既存 Java プログラムのバイトコード変換による機能分散

立堀 道昭 † 佐々木 俊幸 †† 千葉 滋 †††,†††† 板野 肯三 †††

† 筑波大学大学院 工学研究科 †† 筑波大学 第三学群情報学類
††† 筑波大学 電子・情報工学系 †††† 科学技術振興事業団 さきがけ研究 21

要 旨

既存の Java バイトコードの機能分散を目的として開発されたシステム、Addistant を提案する。ここで既存のバイトコードとは、単一の Java 仮想機械（JVM）上で動作するように開発された、通常のバイトコードを意味している。本システムでは、既存の JVM を活用するため、バイトコード変換による手法を用いている。Addistant の利用者は、各クラスのインスタンスを、複数の JVM のうちのどこに配置するかを、実行するプログラムとは別に記述する。Addistant は その指定に従いバイトコードを変更して、指定された特定の部分が遠隔の JVM 上で動作し、ローカルの JVM 上で動作する残りの部分とネットワークを越しに通信するようにする。例えば、Java Swing ライブラリを用いた既存プログラムを、遠隔地にある JVM 上で動作させつつ、その GUI オブジェクトを手元にある別の JVM 上で動作させることができる。

1 はじめに

今日、オブジェクト指向の分散ソフトウェアの開発のために、様々なプログラミング支援が存在する。例えば、Java 標準の RMI を利用して、あるオブジェクトをネットワーク越しにアクセスできるようにする、といったことが簡単にできる。Java RMI では、付属する rmic コンパイラを用いることにより、オブジェクトのインタフェースを定義するだけで、ネットワーク越しの通信に関するプログラムは自動生成される。または、Emerald[1] に代表される、分散に特化した言語を利用することもできる。そのような言語は、遠隔ホストにオブジェクトを生成するなどの言語機構を備えている。

しかしながら、これらの分散プログラミング支援は既存のプログラムを改造して部分的に遠隔ホストで実行させるようにする際にはそれほど有用ではない。ここで、既存のプログラムというのは、単一のホストで実行することを意図して開発されたものを指す。プログラマは手動でソースコードを修正してそれらの道具や言語要素の取り決めに従うようにしなければならない。この修正は時間がかかるし、誤りを犯しやすい。また、ソースコードが手に入らない場合には、そのような修正は困難である。

既存 Java プログラムの分散実行への適応化を支援するために、我々は **Addistant** というシステムを開発している。Addistant は次の特徴をもつ。

- Addistant の利用者に、各クラスごとにそのインスタンスを分散環境中のどこに配置するかを指定させる。プログラムは数多くのオブジェクトを生成するため、全てのオブジェクトについてそれぞれどこに配置するかを指定することは現実的でない。Addistant では、この指定を簡素化するために、1 つのクラスのインスタンスは全て同じ配置方針に従う。
- 対象プログラムのバイトコードを変換して、指定されたクラスは遠隔ホストで動作している Java 仮想マシン (JVM) 上で実行されるようにする。Addistant は変換のために、対象プログラムのソースコードを必要としない。変換されたバイトコードは正規の Java バイトコードであり、実行のために特別な JVM を必要としない。
- 変換されたバイトコードは Addistant の実行系により遠隔 JVM に自動的に配布される。

我々は、このようなシステムを Java の **拡張クラスローダ** [4] として実装した。Addistant はクラスをロードする際、バイトコードを変換する。バイトコード変換には Javassist[2] を用いた。

Addistant では、遠隔オブジェクト参照を、従来の分散プログラミング・ツールで使われてきたアイデアを組み合わせて実現した。この実装は、 **プロキシ・マスタ方式** に基づいたもので、Addistant がバイトコード変換により自動的に行う。Addistant 独特の特徴は、プロキシ・マスタ方式の実装について

複数の選択肢を備え、それらを組み合わせている点にある。利用者はクラスごとに異なる実装方式を選ぶことができる。単一の実装方式のみで、既存プログラムを分散実行に適応化することは難しいため、この特徴は重要である。

Addistant の典型的な使い方は、既存の Java プログラムに機能分散を適用し、プログラムの構成部品の一部を、その部品の機能に適した遠隔ホストで実行できるようにすることである。グラフィカル・ユーザ・インタフェース (GUI) をもったアプリケーションプログラムを想定されたい。もしエンドユーザが遠隔ホストの前にいるならば、プログラムの GUI 部分がプログラムの本体から切り離され、その遠隔ホスト上で動作すると便利である。X Window システム [7] を利用するすることで同様な効果を得られるが、Addistant では GUI の反応速度をよりよくできる。これを示すために、我々は、Addistant を用いて、 Swing クラスライブラリ [8] を使った既存プログラムの GUI 部を分散させ、プログラム中の Swing オブジェクトが遠隔ホストで動作するようにした。我々は、X Window システムに比べて、Addistant による分散化により反応速度をかなり改善できたことを確かめた。

2 Addistant

Addistant は、単一ホスト上で実行する意図で開発された既存のプログラムを分散させ、そのプログラムを構成する一部分を遠隔ホストで実行できるようにするための Java 用のプログラミング・ツールである。この分散化はロード時のバイトコード変換器により実現される。

2.1 設計目標

新しく分散プログラムを開発する場合と異なり、既存のプログラムを分散実行させるためには特別なツールの支援が必要である。プログラマはそのプログラムのいくつかのオブジェクトが遠隔ホストに配置されるようにし、また、ネットワーク越しのメソッド呼び出しを特別に扱うように、プログラムを変更しなければならない。手動でこの種の変更を行うのは骨が折れる上、誤りを含めやすいので、この作業はプログラミング・ツールによって自動化されるべきである。

Java を分散用に拡張した言語はこの目的に適していない。第 1 に、対象となるプログラムのソースコードは常に手に入るとは限らない。第 2 に、プログラマは、拡張言語の特殊な構文を使うようにプログラムを変更しなければならない。例えば、Nagaratnam の提案する言語 [5] では、 `remotenew` という特殊な演算子を用いて、遠隔ホストにオブジェクトを生成することができる。この場合プログラマは、ソースコードを入手して、各所に散らばる `new` の使われている全ての部分を調べ、必要な部分だけ `remotenew` を使うようにしなければならない。

著者らの知る限り、既存の Object Request Broker (ORB) もまた、この目的に適していない。プログラムの構成部分の一部を遠隔ホストに分散させるために ある ORB を使うとすると、プログラマは手動でプログラムを分割し、プログラムが分割された構成要素間の相互作用が、その ORB の規約に従うようにプログラムを修正しなければならない。例えば、Java RMI では、全ての遠隔メソッド呼び出しは Java のインタフェース型を通して行われる。クラス Frame の遠隔オブジェクト `f` に対してメソッド `show()` が呼ばれる場合を想定しよう。まずプログラマは、 `show()` を含んだインタフェース DistributedFrame を宣言し、 Frame のクラス宣言を編集して DistributedFrame を `implements` 節に追加する。次に、ソースコード中で使われている Frame を DistributedFrame で置き換える。さらには、遠隔オブジェクト生成や Frame のサブクラスについて注意を払わなければならない。

既存プログラムの機能を分散させるためのツールは、次のような特徴を備えているべきであると我々は考える。

自動化された遠隔参照の実装 遠隔オブジェクト参照に関する実装の詳細をツールの利用者から隠す。すなわち、利用者は、ツールにより指定される特殊な規約に従うようにプログラムの変更をする必要がない。

簡素なオブジェクト配置の方針 各オブジェクトがローカルと遠隔のどちらのホストに配置されるか利用者が簡単に指定できる。オブジェクトの配置は適切な抽象レベルで指定されるべきである。

プログラムの自動配布 遠隔ホストで実行される構成部品を自動的にそのホストに配布できる。

このうち、プログラムの自動配布は、Java のクラスローダの機構のもと、比較的容易に実現できる。以下本稿では、これを除いた最初の 2 つに焦点をあてる。

2.2 遠隔参照

Addistant では、Remote Proxy パターン [6] としても知られるプロキシ・マスタ方式を用いて、遠隔参照を実装する。この方式では、遠隔メソッド呼び出しがローカルメソッド呼び出しと同様、透過的にオブジェクトへのメソッド呼び出しという形で実現される。[1] 遠隔から呼び出すことができるオブジェクトは、遠隔ホストに存在するプロキシオブジェクト（プロキシ）と関連付けられる。区別のために前者をマスタオブジェクト（マスタ）と呼ぶ。プロキシは、マスタと同等のメソッド群を備え、メソッド呼び出しがあるとマスタに委譲する。プロキシは、その遠隔メソッド呼び出しに必要なネットワーク通信の実装を担い、その詳細を呼び出し側から隠蔽する。

Addistant は、JVM のクラスロード時にバイトコードを変換することによりプロキシ・マスタ方式を実装する。変換されたプログラムを動作させるために特別な JVM は必要ない。Addistant では、プロキシ・マスタ方式を実装するために、複数の手法を用いている。これらの手法は全て、対象プログラムのバイトコード変換によって実現される。相違点は、どのようにプロキシのクラス（プロキシクラス）を定義するか、どのようにマスタのクラス（マスタクラス）を変更するか、どのように呼び出し側のコード（遠隔オブジェクトにアクセスするコード）を変更するか、という実装の方法である。

どの手法をとっても、それだけでは全ての種類のマスタに適用することはできない。各手法には、適用するマスタが満たさねばならない、その手法特有の制約がある。我々は、その様な制約を考えずに書かれた、既存のプログラムのためのツールを設計しているため、単一の手法を選んでプログラム全体にその手法の制約を課すことはできない。例えば、ある手法は、マスタクラスの宣言を変更する必要がある。JVM は java.util.Vector のようなシステムクラスの変更を許さないため、もしシステムクラスのインスタンスが遠隔オブジェクトならば、この手法を用いることができない。

この問題を回避するため、 Addistant では、クラスごとにこれらの手法のうちの１つを利用者が選択できる。利用者がある手法を選択するために注意しなければばならない制約は、次の事項である。

[1] これに対し、コード中のメソッド呼び出しごとに対象がローカルか遠隔かを調べ、分岐を設ける方式もあるが、本稿では単純化のため、プロキシ・マスタ方式に絞って議論する。

参照渡し 遠隔メソッド呼び出しのパラメータとして、マスタが参照の形で渡されなければならない場合。複製の形で渡しても問題のない場合には無視してよい。

異種性 マスタに対して、ローカルと遠隔の参照の両方が同一ホスト上に存在しなければばならない場合。あるクラスの全てのインスタンスが一方のホストに存在する場合は、ローカルと遠隔が共存する必要はないので、無視してよい。

非可変バイトコード 遠隔参照の実装に必要なバイトコードが変更不可能な場合。例えば、JVM は java.util.Vector のようなシステムクラスを変更することを禁じている。あるマスタクラスについて対象プログラムのバイトコード中のどの部分がこの非可変なバイトコードにあたるかにより、この制約は次の 3 つに細分化される。

1. マスタクラス自身のクラス宣言（**クラス宣言**）
2. マスタクラス型が現れる他のクラス（**参照者クラス**）
3. マスタクラスのインスタンスを生成している参照者クラス（**生成者クラス**）

現在の Addistant の実装では、「置き換え」、「名前変更」、「サブクラス」、「複製」、と呼ぶ 4 手法を選択肢として提供している。以下では、これらの方式の詳細と適用できない状況を述べていく。先に、まとめたものを表 1 に示しておく。

表 1: 4 手法の適用可能性

適用上の制約	置き換え	名前変更	サブクラス	複製
参照渡し				x
異種性	x	x		
非可変クラス宣言	x		(x)	(x)
非可変参照者クラス		x		
非可変生成者クラス		x	x	

x は、左の制約を満たすことが必要な際にその手法が適用できないことを示す。(x) は、適用できない場合があることを示す。

「置き換え」手法

「置き換え」手法は、異種性の必要がなく、かつ、クラス宣言のバイトコードが可変である場合に適用できる。

クラス Widget にこの手法を適用する場合をを考える。異種性が必要ない場合であるため、1 つの JVM 上では、全ての Widget オブジェクトへの参照は、ローカルか遠隔のどちらか一方である。ゆえに、Addistant は元の Widget クラスをローカル参照に用いる。そして、名前が同じで実装がプロキシになっている別の版の Widget クラスを遠隔参照に用

いる。

「名前変更」手法

「名前変更」手法は、「置き換え」手法と異なり、クラス宣言のバイトコードが非可変である場合にも適用できる。ただし、参照者クラスまたは生成者クラスが非可変である場合には適用できない。「置き換え」手法と同様、「名前変更」も、異種性が必要な場合には適用できない。

この手法では、Addistant は元のクラス Widget に対して、 WidgetProxy のような異なる名前でプロキシクラスを生成する。Addistant はそのプロキシクラスを Widget オブジェクトの遠隔参照に用いる。

1 つの JVM 上では Widget オブジェクトへの参照は全てローカル参照となる。それ以外の JVM 上では、 Widget という名前が現れるバイトコードは編集されて、元の名前は全てプロキシクラスの名前 WidgetProxy で置き換えられる。すなわち、全て遠隔参照となる。

Widget オブジェクトへの参照が遠隔参照であるホストでは、参照者クラスのバイトコードは編集される。例えば、

```
Widget w = new Widget();
```

は、次のように変更される。

```
WidgetProxy w = new WidgetProxy();
```

「サブクラス」手法

「サブクラス」手法では、異種性の制約がある場合でも適用できる。

この手法では、プロキシクラス WidgetProxy は元のクラス Widget のサブクラスである。ローカルと遠隔の両方の参照は Widget 型であり、同一ホスト上で共存できる。ある参照は、ローカルの場合 Widget オブジェクトを指し、遠隔の場合 WidgetProxy オブジェクトを指す。

この手法は、「名前変更」手法と同様、マスタへの参照が遠隔参照であるホストでは、Addistant は生成者クラスのバイトコードを編集する必要がある。さらに、マスタクラスを宣言しているバイトコードを編集しなければばならない場合もある。第 1 に、元クラスが `final` クラスであるか `final` メソッドをもつ場合、 `final` を取り除かなければならない。そうしないと、サブクラスを作れなかったり、サブクラスで上書きできないメソッドによる不都合が生じる。第 2 に、元クラスのコンストラクタがプロキシとしては都合の悪い副作用を生じる場合、Addistant はそのクラスになにもしない別のコンストラクタを追加して、プロキシクラスのコンストラクタでそれを呼べるようにしなければばならない。例えば、マスタクラス Widget のコンストラクタがローカルのグラフィック装置にアクセスすると都合が悪い。

「複製」手法

「複製」手法は、 `int` のようなプリミティブ型で常に用いられる。また、メソッド呼び出しによって状態の変化しない、 java.lang.String などのクラスに用いることができる。複製されたオブジェクト間では、状態の変更がお互い反映されないため、これが問題になる場合には適用できない。

この手法では、ネットワーク越しに、オブジェクト単位で浅い複製が渡される。各フィールドについてはその型の遠隔参照の実装手法に準じて渡される。現在、Java 標準の ObjectInputStream と ObjectOutputStream を拡張して実装しているため、複製できるオブジェクトのクラスは、標準の直列化可能なクラスに準ずる。準じていない場合には、直列化可能にするために、マスタクラスを宣言しているバイトコードを編集する必要がある。

さらに Addistant は、「書き戻し複製」手法と呼ぶ、少し異なる「複製」方式も提供する。この手法が適用されると、遠隔メソッド呼び出しの引数として渡された複製の中身が、その遠隔メソッドを実行した後マスタオブジェクトに書き戻される。例えば、`byte` 配列型にこの手法が適用される場合を考える。次のコード

```
byte[] buf = ... ;
istream.read(buf);
```

において、遠隔オブジェクト `istream` の `read()` を呼ぶと、 `buf` の複製がローカル参照として遠隔ホストに渡される。 `read()` を実行し終わると、その複製の中身が `buf` に書き戻される。したがって、入力ストリームから読み込まれたバイトデータは、最終的に `buf` に格納される。

2.3 オブジェクトの配置

Addistant では利用者がオブジェクト配置の方針を各クラスごとに指定できる。これは、既存プログラム中に現れる各 new （オブジェクト生成式）ごとに、配置方針をユーザが指定することは現実的でないためである。

利用者はあるクラス C の全てのインスタンスがあるホスト H に配置されるように指示できる。この場合、H 以外のホストで実行される “ `new C()` ”（C インスタンスの生成）のような式は、C のインスタンスを遠隔ホスト H 上に生成するように実行される。もし、クラス C について特にホストを指示しない場合、“ `new C()` ” のような式は、それを実行しているホスト H' 上にローカルに生成するように実行される。

利用者による指示は、設定ファイルに記述され、Addistant を起動する際にシステムに読み込まれる。設定ファイルは XML 風に記述される。例えば、

```
<import proxy="rename" from="display">
    java.awt.*
</import>
```

は、`java.awt` パッケージに含まれるクラスの全てのインスタンスが変数 `display` で指定されるホストに配置されるようにすることを指示している。それらのインスタンスへの遠隔参照は「名前変更」手法により実装される。変数 `display` はシステムを起動する際に実際のホスト名に束縛される。もし、`from` 属性が省かれると、クラス C のインスタンスは “ `new C()` ” 式が実行されたホスト上に生成される。

`java.awt.*` は、`java.awt` に含まれる全てのクラスを意味している。サブパッケージのクラスを含めた全てのクラスを指定するときは、`java.awt.-` を用いる。

あるクラスの全てのサブクラスという指定もできる。例えば、

```
<import proxy="rename" from="display">
    subclass@java.awt.Component
</import>
```

は、「名前変更」手法を、Component とそのサブクラス全てに適用することを意味している。指定したクラス自身は除いた全てのサブクラスを指定するには、`subclass` の代わりに `exactsubclass` を用いる。

「置き換え」手法か「名前変更」手法を適用する場合、`from` 属性を必ず指定しなければならない。逆に「複製」手法を適用する場合、`from` 属性を指定できない。

3 分散 Swing アプリケーション

Swing ライブラリ [8] を用いた既存プログラムを、Addistant による分散化によって、GUI オブジェクトがディスプレイやマウスなどの入出力装置のついたホスト（GUI ホスト）で動作し、それ以外が別のホスト（App ホスト）で動作するようにした。Swing は、標準の Java 実行環境に含まれている GUI クラスライブラリである。

同様の効果は、既存の X Window システム [7] でも得られる場合がある。また、 Rawt[3] では、分散用の特製 Swing ライブラリを提供しており、標準のライブラリを置き換えて使うことで、やはり同様の効果を得られる。しかし、 Addistant によるものでは、App ホストで発生した描画命令が直接 GUI ホスト上の GUI オブジェクトによって行われる。例えば、ユーザの操作によりウィンドウを表示する例を考える。Addistant では、「表示する」という命令が App ホストから GUI ホストへネットワーク越しに渡り、ボタンの描画は GUI ホスト上のボタンオブジェクトが直接行う。一方、X Window では、「線を書く」などの低レベルの命令を数多く組み合わせたものがネットワーク越しに渡されることになり、膨大な通信量が発生する。これは、ネットワークの帯域が広い場合は問題にならないかもしれないが、そうでない場合、反応速度の低下を引き起こす。さらに Addistant では、例えばスクロール表示された画像のスクロールを、プログラムによっては通信なしに行うことができる。

3.1 設定ファイル

この分散適用を指示する設定ファイルは以下のように記述した。

```
<policy>
  <import proxy="rename" from="display">
    subclass@java.awt.-
    subclass@javax.swing.-
    subclass@javax.accessibility.*
    subclass@java.util.EventObject          </import>
  <import proxy="rename" from="application">
    exactsubclass@javax.swing.filechooser.*
    exactsubclass@java.io.[InputStream|OutputStream
                          |Reader|Writer] </import>
  <import proxy="subclass">
    subclass@java.util.[Dictionary
        |AbstractCollection|AbstractMap|BitSet]
                                          </import>
  <import proxy="writeBackCopy">
    array@-                               </import>
  <import proxy="replace" from="application">
    user@-                                </import>
  <import proxy="copy">
    -                                     </import>
</policy>
```

GUI オブジェクトのためのクラス、すなわち `java.awt` 、 `javax.swing` 等のパッケージのクラスとそのサブクラスについて、マスタが `display` で示されるホスト側に配置されるようにした。逆に、 java.io.InputStream 等の、それ自身を除いたサブクラスについては、マスタが `application` 側に配置されるようにした。これらのクラスには「サブクラス」手法を適用できないため、「別名」手法を適用している。

java.util.AbstractCollection 等のサブクラス（ java.util.Vector など）については、「サブクラス」手法を適用し、どちらのホストでもマスタとプロキシが混在できるようにした。配列については、一律「書き戻し複製」手法を適用した。ユーザクラスについては、マスタが `application` 側に配置されるようにして、「置き換え」手法を適用した。残りの、 java.util.Locale などのシステムクラスについては、「複製」手法を適用した。

3.2 応答性能の実験

Addistant による分散の効果を確かめるため、簡単な実験を行った。実験に用いたプログラムは、まず大きさ 1200×900 のウインドウを表示する。このウィンドウの中で、ユーザがクリックしていくと、内部ウィンドウの表示と非表示を繰り返す。この内部ウィンドウは、大きさ 1148×778 の JPEG 画像をちょうどいっぱいに表示するもので、javax.swing.JInternalWindow のサブクラスとして定義されている。クリックイベントは、GUI クラスのサブクラスではない別のクラスのインスタンスに送られる。Addistant により分散化されている場合、この通知は GUI オブジェクトのある JVM とは別の JVM 上のオブジェクトに送られることになる。そこでは、内部ウィンドウのメソッドを呼び出して表示と非表示を制御する。

X Window システム、 Rawt 1.3、 Addistant のそれぞれを用いてこのプログラムを動作させ、クリックして表示するまでに要した時間と通信量を測定した。何も表示していない初期状態から表示させる場合と、いったん表示して隠した後にまた表示させる場合について測定した。GUI ホストに 500MHz PentiumIII (Linux 2.2)、App ホストに 440MHz UltraSparcII (Solaris 2.7) を用いた。JVM は両ホストとも Sun の HotSpot Client VM (build 1.3 mixed mode) を用いた。接続には、100Base-TX full-duplex と 10Base-T half-duplex の両方を試した。

測定結果を表 2 に示す。GUI オブジェクトが GUI ホストのメモリ上に画像を保持するため、Rawt と Addistant では、2 回目のクリックに対する応答時間は短くなっている。ネットワークが 10Base-T の場合には、1 回目でさえ、Addistant での応答時間は最短になっている。これは、X Window システムや Rawt ではホスト間でより多くの通信量を必要とするためである。X Window システムでは数メガバイトの通信量を必要とするのに対し、Addistant では、百キロバイトに満たない。大きな通信量は実行性能のボトルネックになりえる。

表 2: クリック後ウィンドウ表示を完了するまでの応答性能

応答時間（秒）（10Base-T(100Base-TX)）

	X Window	Rawt	Addistant
1 回目	5.6(1.6)	3.2(2.6)	2.0(2.0)
2 回目	5.6(1.4)	0.0(0.0)†	0.0(0.0)†

†0.0 は 0.1 秒以下を示す。

通信量（キロバイト）

	X Window	Rawt	Addistant
1 回目	3493.57	116.20	81.88
2 回目	3438.96	10.95	0.06

4 まとめ

本稿では、既存の Java プログラムを分散実行可能なものにするプログラミング・ツール、 Addistant を提案した。Addistant は、この分散化をロード時のバイトコード変換により実現する。Addistant の利用者は、各クラスのインスタンスがどこに配置され、それらのインスタンスの遠隔参照がどのように実装されるかを指定するファイルを与えるだけでよい。利用者は、Addistant により提供される 4 種類の実装方式から選ぶことができる。

Addistant の典型的な利用例として、 Swing ライブラリを用いたプログラムについて、GUI オブジェクトがエンド・ユーザ側にあるホスト（GUI ホスト）で動作し、その他のオブジェクトが別のホストで動作するシステムを示した。同様な機能分散は、従来の X Window システムで実現されてきたことであるが、この機能分散により、 GUI の反応速度を改善することができる。さらに、特製の分散 Swing ライブラリを提供する Rawt に比べても、よりよい実行性能が得られた。Rawt では、GUI ホスト上に配置されるオブジェクトのクラスは固定であるのに対して、Addistant では、Swing クラスを継承して作っ

た、標準でない GUI クラスのインスタンスも GUI ホスト上に配置することができるためである。

参考文献

[1] Andrew Black, Norman Hutchinson, Eric Jul, Henry Levy, and Larry Carter, Distribution and Abstract Types in Emerald, In *IEEE Transactions on Software Engineering, Vol.SE-13, No.1*, IEEE, pp.65–76, 1987.

[2] Shigeru Chiba, Load-time Structural Reflection in Java, In *Proccedings of ECOOP 2000, LNCS 1850*, Springer Verlag, pp.313–336, 2000.

[3] IBM, *Remote Abstract Windowing Toolkit (RAWT)*, Online publishing, URI `http://www.s390.ibm.com/-java/rawt.html`

[4] Sheng Liang and Gilad Bracha, Dynamic Class Loading in the Java Virtual Machine, In *Proceedings of OOPSLA '98, ACM SIGPLAN Notices, Vol.33, No.10*, pp.36–44, 1998.

[5] Nataraj Nagaratnam, Arvind Srinivasan, and Doug Lea, Remote Objects in Java, In *IASTED '96, International Conference on Networks*, 1996.

[6] Hans Rohnert, The Proxy Design Pattern Revisited, In *Pattern Languages of Program Design 2*, Addison-Wesley, pp.105–118, 1995.

[7] Robert Scheifler and Jim Gettys, The X Window System, In *ACM Transactions on Graphics, Vol.5, No.2*, pp.79–109, 1986.

[8] Sun Microsystems, Inc., *Java Foundation Classes*, Online publishing, URI `http://java.sun.com/-products/jfc/index.html`