearance of low cost computers with high performance, the opportunities that we work with two or three computers are increased nowadays. Though the efficiency of user tasks is achived by using appropriate applications, it is hard for a user to install and/or update his/her necessary applications on each computer as the number of applications they use increase. Besides, we usually use a part of functions in an application despite of it becomes to have rich functionality. This paper provides a design and implementation of a system called MobileStart that can execute an application while downloading necessary functions from network. MobileStart enables us to use an application with only our required functions and can download, install, execute programs of an application seamlessly. In addition, we show the behavior and the availability of MobileStart by implementing and evaluating an application with MobileStart. A user can use the same application via several computers without repetitive installation effort and utilize the computer resource efficiently by MobileStart.

1. はじめに

現在，我々の日常的な作業は情報システムを利用して効率良く処理することが一般化し，高性能な計算機の低価格化によって，複数の計算機を使用して作業を行う機会も増加している．我々の多様な作業がシステム化されるにともなって，使用するアプリケーションの種類は増え，それぞれのインストールやアップデート作業は煩わしくなる．これは多数の計算機を利用する場合特に顕著になる．

また，アプリケーションの多機能化にともなってインストールおよび実行に必要なリソース（メモリやハードディスクなどの記憶メディア）も増大しているが，ユーザが一度に使用するのは一部の機能に限られる[1)]．

しかし，多くのアプリケーションは実質 1 つの実行プログラムであり，分割不能であるため，すべての機能をインストールしなければ実行できない．さらに近年のアプリケーション開発では，すべての機能を 1 人の開発者が実装することは稀であり，既存ライブラリを利用することが一般的であるため，実行にはそれらのライブラリもインストールしておく必要がある．一般的にライブラリは汎用的に作成されているため，

† 慶應義塾大学大学院理工学研究科開放環境科学専攻
Graduate School of Science and Technology, Keio University

ユーザが一部の機能しか使用しないのと同様に，開発者が使用するのはその中の一部である．あるいは，機能をプラグインとして実装したアプリケーションでも，組合せの問題やインストールの煩わしさからすべてのプラグインがインストールされたものを選択し，数百 Kbyte で動くアプリケーションが結果的に数百 Mbyte にふくれ上がってしまう．このことは，個人の計算機であればそれほど問題ではないが，不特定多数で計算機を共有するとき，多くの場合ユーザごとに利用するアプリケーションは異なるため，場当たり的にインストールした結果リソースの浪費が顕著になり，管理も煩雑になる．

そこで本論文では，アプリケーションの配布作業および，インストール，実行，アップデートといった実行するための作業（以下，起動作業と呼ぶ）をネットワークを介してシームレスに行い，かつ実際にユーザが使用する機能だけでアプリケーションを実行するシステム，MobileStart の設計と実装を行う．

通常の起動作業では，ユーザがすべての機能を含んだプログラムをネットワークやメディアを介して取得し，インストールすることではじめて実行可能になる．MobileStart では，すでに一般化した常時接続のネットワーク環境に着目し，アプリケーションの起動を契機に，実際に使用する機能（プログラム）だけをネットワークを介して取得しながら実行する．この方式では，プラグイン機構を持たない分割不能なアプリケーションであっても，ユーザが使用しない機能は取得しないため，多くの機能を有し，実質その一部の機能しか使用されないアプリケーション（たとえばオフィスアプリケーション）を実行する場合，リソースの有効活用が特に期待できる．

MobileStart 自身とアプリケーションの実装には Java 言語を使用しており，Java ではネットワークを介したアプリケーション配布手段として Java Web Start [2]（以下，JWS と呼ぶ）がある．しかし，JWS で配布されるのはクラスをまとめた jar ファイルであり，そこにはユーザが実際には使用しない機能も含まれている．MobileStart では，Java の動的なクラスローディング機構を拡張することによってユーザが使用する機能に必要なクラスデータだけをオンデマンドで取得する．そして，JWS と同様に開発者は通常のアプリケーション開発方法を変更する必要はない．

以下，2 章では，アプリケーションの配布作業と起動作業に必要な処理を述べ，その中で MobileStart が支援する部分および開発者の役割について述べる．3 章では，本論文で注目したオブジェクトの生成処理と，MobileStart のアプローチを述べる．次に 4 章では，MobileStart の設計と実装および具体的な動作について述べ，5 章でアプリケーションを作成して評価する．最後に 6 章で関連研究について述べ，7 章でまとめと今後の課題について述べる．

2. アプリケーションの配布作業と起動作業

本章では，アプリケーションの配布作業について述べた後，一般的な起動作業と MobileStart を用いた起動作業の手順を示し両者の違いについて述べる．そして，MobileStart を用いて配布および起動作業を行うときの，開発者と MobileStart の役割について述べる．なお，アプリケーションを配布する計算機をサーバとし，実行する計算機をクライアントとする．また，サーバとクライアントで動作する MobileStart のシステムをそれぞれ MSServer，MSClient とする．

2.1 アプリケーションの配布作業

一般的にアプリケーションを配布するには，開発したプログラムや依存するライブラリをパッケージ化（package）した後，アプリケーションの特徴や取得法をユーザに公開する[3]．実際の配布にはネットワークや CD/DVD などのメディアが用いられる．

2.2 起 動 作 業

図 1 に一般的な起動作業 (a) と，MobileStart を用いた起動作業 (b) の流れを示す．

2.2.1 一般的な起動作業

起動作業は，一般的にアプリケーションの転送（transfer），インストール（install），実行（activate）の順序でそれぞれ独立に行われている．アップデート（update）はアプリケーションに変更が生じたとき，インストールの特別な場合として実行される[3),4)]．図 1 (a) で明らかなように，通常アプリケーションのインストール（アップデート）が終了するまでは実行できない．そして，ユーザが実際に使用する機能は予測できないため，クライアントにはすべての機能をイ

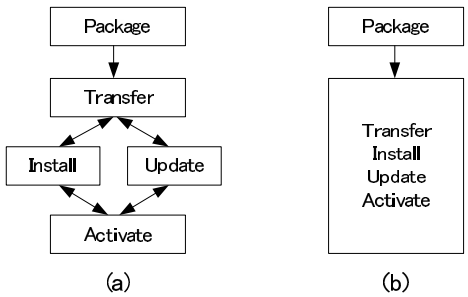

図 1 起動作業の手順
Fig. 1 Process flow of starting an application.

ンストールしておくことになる．また，この場合ネットワークはメディアを用いた配布手段（転送）の置き換えにすぎない．

2.2.2 MobileStart の起動作業

MobileStart を用いた起動作業は，ユーザによるアプリケーション実行の動作を契機に，転送，インストール，実行をシームレスに行う．この方式では，ネットワークを介してプログラムを転送しながら実行し，ユーザが必要な機能（プログラム）しか転送しない．また，アプリケーション実行中に必要になった機能は，オンデマンドでプログラムを取得して実行する．そのため，アプリケーションで必要なプログラムをすべて転送してから実行する従来の方式に対し，必要な部分しか転送せずに実行できるため，転送量の削減および，メモリやハードディスクといった記憶メディアの節約が可能である．

なお，これらの動作は MobileStart によって隠蔽されるため，ユーザは特に意識することなく必要最小限のプログラムでアプリケーションを実行できる．また，アプリケーションの実行を契機に起動作業が行われるため，実行前にアプリケーションをインストール（アップデート）しておく必要がない．

2.3 開発者と MobileStart の役割

MobileStart を用いた起動作業に必要な，開発者と MobileStart の役割について述べる．MobileStart では，MSServer が配布するアプリケーションを MSClient が実行する．MSServer と MSClient はともに複数のアプリケーションを配布および実行できるが，本論文では複数の MSServer がそれぞれ 1 つずつアプリケーションを配布し，それらを 1 つの MSClient が実行するものと仮定して説明する．

2.3.1 開発者の役割

1 章で述べたように，アプリケーションの実装には Java を使用するため，開発者は jar コマンド☆を使用してパッケージ化を行い，アプリケーションの情報を記述したデプロイファイル（5.1 節参照）とともにサーバに配置する．また，MSClient から MSServer に接続するための情報をユーザに公開する（5.2 節参照）．なお，MobileStart の特徴を生かすためにはオブジェクト指向の概念に基づき，アプリケーションの各機能を別々のクラスに分けて実現する必要がある（3.2 節参照）．

☆ 複数のファイルやディレクトリを 1 つのアーカイブファイルに統合するツール

2.3.2 MobileStart の役割

MobileStart では，アプリケーションを起動するためにユーザのクライアントに MSClient を提供する．ユーザがクライアントの MSClient からアプリケーションを実行すると，MSClient は MSServer を通じてサーバから必要なクラスデータを取得しながら，インストール（アップデート）と実行をシームレスに行う．このとき転送されるのは開発者がサーバに配置した jar ファイルそのものではなく，実際に必要なクラスデータだけであり，それらをクラスファイルとしてクライアントに保存するか否かはユーザが指定できる（6.2 節で具体例を示す）．保存したクラスデータは同一アプリケーションの 2 回目以降の実行に使用されるため，転送データの削減やネットワーク接続性のない環境でのアプリケーション実行に効果を発揮する．

また，アップデートはクラスファイルが保存されているか否かで動作が異なる．クライアントにクラスファイルが存在しない場合，アップデートとインストールの違いはない．一方クライアントにクラスファイルが存在する場合，変更されたクラスのクラスデータだけを必要に応じてサーバから転送する．なお 5.3 節で述べるように，変更のあるクラスの検出は MSServer が行うため，開発者はサーバに新しい jar ファイルを配置し，デプロイファイルを変更するだけでよい．

3. アプローチ

本章ではまず，Java 言語のオブジェクト生成処理とクラスファイルの関係について述べる．次に，一般的なユーザが最も使用する GUI アプリケーションを例に，アプリケーション設計とオブジェクトの関係を述べる．最後に，シームレスな起動作業を実現する方法について述べる．なお，オブジェクトとはクラスから生成される実体を指し，クラスを表現する場合は “クラス名” クラスと表記する．

3.1 オブジェクト生成とクラスファイルの関係

図 2 で示すように，Java では “new” を使用してオブジェクトを生成し，他のオブジェクト指向型言語と同様にメソッドを呼び合うことで処理を進めていく．

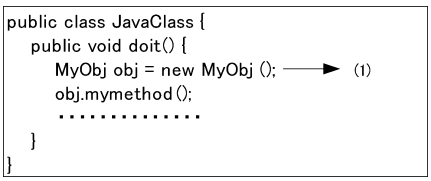

図 2 オブジェクト生成プロセス
Fig. 2 Object creation process.

Java の実行環境（以下，JVM と呼ぶ）は，実際にオブジェクトを生成するときにローカルディスクのクラスファイルからクラスデータを読み込んでオブジェクトを生成する．図 2 の例では，(1) が実行されるときに “MyObj.class” ファイルを検索し，MyObj オブジェクトを生成する．いい換えると，(1) を実行しない限り “MyObj.class” ファイルは必要ない．

3.2 アプリケーション設計とオブジェクトの関係

GUI アプリケーションの処理形式はイベント駆動型であり，ユーザがボタンなどのイベントソースオブジェクト（以下ソースオブジェクトと呼ぶ）を操作することでイベントが発生し，イベントリスナーオブジェクト（以下リスナーオブジェクトと呼ぶ）がそのイベントを処理する．**図 3** では 1 つのメニューに属する複数のボタン（ソースオブジェクト）に，同一のリスナーオブジェクトを登録し，リスナーオブジェクトがソースオブジェクトを判定してイベントに対応するオブジェクトを生成した後に処理を委譲している．このように，オブジェクト指向（型言語）では役割ごとにクラスを定義し，1 つのコンポーネントとして扱って処理を進めることが一般的である．

アプリケーションがこのように設計されている場合，3.1 節で述べたように実際に機能が使用されるまでそれに対応するオブジェクトおよびクラスファイルは不要であるが，どの機能を使用するかはユーザによって異なり，開発者が予測することはできない．したがって，現状のインストールやアップデートにはアプリケーション実行前にすべてのクラスファイルを計算機に配置しておく必要がある．

3.3 シームレスな起動作業の実現

Java では，JVM が java コマンドで指定されたクラスの main メソッドを実行してアプリケーションを起動する．JVM は，クラスデータからオブジェクトを生成するために 4.1 節で述べるクラスローダを使用するが，Java 標準のクラスローダはローカルディスクのクラスデータしか読み込むことができない．そこで MobileStart では，MSClient からアプリケーションを起動する（6.2 節で具体例を示す）ことによってオブジェクトの生成過程に割り込み，クラスローダを拡張することによって必要なクラスデータをネットワークを介して取得する．そのため，3.2 節で述べたように，機能ごとにクラスを分離し，必要に応じて生成する設計にすることにより，不要なクラスデータ（実行されない機能）の転送を防ぎ，MobileStart の特徴を生かすことができる．

なお 4.1 節で述べるように，個々のアプリケーションでクラスデータをオンデマンドに取得することもできるが，クラスローダの制約のため，アプリケーションレベルで行う方式と MobileStart で行う方式は根本的に異なる．

4. MobileStart の実現

4.1 クラスローダ

図 4 (a) に示すように，Java では複数のクラスローダが木構造を形成し，ロードするクラスに応じて使い分けられる．また，新たにクラスローダを定義することができる．したがって，図 4 (b) のように記述することによって，必要なクラスデータをオンデマンドに（外部デバイスやネットワークから）取得して図 3 (*) で示した Func1 オブジェクトを生成することができる．しかし Java では，異なるクラスローダでロードされたオブジェクトは同じクラスデータから生成されても別オブジェクトとして扱われる．したがって，図 4 (b) の (1) では独自クラスローダ（以下，MyLoader と呼ぶ）で生成した obj に対し，キャスト（型変換）する Func1 には暗黙のうちに JVM のクラスローダ（以下，Application1 と呼ぶ）が使用されるため例外が発生する．これを回避するには，図 4 (b) の (2) のように Application1 でインタフェース（Func1Int）を生成し，MyLoader で生成したオブジェクトをそのインタフェースにキャストする．この結果，JVM はすべてのオブジェクトを Application1 で生成したものと見なすことができ，例外も発生しない．しかしこの方法では，MyLoader で生成するオブジェクトには必ずイン

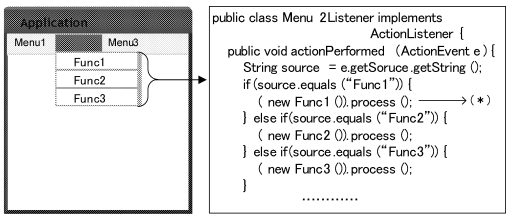

図 3 アプリケーションの設計
Fig. 3 Design of an application.

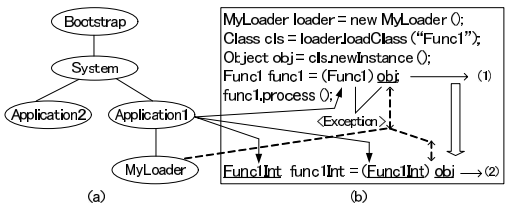

図 4 クラスローダとオブジェクトの関係
Fig. 4 Relationship between classloaders and objects.

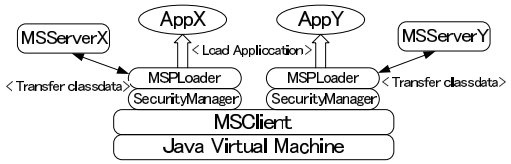

図 5 MobileStart のアーキティチャ
Fig. 5 Architecture of MobileStart.

タフェースが必要になり，あらかじめクライアントに配置しておかなければならない．そのため，MyLoader で生成するオブジェクトが多い場合それぞれに専用のインタフェースを用意するか，1 つのインタフェースに制約された実装が必要になる．また，クラスデータを取得するプログラム自身もクライアントに配置しておく必要がある．

したがって，各アプリケーションでクラスローダを拡張して用いる場合，アプリケーションごとに専用インタフェースやクラスローダを事前に計算機に配置しておく必要がある．そのため，それらを事前に配置することなく，クラスローダの制約に違反せずにアプリケーションを起動するには，アプリケーションとは独立してクラスデータの取得やオブジェクトを生成する実行環境，すなわち MSClient が必要となる．

4.2 MobileStart の設計

本章では MobileStart のアーキテクチャとセキュリティについて述べる．

4.2.1 MobileStart のアーキテクチャ

図 5 に MobileStart のアーキテクチャを示す．3.3 節で述べたように，ユーザが MSClient からアプリケーションを起動すると，MSClient は実行するアプリケーションごとに専用のクラスローダ（以下，MSPLoader と呼ぶ）を割り当て，JVM のクラスローダに代わってそれぞれの MSPLoader にオブジェクトの生成処理を委譲する．MSPLoader は，必要に応じてアプリケーションのクラスデータが配置されているサーバから MSServer を通じてクラスデータを取得する．これによって，1 つのアプリケーションに必要なオブジェクトはそれ専用の MSPLoader によって生成されるため，4.1 節で述べたクラスローダの制約に違反することなくアプリケーションを起動することができる．また，アプリケーションでクラスデータを取得するプログラムを用意し，前もって配置しておく必要はない．さらに，アプリケーションごとにサーバが異なる場合や，5.3 節で述べるアップデートを行う場合も，各 MSPLoader がサーバやバージョン情報を管理するため，開発者は通常の手法で実装すればよい．

なお MSPLoader は，アップデートされた場合を除いてクライアントに保存されているクラスデータを優先的に使用する．

4.2.2 セキュリティ

4.2.1 項の方式では，クラスデータ（プログラム）を転送して実行するため，悪意のあるプログラムによるクライアントリソースへのアクセスや，ネットワークの使用を制限するなどのセキュリティを考慮する必要がある．MobileStart では，MSClient から起動されるアプリケーションを Sandbox で実行する[5),6)]．MobileStart の Sanbdox は Java のセキュリティマネージャ（SecurityManager）☆を使用し，アプリケーションのファイルアクセスやネットワーク使用を制限する．図 5 で示すように，MSClient は SecurityManager がインストールされた MSPLoader をアプリケーションに割り当てる．ただし，ネットワークやファイルへのアクセスを要求するアプリケーションに対しては，起動時にユーザの指定によって SecurityManager をインストールせずに MSPLoader を割り当て，アプリケーションにすべてのアクセス権を与えることができる．この SecurityManager のポリシを変更することによって，アプリケーション単位でアクセス権をより詳細に制御できるが，現在はアプリケーションごとにアクセス権の有無だけを実装している．

5. MobileStart の動作

本章では，MobileStart を用いた配布作業と起動作業を，パッケージ化，アプリケーションの実行，アップデートの順で述べる．なお，例として示すアプリケーションの名前は “SampleApp” で，“foo.bar.Sample” クラスの “init” メソッドで起動するものとする．

5.1 パッケージ化

開発者はアプリケーションに必要なクラスファイルをまとめた jar ファイルと，**図 6** (a) に示すデプロイファイルを “アプリケーション名．xml” という名前で作成する．デプロイファイルは (i) 部分で示す (1) アプリケーション名，(2) MSServer のベースディレクトリ，(3) 現在のバージョンといった一般的な情報と，(ii) 部分で示す (5) 実行クラス名，(6) 実行メソッド名，(7) JVM のバージョン，(8) jar ファイル群といったバージョンごとに異なる情報で構成され，(ii) 部分を追加して複数のバージョンを記述することもできる．次に，このデプロイファイルの記述内容に基づき，図 6 (b) のように各要素をサーバに配置する．

☆ 信頼できないコードからローカルのファイルアクセスや，ネットワークの利用を制限する仕組み

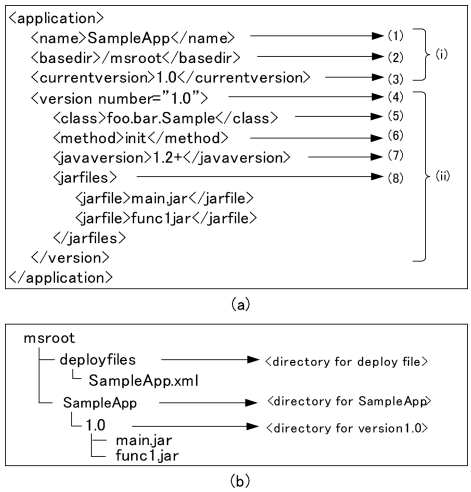

図 6 デプロイファイルとサーバのディレクトリ構造
Fig. 6 Deployfile and directory structure on a server.

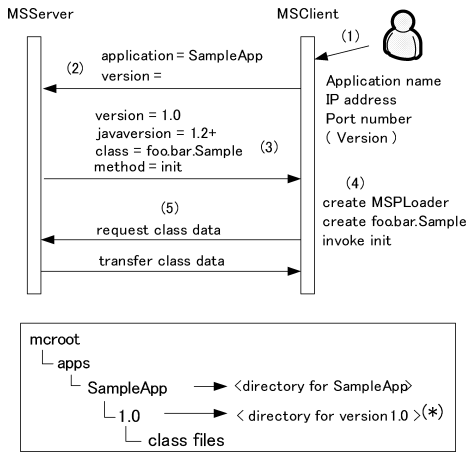

図 7 アプリケーションの起動
Fig. 7 Start on an application.

5.2 アプリケーションの実行

図 7 にアプリケーションの起動から終了までの手順を示す．(1) ユーザはアプリケーションが配置されたサーバの IP アドレス，MSServer のポート番号，そしてアプリケーション名，バージョンといった開発者が公開するアクセス情報を MSClient に入力してアプリケーションを起動する．アクセス情報はバージョン以外必須であり，入力した情報は MSClient に保存される．(2) MSClient は MSServer に接続してアプリケーション名とバージョン（任意）を送信する．なお，アクセス情報の公開は開発者が MobileStart 以外の手段（Web やメール）を利用して行う．(3) MSServer は，要求されたアプリケーションのデプロイファイルから対応するバージョンの図 6 (a)(ii) 部分を jar ファイル群（図 6 (a) の (8)）を除いて MSClient に送信する．バージョンが空の場合は現在のバージョン（図 6 (3)）となる．(4) MSClient は，実行するアプリケーション専用の MSPLoader を生成してアクセス情報を埋め込んだ後，受信した情報をもとにオブジェクト（foo.bar.Sample）を生成してメソッド（init）を実行する．また，バージョンを指定せずに実行した場合，現在のバージョンがアクセス情報に追加され，2 回目以降はそのバージョンが使用される．(5) 4.2 節で述べたように，MSPLoader はオブジェクトの生成処理に割り込み，“foo.bar.Sample” をはじめとする必要なクラスデータをサーバから取得する．また，ユーザがクラスデータの保存を指定した場合は図 7 に示す階層構造に従ってクラスデータを保存する．図 7 中の “mcroot” は MSClient がインストールされているディレクトリを示す．

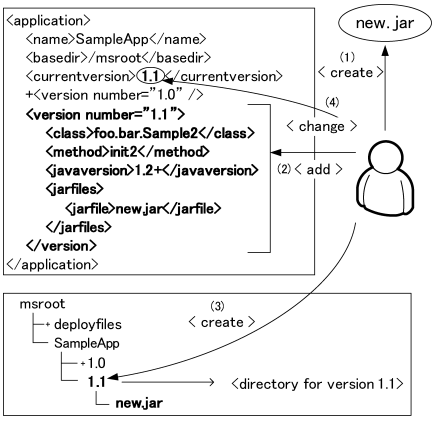

図 8 開発者のアップデート作業
Fig. 8 Update process for a developer.

5.3 アップデート

アップデートにおける開発者の作業手順を**図 8** に，MSServer と MSClient の動作を**図 9** に示す．開発者はアプリケーションを変更し，新しく jar ファイルを作成する（図 8 (1)）．変更後のアプリケーションはこれらの jar ファイルだけで動作しなければならない．次に，バージョンを上げてデプロイファイルに追加し（図 8 (2)），jar ファイルを適切な位置に配置する（図 8 (3)）．最後に，図 6 (3) で示した現在のバージョンを変更する（図 8 (4)）．

次に，ユーザ側アプリケーションのアップデートは，起動時に MSClient から MSServer に送られるバージョンと現在のバージョンが異なる場合に行われる（図 9 (1)）．要求されたバージョンが古い場合，MSServer は MSClient にアップデートメッセージを送り，MSClient がユーザにアップデート承認用のボタンを提示する（図 9 (2)）．これ以降，クラスファイ

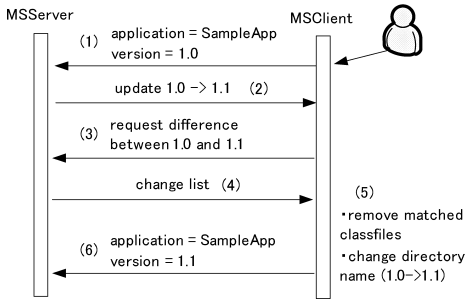

図 9 アプリケーションのアップデート手順
Fig. 9 Update steps of an application.

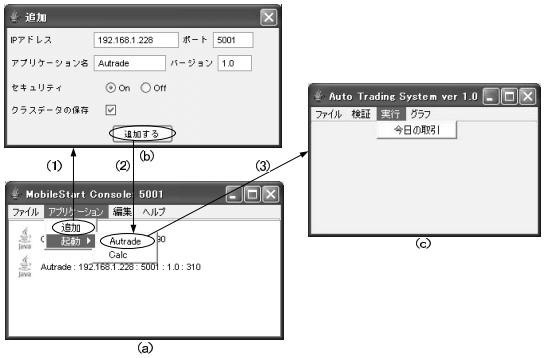

図 10 アプリケーションの実行例
Fig. 10 An execution example of an application.

ルが保存されていない場合といる場合で動作が異なる．前者の場合，アップデートが承認されると新しいバージョンで 5.2 節のように動作してアプリケーションを起動する．一方後者の場合，アップデート承認後に MSClient が MSServer に変更されたクラス一覧を要求する（図 9 (3)）．MSServer は，旧バージョンと新バージョンの jar ファイルをクラスファイルのハッシュ値を利用して比較し，変更があるクラスの一覧を MSClient に送信する（図 9 (4)）．MSClient はクライアントに存在し，かつ受信したクラス一覧に含まれるクラスファイルを削除した後，クラスファイルが保存されているディレクトリ名（バージョン名になっている）を変更する（図 9 (5)）．これ以降は新バージョンで 5.2 節のように動作し，変更されたクラスファイルはすでに削除済みのため必要に応じてサーバから取得する．

6. アプリケーションの実行と評価

本章では，MobileStart を用いてアプリケーションを実行し，動作を確認するとともに基本性能を評価する．

6.1 サンプルアプリケーション

MobileStart の動作を確認するため，株式自動売買システム（以下，Autrade と呼ぶ）を実装した．Autrade は，データの取得，複数の検証アルゴリズム実行，グラフ表示，自動取引など，合計 10 個の機能を別々のクラスで実装している．また，3.2 節で述べたように，メニューごとにリスナーオブジェクトを作成し，イベントソースによって処理を分岐した後に対応するオブジェクトを生成して処理を委譲する．そのため，機能に対応するオブジェクトは実行されない限り不要であり，クラスデータも必要ない．なお，Autrade はデータの取得に HTMLParser [7]，検証結果のグラフ表示に JFreeChart [8] および，JCommon [9] といった外部ライブラリを使用している．

6.2 実 行 例

MSClient から Autrade を起動する様子を**図 10** に示す．(1) ユーザは MSClient（図 10 (a)）の追加ボタンから図 10 (b) を表示し，(2) アクセス情報および，セキュリティやクラスデータを保存するか否かを選択して MSClient にアプリケーションを追加する．すると，(3) 追加された “Autrade” がメニューに表示され，ユーザはそのメニューをクリックして Autrade（図 10 (c)）を起動する．以降，MSClient は実行する機能に必要なクラスデータを MSServer から取得する．

6.3 評 価

MobileStart の基本性能を示すため，Autrade の起動にかかる時間（以下，起動時間と呼ぶ）とデータ転送量を測定する．起動時間の測定は，アップデートを行わない場合（ST）と行う場合（STU）の 2 種類行う．次に，外部ライブラリを使用する 2 つの機能（データ取得とグラフ表示）を実行し，それぞれに必要なクラス数，データ転送量，実行時間を測定する．

比較対象として，JWS を用いる場合の起動時間とデータ転送量を測定する．また，Autrade 単体で先に述べた機能の実行時間を示す．なお，測定には j2sdk1.4.2_12 がインストールされ，100 Mbps で接続された 2 台の計算機（Pentium4 1.6 GHz，1 GB メモリ）を使用した．

以下，**表 1** に MobileStart を用いた起動と機能実行時間，**表 2** に JWS を用いた起動時間，**表 3** に Autrade と使用したライブラリのサイズ，クラス数を示し，起動と機能実行時間の評価，クラス数とデータ転送量の評価，総合評価の順に述べる．

6.3.1 起動と機能実行時間

表 1 で示すように，MobileStart を用いた Autrade の起動時間（TCP コネクションの確立，図 7 (1)〜(4) の処理，クラスデータの要求と転送，オブジェクト生

表 1 MobileStart を用いた起動と機能実行時間
Table 1 Time for starting an application and executing functions using MobileStart.

	起動（ST）	起動（STU）	データ取得	グラフ表示
時間 [単体]（ms）	310	1,343	1,413[822]	1,813[1192]
サイズ（KB）	21.3	21.3	206.8	678.4
クラス数（個）	13	13	73	160

表 2 Java Web Start での起動時間
Table 2 Starting time with Java Web Start.

サイズ（kbyte）	1	2,204
時間（ms）	1,853	5,275

表 3 Autrade とライブラリの詳細
Table 3 Details of Autrade and each library.

	HTMLParser	JCommon	JFreeChart	Autrade
サイズ（KB）	617	306	1,075	261
クラス数（個）	482	232	549	97

成とメソッド実行の合計時間）は 310 ms であり，64 個のクラスが変更されたバージョンへのアップデートを含めた場合，1,343 ms であった．5.3 節で述べたように，アップデート時は旧バージョンと新バージョンの jar ファイル群に含まれる，すべてのクラスファイルのハッシュ値を算出して比較するため，クラス数が増えるほどアップデートには時間がかかる．しかし，現実的にアプリケーションの頻繁（数時間，数日単位）なアップデートは起こらないため，実用上問題ない．

一方，JWS での起動時間は 5,275 ms であり，MobileStart を用いた場合の約 17 倍であった．しかし，1 kbyte という小さなアプリケーションであっても，起動に 1,853 ms かかるためデータ転送量と生成するオブジェクトの個数やメソッド実行回数の増加が起動時間に与えた影響は 3,400 ms 程度であった．

MobileStart の方式は，必要なクラスデータをクラスファイル単位で要求/転送を行うため，起動時に必要なクラス数が増えた場合，JWS よりも起動に時間がかかる可能性がある．しかし，必要な機能をオンデマンドに取得して実行できる MobileStart の特長を生かし，起動に必要なクラス数を減らすことで，起動時間を短縮することができる．なお GUI アプリケーションでは，機能を実行する契機となるソースオブジェクトは起動時に生成する必要があるが，それらのオブジェクトには一般的にクライアントにすでに存在する Java の標準ライブラリを用いるため，それらを使用する限りクラスデータの要求/転送は発生しない．

次に，機能の実行時間においても，Autrade 単体で実行する場合に比べデータ取得に 600 ms，グラフ表示に 800 ms 程度多く時間がかかった．これも起動時間同様，オンデマンドなクラスデータの要求/転送の影響であるが，このオーバヘッドが生じるのは最初に機能を実行するときだけである．また，本論文で想定する多機能なアプリケーションでは，ユーザがすべての機能を一度には実行しないため，オーバヘッドも分散され実用上問題ない．なお，今後 MSServer にクラスデータのキャッシュ機構を実装することで，起動や機能実行時間の向上が期待できる．

6.3.2 クラス数とデータ量

表 3 に示すように，Autrade と参照するライブラリのデータ量は，合計 2,259 KB（非圧縮時 8,790 KB）であり，クラス数は 1,360 個であった．MobileStart を用いる場合，表 1 に示すように起動だけであれば 13 個のクラスファイル，合計 21.3 KB（非圧縮）のクラスデータで実行が可能であり，必要なクラス数，データ量ともに全体の 1%程度で済む．またデータ取得機能では，HTMLParser 全体で 617 KB のサイズ，クラス数 482 個のうち，実際に使用するのは 206.8 KB，73 個のクラスであり，それぞれ全体の約 30%のサイズ，約 15%のクラス数で実行可能である．同様にグラフ表示で実際に使用するのは 618.4 KB，160 個のクラスであり，JCommon，JFreeChart 全体（1,381 KB，クラス数 781 個）のそれぞれ約 50%のサイズ，約 20%のクラス数であった．ライブラリの中で実際に使用する部分は，それを使用するアプリケーションに依存するが，一般にライブラリは汎用的に作られているため，ユーザがアプリケーションの一部の機能しか使用しないのと同様に，開発者がライブラリのすべての機能を使うことは稀である．したがって，既存ライブラリを多用する多機能なアプリケーションになればなるほど，

MobileStart を用いることで節約できるリソースは大きくなる．

6.3.3 総合評価

これまでの評価から，MobileStart は本論文で想定する多機能なクライアントアプリケーションの実行に効果を発揮する．また，汎用ライブラリやプラグインを多用するアプリケーションでも，MobileStart によってプログラムの最小単位であるクラスファイルごとに取得するため，開発者主導ではなく，ユーザ主導で必要なプログラムの取得と実行が可能である．このような実行方式は MobileStart が提供するため，ユーザは 150 KB 程度の MSClient をインストールするだけでよい．また，アプリケーションの配布にも 210 KB 程度の MSServer を用意すればよく，開発上の制約もない．そして，アップデートにも新バージョンのアプリケーションを jar 形式で準備するだけでよく，差分を気にする必要はない．また，3.2 節で述べたアプリケーションの設計を考慮することにより，起動時間の向上が期待できる．

7. 関連研究

1 章でも述べたように，Java ではアプリケーションをダウンロードして実行する Java Web Start [2] がある．このシステムでは，クライアントにインストールされたアプリケーションマネージャが必要な jar ファイルをダウンロードして実行する．また，1 つのアプリケーションを複数の jar ファイルに分割して管理することもできるため，通常は必要ないヘルプなどの機能を分割しておくことにより転送量を削減できる．そして，クラスファイル単位で更新を行うバージョン管理機構も備えている．しかし，jar ファイル分割はパッケージ化のポリシや既存ライブラリ単位など，開発者の視点，あるいは実装の都合で行われ，ユーザが実際に使用する機能と分割された jar ファイルとは必ずしも一致しない．そして，jar ファイルに含まれるクラスが 1 つでも必要になると，jar ファイル全体がダウンロードされてしまう．

Maven [10] では，依存するライブラリを指定しておくことによってリモートレポジトリから自動的に取得してインストール（アップデート）を効率化しており，One-Jar [11] では複数の jar ファイルを入れ子構造にして 1 つにまとめ，確実にインストールを行う手段を提供している．これらのツールはプロジェクト管理や，実行可能なアプリケーションの自己完結を目的としているため，ユーザが実際に使用するか否かにかかわらずすべての jar ファイルがダウンロード，あるいは入れ子構造に含まれる．

MobileStart では，ユーザが使用する機能に必要なクラスデータを，プログラムの最小単位であるクラスファイルごとに取得しながらオブジェクトを生成するため，jar ファイル分割に関係なく，ユーザの視点から必要最小限のクラスデータでアプリケーションを実行することができる．

一方，CCM [12] などの分散オブジェクト（コンポーネント）技術を導入し，遠隔オブジェクトに必要な処理を依頼することによって機能を分割することができる．この方法では，実行結果だけをやりとりするため，クライアントには遠隔オブジェクトの機能を呼び出すプログラムだけあればよい．しかし，遠隔オブジェクトを呼び出すプログラムはアプリケーションごとに異なり，複数のアプリケーションを複数の計算機で実行するときには，そのプログラムをすべての計算機にインストールしておく必要がある．また，機能を実行するたびにネットワーク接続が必要になり，機能を呼び出すプログラムと機能自身のプログラムの両方を開発しなければならない．

MobileStart では，MSClient がインストールされた計算機であれば任意のアプリケーションを実行でき，一度転送した機能はネットワークの接続性に依存せずに実行可能である．また，通常のアプリケーション開発方法を変更する必要はない．

8. まとめと今後の課題

本論文では，アプリケーションの配布，インストール，実行，アップデートといった作業をネットワークを介してシームレスに行うシステム，MobileStart の提案と実装を行った．

MobileStart では，オブジェクトの生成処理に割り込み，アプリケーションの中でユーザが実際に必要とする機能だけをネットワークを介して取得しながら実行する．そのため，新規アプリケーションのインストールが許可されていない不特定多数が利用する計算機でも，MobileStart がインストールされ，ネットワークに接続されていれば任意のアプリケーションを必要な機能だけで実行することができる．

本論文では，実際にアプリケーションを MobileStart で起動して動作を確認し，データ転送量と起動時間を Java Web Start と比較して有効性を確認した．MobileStart は GUI アプリケーションだけでなく，Java で記述されていればアプリケーションの種類によらず適用することができるが，サーバアプリケーションのようにすべてのオブジェクトを生成しなければ動

作しないものでは，従来の実現方法との相違はない．

現在の実装ではクラスデータを転送時に圧縮しておらず，検索もキャッシュ機構を持っていない．そして，J2SE 上で実装されているため携帯端末上では動作しない．今後これらの機能を含めシステム全体を J2ME で実装し，URC[13] などの携帯端末で実行するアプリケーションの基盤技術としての応用を考えている．また，ネットワークに接続できない環境で使用する場合に備え，あらかじめ必要なクラスデータをすべて取得する機能を MSClient に追加し，ユーザの用途に合わせた利用法を提供する予定である．

最後に，本論文の提案方式は動的なローディング機構を持つ環境であれば適応可能であるため，.Net Framework への対応を予定している．

参 考 文 献

1) Mayer, J.: Graphical User Interfaces Composed of Plug-ins, *Proc. GCSE Young Researchers Workshop 2002*, pp.25–29 (2002).
2) Sun Microsystems Inc.: Java Web Start Technology. http://java.sun.com/products/javawebstart/index.jsp
3) Carzaniga, A., Fuggetta, A., Hall, R.S., van derHoek, A., Heimbigner, D. and Wolf, A.L.: A Characterization Framework for SoftwareDeployment Technologies, Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado (Apr. 1998).
4) Lestideau, V., Belkhatir, N. and Cunin, P.-Y.: Towards Automated Software Component Configuration and Deployment, *8th International Conference on Information Systems Analysis and Synthesis* (2002).
5) Gong, L., Mueller, M., Prafullchandra, H. and Schemers, R.: Going beyond the sandbox: An overview of the new security features in the Java Development Kit 1.2, *Proc. USENIX Symposium on Internet Technologies and Systems*, USENIX Association (1997).
6) Gong, L.: Secure Java Class Loading, *IEEE Internet Computing*, Vol.2, No.6, pp.56–61 (Nov./Dec. 1998).
7) HTMLParser. http://htmlparser.sourceforge.net/
8) JFreeChart. http://www.jfree.org/jfreechart/
9) JCommon. http://www.jfree.org/jcommon/index.php
10) The Apache Software Foundation: Apache Maven. http://maven.apache.org/
11) Tuffs, P.S.: Deliver Your Java Application in One-JAR. http://one-jar.sourceforge.net/
12) Object Management Group: CORBA Component Model. http://www.omg.org/technology/documents/formal/components.htm
13) Universal Remote Console Consortium: Universal Remote Console. http://www.myurc.com/

(平成 18 年 4 月 26 日受付)
(平成 18 年 11 月 2 日採録)

福田 浩章（正会員）

1998 年慶應義塾大学理工学部計測工学科卒業，計算機科学専攻修士．現在，慶應義塾大学大学院開放科学専攻後期博士課程在籍．主としてユビキタス環境における分散アプリケーションおよび，プラットフォームの研究に従事．日本ソフトウェア科学会会員．

山本 喜一（正会員）

1969 年慶應義塾大学工学部管理工学科卒業，管理工学専攻修士，工学博士（1987 年，慶應義塾大学）．現在，慶應義塾大学理工学部情報工学科助教授．ソフトウェア科学，特にシステムの動的適合，ソフトウェア工学，ヒューマンインタフェース等の研究に従事．ACM，IEEE，情報システム学会，日本ソフトウェア科学会，電子情報通信学会，日本シミュレーション学会各会員．情報システム学会理事，編集委員長．