Java™ 分散オブジェクトを利用したアリケーション・インストーラの開発

藤生 正樹　箱崎 勝也

電気通信大学大学院情報システム学研究科

概要

本論文では、Java™ 分散オブジェクトを利用したマルチプラットフォーム対応のアプリケーション・インストーラである“アプリケーション・ナビゲータ“を提案する。近年、異なる OS の混在するシステムは増加しており、ユーザのアプリケーションのインストール作業負担は大きくなっている。“アプリケーション・ナビゲータ“は、OS に依存しない共通の GUI によって、ユーザのインストール作業を簡易化するものであり、製品市場でリリースされているマルチプラットフォーム対応インストーラ製品の欠点を解決したものでもある。また、既存のインストーラ製品の機能を再利用することで、開発者負担の低減も行っている。

Development of Application Installer using Java™ Distributed Object

Masaki FUJIU, Katsuya HAKOZAKI

Graduate School of Information Systems, University of Electro-Communications

Abstract

This paper proposes an application installer corresponding multi platform utilizing Java distributed object, called "Application Navigator".

In recent years, computer systems using different operating system (OS) has increased and users' workloads of application installing become heavier. "Application Navigator" gives users easy installing with common GUI independent of OS platform and solves defects in released multi platform installer products on the market. And "Application Navigator" has also decreased developers' workloads by recycling the functions of released installer products.

1. はじめに

コンピュータ・システムのオープン化が叫ばれて久しいが、現在では、オペレーティングシステム(以下OSと記す)の異なるコンピュータの混在したシステムを基幹業務に適用する例も珍しくない。ビジネス領域においては、Sun Microsystems社、Hewlett Packard社、NECなどのベンダが提供する基幹UNIXシステム、Microsoft社のWindows、さらに近年ビジネス領域で注目されているLinuxなど、プラットフォームの異なるOSによって構成されるシステムは、より複雑化してきている。

こうした流れの中で、UNIXサーバ領域においては、コンソリデーション技術というものがキーワードになりつつある。これは、システムのオープン化が進み、サーバへの負荷低減のために分散環境で処理してきた業務を、1台(クラスタリングや処理能力によっては複数台)の高性能UNIXサーバに統合して行うというものである。しかし、システムのコンソリデーションもすべてのシステムに最適なものというわけではなく、信頼性やプラットフォームに固有の機能を利用したい場合など、異なるOSの混在するシステムそのものは今後も利用されていくものであるが、近年の分散システムの管理、運用の複雑化の中で、コンソリデーションによるシステムの簡易化技術が注目されてきている点は興味深い。

こうした環境下で、アプリケーション・ソフトウェア(以下アプリケーションと記す)のインストール作業はOSごとにインストール・ロックの解除、インストール・コマンドの実行手順が異なり、特にUNIX対応アプリケーションのインストール作業は、Windows用のアプリケーションと比較すると、専門的な知識が必要とされる場合が多い。また、アプリケーションのインストール作業は非定期的に行われるため、作業手順や知識を蓄積しにくいという一面を持っており、作業者の負担はさらに大きくなりつつある。

そこで本研究では、インストール作業の簡易化により作業者負担の軽減を図る、マルチプラットフォーム対応のアプリケーション・インストーラ(アプリケーション・ナビゲータと呼ぶ)を提案する。これは、マルチプラットフォーム対応、ネットワーク言語の特性を持つJavaを利用して、OSプラットフォームに依存しない共通のインタフェースと作業手順でのアプリケーションのインストールを行うものであり、Java分散オブジェクト技術を利用して、データを管理するサーバ部(インストール・サーバ)とインストールを実行するクライアント部とで構成される。これは、インストール・サーバから複数の異なるプラットフォームのクライアントに対して同様のインタフェースでインストール作業を可能とするものである。

また、製品市場では、マルチプラットフォーム対応のものも含めて、ユーザの作業支援を目的としたインストーラ製品[1]が多数リリースされている。このような製品は既に動作実績のあるプログラムであり、動作テスト等十分に実施されているはずである。そこで、本研究では、これら既存のインストーラ製品の機能を再利用することで、開発者の負担の軽減(開発工数の低減)、動作信頼性の向上を図るとともに、その開発手法を検証する。

NEC製UNIX OS、UX/4800には、"48Startup[2]"というアプリケーションのインストール作業をGUIを用いて行うことのできる製品がリリースされており、本研究では、C言語で作成されているこの"48Startup"の各種機能を、Java言語で作成されるアプリケーション・ナビゲータに実装する。

2. 既存のマルチプラットフォーム対応のインストーラの問題点

現在、"InstallShield Java Edition"(InstallShield Software Corporation)、"InstallAnywhere"(Zero G Software)、"ブロッス"(テンアートニ(株))などのJava言語を利用した、主要UNIX OSとWindowsをサポートしたマルチプラットフォーム対応のインストーラ製品がリリースされている。また、マルチプラットフォーム対応ではないが、"InstallStudio"(Bunka Orient

Corporation) 、 "InstallMaker" (Wise Solutions, Inc.)、"Installer VISE" (MindVision Softwar)などのユーザの作業負担を軽減させるインストーラ製品も多数リリースされている。これらインストーラ製品の特徴は、GUIを用いてアプリケーションのインストール作業を簡易化させ、インストール・パッケージ(インストールするためのアプリケーション・データ)にインストール処理スクリプトを付加した形式となっている点である。

しかし、これらの製品を UNIX 上で利用するにはいくつかの問題がある。

(1) UNIX OS ではインストール履歴が残らない

Sun Microsystems 社の Solaris や UX/4800 などはアプリケーションのインストール・データは、インストール処理スクリプトと共に CPIO 形式でパッケージ化されており、Hewlett Packard 社の HP-UX などは tar 形式でパッケージ化されているなど、ベンダによりその形式は異なる。しかし、上記のマルチプラットフォーム対応のインストーラ製品は、各 OS に合わせたパッケージ形式を採用しているわけではなく、独自のパッケージ形式を採用している。このため標準のインストール処理と異なり、インストール履歴を独自に管理しているなど、本来 OS で管理すべきインストール情報を残さないことによる不具合が発生する可能性がある。

(2) 既存のインストール・パッケージを利用できない

既に OS 標準のインストール・パッケージとして作成、リリースされているアプリケーションを上記マルチプラットフォーム対応のインストーラ製品に適用する場合、パッケージを再作成する必要があり、作業工数が再度発生してしまう。

3. アプリケーション・ナビゲータの構成

アプリケーション・ナビゲータの概要を図 1 に示す。本システムは、アプリケーション・データを管理するサーバ部(インストール・サーバ)とインストール・データの

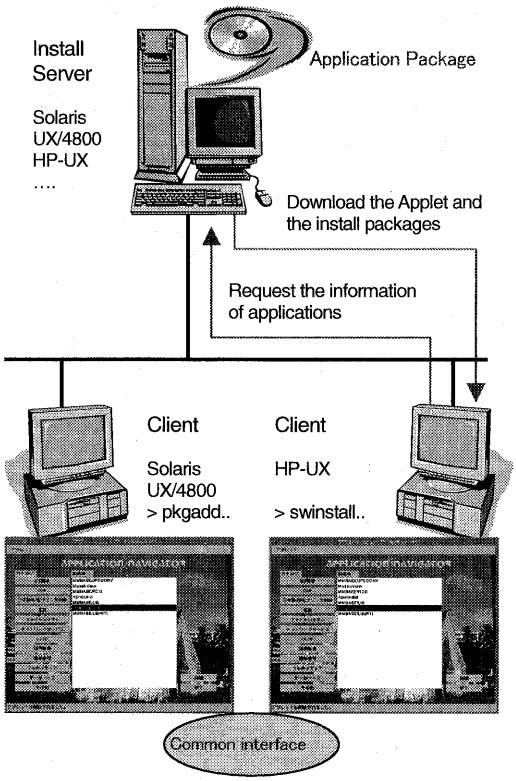

図 1 システム概要図

要求、インストールを実行するクライアント部から構成される。また、アプリケーション・ナビゲータは、Solaris 2.6、HP-UX 11.0、UX/4800 R13.1 (ただし、本 OS では Java2 がリリースされていないため、非分散環境での評価に利用)上で評価を行う。

ユーザは、プラットフォームの異なる OS から Java アプレットで作成されたクライアント・アプリケーションを通じてインストール・サーバへアクセスし、アプリケーションの情報を入手、次いで、インストールを選択すると、再度インストール・サーバからインストール・パッケージをダウンロードし、それぞれの OS に沿ったインストール処理が実行される。この一連の動作は、ユーザから見た場合、OS に依存せず、共通のインタフェースと手順で実行される。

次に、図 2 に実装図を示す。アプリケーション・ナビ

ゲータを、アプリケーションのデータを管理し、クライアントからの要求に応じて、必要なデータを返す"アプリケーション・データ検索・管理機能"とクライアント側でインストール処理を実行する"アプリケーション・インストール機能"に分けて、それぞれについて説明する。

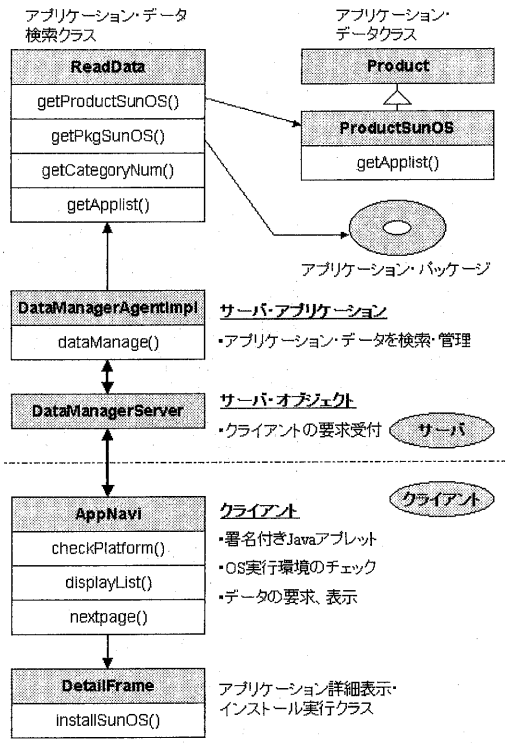

図2 アプリケーション・ナビゲータ実装図

3.1. アプリケーション・データ検索・管理機能

インストール・サーバは、名称、概要などが記述された CSV 形式のアプリケーションのデータを保持している。クライアント(AppNavi)は、Java2 にバンドルされた CORBA 互換の ORB、JavaIDL を用いて、分散オブジェクトとして実装されたサーバ上にある検索処理を行うオブジェクト(DataManagerAgentImpl)にアプリケーションのデータを要求する。この検索オブジェクトは、クライアントの要求に対して、クライアントの OS 環境に応じたデータを検索し、その結果を引き渡す Java アプリケーションである。この時インストール・サーバは、クライアントから OS バージョンなどの情報を受け取り、この情報を蓄積している。これは、将来、あるアプリケーションのパッチがリリースされた場合など、どのマシンにパッチ・プログラムを適用したらよいかを判別できるようにするためである。

検索・管理機能は Java アプリケーションとして実装されるが、同時に UX/4800 用にリリースされている GUI インストーラ製品である"48Startup"のデータ検索機能を JNI(Java Native Interface)を用いたネイティブ・メソッドとして実装した。"48Startup"は、X Window 上で実行可能なインストーラ製品であり、アプリケーションの検索、インストールロックの解除、インストールの実行を行うことが可能で、C 言語で作成されている。

本機能の再利用の目的は、以下の 3 つである。

(1) 開発工数の低減
新規に機能を作り込む必要がない

(2) 信頼性の向上
すでに動作実績のあるプログラムであるため、信頼性が高く、新規バグの作り込みを防止

(3) 実行速度の向上
ネイティブコードで実行されるため、Java の実行コードに比べ、高速である

そして、Java のオブジェクトと検索データのやり取りを行うための関数(libsearch.so)を C 言語で作成し、もとの"48Startup"のプログラムには手を加えずに実装している。これは、すでに実績のあるソース・コードに手を加えて、バグを作り込んでしまう事を防ぎ、デバッグも容易するためである。図 3 にネイティブ・メソッドでの実装図を示す。

上述のように、検索・管理機能は、Java アプリケーション版と C 言語によるネイティブ・メソッド版の両方を実装しているが、オプションで切り替えられるようになっている。ただし、ネイティブ・メソッド版は、Solaris と UX/4800 の場合に実行可能としている。

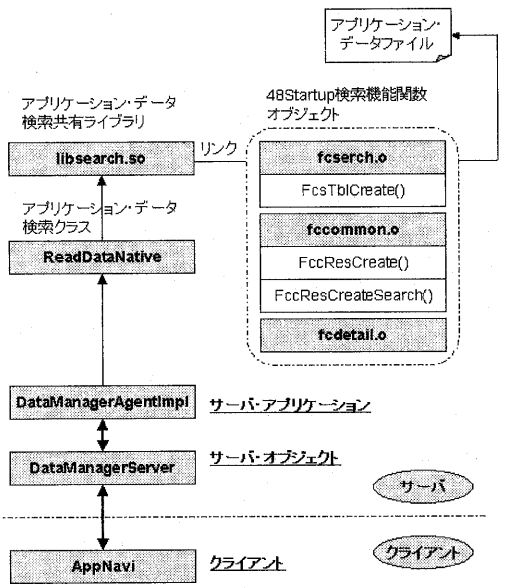

図3 ネイティブ・メソッドによる検索・管理機能実装図

CORBAの実装は、通常ORB製品を用いて、IDL定義ファイルをこの検索機能を持つCプログラムにマッピングさせるが、本システムでは、JavaIDLの機能を用いてJava言語マッピングをし、サーバ側でネイティブ・メソッドとして実行することにより、ORB製品を使用せずに、Java実行環境だけで、JavaとC言語プログラムによる分散処理を可能としている。

3.2. アプリケーション・インストール機能

クライアントで実行されるJavaアプレット(AppNavi)は、JavaIDLを用いてインストール・サーバ上にある検索オブジェクトに対してアプリケーションの情報を要求する。このクライアント・アプリケーションは、カテゴリごとにアプリケーション名の一覧を表示する"カテゴリ別一覧表示画面"とリストから選択したアプリケーションの概要や使用ディスク容量などの詳細情報を表示する"詳細表示画面"を持ち、Javaの実行環境を備えたOSであれば、どこでも動作可能である。

次に、アプリケーションのインストールが選択されると、クライアントは、インストール・サーバが保持しているインストール・パッケージをダウンロードする。次いで、クライアントのOSに応じたインストール・スクリプトが実行される。つまり、Solarisであれば"pkgadd"であり、HP-UXであれば"swinstall"が適切なオプションで実行される。このように、クライアント・アプリケーションは、内部コマンドを実行するため、署名付きJavaオブジェクトとして実装されている。

また、クライアント・アプリケーションは、インストール・サーバにデータを要求する際に、自らの動作環境情報(OSプラットフォーム、ホスト名、メモリ容量など)を引き渡している。ここで、コンピュータの実行環境を調べる手段として、Java標準API、System.getPropertyが提供されているが、実装メモリ容量等の詳細情報をとることはできない。そこで、Runtime.execメソッドを使用して、UNIXコマンドおよびライブラリを使用して詳細情報をとるネイティブ・メソッドを実行し、そこから得られたデータを基にOSプラットフォーム、コンピュータの環境を識別する。このようにして、コンピュータの実行環境取得機能を強化している。

4. アプリケーション・ナビゲータ実行図

図4、5において、Solaris上で実行したアプリケーション・ナビゲータの実行画面を示す。カテゴリ別一覧表示画面では、カテゴリごとにアプリケーションのリストが表示される。アプリケーションを選択し、詳細ボタンをクリックすることで、図5の詳細表示画面が表示される。データ検索では、Javaアプリケーション版もネイティブ・メソッド版も同じ表示画面となる。

詳細表示画面では、アプリケーションの名前、概要、特記事項などの情報を見ることができる。ここで、インストールボタンを選択すると、ユーザはインストール・パッケージをダウンロードしたり、インストール・コマンドを入力したりせずに、GUI操作だけで、インストール作業を行うことが可能である。

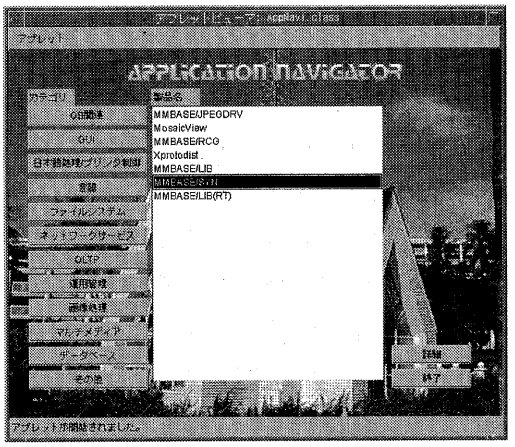

図4 カテゴリ別一覧表示画面

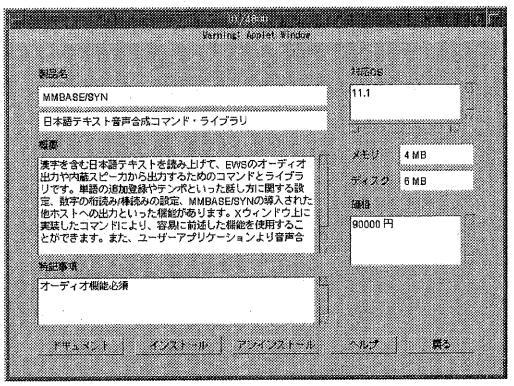

図5 詳細表示画面

5. おわりに

本研究では、アプリケーションのデータを管理するサーバとインストール作業を行うクライアントを JavaIDL を用いて分散システムとして実装し、マルチプラットフォーム対応の GUI アプリケーション・インストーラを開発した。本インストーラの特徴は、以下である。

- 専門的な知識を必要とする UNIX OS において、インストール作業そのものを簡易化
- プラットフォームが異なっても同じ GUI で、インストール作業が可能
- 既存のインストーラ製品の機能を流用し、開発工数を低減、動作信頼性を確保
- マルチプラットフォーム対応インストーラ製品と比較して、OS デフォルトのインストール・スクリプトを利用するため、OS から見たときに通常のインストール作業と何ら変わりがない
- 既存のインストール・パッケージを、そのまま利用できる

ただし、C 言語プログラムをネイティブ・メソッドを用いて再利用しているために、プラットフォームに依存してしまう、意図的なメモリアクセスを許してしまうためセキュリティ・レベルが低下する、ネイティブコード呼び出しによるオーバヘッドが起こるなどの問題があり、今後、さらに設計コンセプトの正当性を検証する必要がある。そのためには、OS デフォルトのインストール方法と本インストーラにおける作業をステップごとに比較し、インストール作業負担の軽減を評価し、さらに既存プログラムの再利用による開発工数の削減、動作信頼性の向上については、Java 言語で実装している検索機能と、再利用した C 言語で実装された検索機能とを比較し、定量的に評価する方向で研究を進める予定である。

参考資料、文献

[1] InstallShield Java Edition:
(http://www.installshield.com/)
InstallAnywhere: (http://www.zerog.com/)
ブロッス: (http://www.10art-ni.co.jp/)
InstallStudio: (http://www.boc.co.jp/)
InstallMaker: (http://www.wisesolutions.com/)
Installer VISE: (http://www.mindvision.com/)

[2] 48Startup: http://www.mid.comp.nec.co.jp/

[3] 手塚、藤城、本林、"パソコン LAN システムにおける遠隔構築支援ツール:Remote Easy Installer"、情報処理学会論文誌、Vol.39、No.4、pp.1026-1038、1998.

[4] 熊谷、原、森、"CORBA によるアプリケーション実装の手引き"、Java WORLD、IDG コミュニケーションズ、pp.158-162、September、1998.

[5] Michael Morrison and Jerry Ablan 著、福井、久野ら訳、"続・Java 言語入門"、プレンティスホール、1998