Addistant：アスペクト指向の分散プログラミング支援ツール

立堀道昭[†1] 千葉滋[†2,†3] 板野肯三[†4]

複数の Java 仮想マシン（JVM）を利用して機能分散を行うソフトウェアの開発を支援するシステム Addistant について，その「関心の分離」（separation of concerns）機能を議論する．Addistant では，プログラマは分散オブジェクトの配置を，分散アスペクトと呼ばれるファイルに，通常の Java プログラムから分離してまとめて記述することができる．また，遠隔参照の複数の実装から適したものをクラスごとに適用することができる．Addistant は分散アスペクト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越しに通信するようにする．たとえば，簡潔な分散アスペクトを記述することにより，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．

Addistant: An Aspect-oriented Distributed-programming Helper

Michiaki Tatsubori,[†1] Shigeru Chiba[†2,†3] and Kozo Itano[†4]

This presentation discusses the function of “separation of concerns” in Addistant, which is a system supporting development of software providing functional distribution on multiple Java virtual machine (JVM). With Addistant, programmers can describe the allocation of distributed objects in a file called distribution aspect separated from a regular Java program. Also, it allows programmers to apply one of several techniques to each class for implementing remote references of the class. According to the specification in that distribution aspect, Addistant automatically transforms the bytecode of the program so that a specified part of the software run on a remote host communicating with the other part. For example, programmers can give Addistant a simple distribution aspect and an existing program written with the Swing library so that Swing objects are executed on a local JVM while the rest of objects are on a remote JVM.

1. はじめに

本稿は，複数の Java 仮想マシンを利用して，機能分散を行うソフトウェアの開発を支援するシステム**Addistant** [12] について述べる．本稿では特に，先行する論文 12) では詳しく触れなかった，Addistant の「関心の分離」（separation of concerns）を行う機能について議論する．

今日，分散ソフトウェア，つまり複数の計算機上で動作するソフトウェアの必要性が高まる一方，その開発にかかるコストが問題となっている．これは，分散プログラムを作成する場合にネットワークなどの分散環境特有の問題に対処しなければならないためである．それらの処理の記述を含んだ分散プログラムは煩雑になり，非分散プログラムの作成に比べて，分散プログラムの作成や維持にかかる人的コストは飛躍的に大きくなりがちである．

分散プログラミングが煩雑であることの要因の 1 つに，プログラムの分散に関係のないロジックの記述の中に分散に関わる処理が拡散して入り交じっていることがあげられる．このようなプログラムは可読性が低く，変更も大変である．分散に関係のないロジックが分かりにくくなるうえ，分散に関わる処理を変更するためにはプログラムのあちこちを修正しなければならなくなるためである．

Addistant では，分散に関わる処理がプログラム全体に拡散して入り交じること避けるため，利用者は，

†1 筑波大学大学院工学研究科
Doctral Program in Engineering, University of Tsukuba
†2 東京工業大学情報理工学研究科数理・計算科学専攻
Department of Mathematical and Computing Sciences, Tokyo Institute of Technology
†3 科学技術振興事業団さきがけ研究 21
PRESTO, Japan Science and Technology Corporation
†4 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

分散に関係のないロジックを記述した非分散プログラムとは別に，分散に関わる記述をまとめて記述する．このまとめて別に書かれた分散に関わる記述を**分散アスペクト**と呼ぶ．Addistant の処理系は，通常の Java 言語で書かれた非分散プログラムを分散アスペクトに基づいて変換して，分散して実行されるプログラムを生成する．

Addistant の特徴は次のようなものである．

- Addistant の利用者は，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する．プログラムは数多くのオブジェクトを生成するため，すべ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すべて同じ配置方針に従う．
- Addistant の処理系は，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．バイトコードを直接変換するため，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．

バイトコード変換器は Java クラス・ローディングの機構[5] を利用し，**拡張クラスローダ**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 [1] を用いた．

本研究では Addistant における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．

典型的な Addistant の使い方は，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）を持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．我々は，Addistant を用いて，Swing クラスライブラリ[11] 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．そして実験により，Addistant と表面的に同様な遠隔表示を得られる X Window システム[9] に比べて，GUI の応答速度をかなり改善できたことを確かめた．

2. 機能分散プログラムの開発

本研究では，分散ソフトウェアの中でも，機能分散を行う種類のものに注目した．本章では，まず，機能分散について説明する．次に，既存の開発支援を用いて機能分散を行う分散プログラムを作成する際の問題点を述べる．

2.1 機能分散

分散ソフトウェアは，複数の計算機上にまたがって動作するソフトウェアのことであるが，その用途は主に 2 つに分けることができる．1 つ目の用途は，複数の計算機で並列に計算させることによって計算時間を短縮することである．2 つ目は，ソフトウェアの各モジュールに適した計算機を組み合わせて利用することである．後者の用途は機能分散と呼ばれ，本研究は，この機能分散を行う分散ソフトウェア開発にかかるコストを軽減することを目的としている．

機能分散ソフトウェアは，計算機ごとに異なるリソース，たとえば，計算力，画面表示，記憶領域，ネットワーク接続などを組み合わせて利用することができる．たとえば，遠隔地にあるアプリケーション・サーバ上のソフトウェアを動作させるときに，手元にあるマシンのディスプレイやマウスなどのユーザ・インタフェースを利用して操作できると便利である．このような遠隔表示のためには，ソフトウェアのグラフィカル・ユーザ・インタフェース（GUI）を司る部品を手元にある計算機で動作させ，それ以外の部品をアプリケーション・サーバで動作させればよい．

並列計算と異なり，機能分散ソフトウェアではその目的上，プログラマが各ソフトウェア・モジュールを適切な計算機に割り当てることが必要となる．

2.2 既存の開発支援

オブジェクト指向の分散プログラミングでは，Remote Proxy パターン[8] としても知られるプロキシ・マスタ方式を用いて，遠隔オブジェクトへの通信を実現するのが一般的である．この方式では，遠隔オブジェクトへのメソッド呼び出しがローカル・オブジェクトへの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．遠隔か

ら呼び出されるオブジェクトは，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ネットワークを介してマスタの対応するメソッドを呼び出す．

プログラムを複数の JVM 上で動作させるための典型的な分散開発支援は 2 つに分けられる．1 つ目は，Java 言語を分散向けに拡張した分散言語処理系である．2 つ目は，Java RMI[10] のように，通信を扱う実行時ライブラリとプロキシなどのコードを自動生成するツールによる支援で，Object Request Broker（ORB）と呼ばれる．

機能分散を実現するために Java を拡張した分散言語を利用する場合，プログラマは，拡張言語の特殊な構文を使ってプログラムを記述する．たとえば，Nagaratnam の提案する言語[7] 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，プログラム中の各所に散らばる`new` にあたる部分のうち，適切なものについてのみ`remotenew` を使うようにしなければならない．

Java RMI のような ORB を使って機能分散を実現する際にもやはり，散在する`new` にあたる部分に注意を払わなければならない問題が生じる．さらには，プログラマは手動でプログラムの構成要素を分割し，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記述しなければならない．たとえば，Java RMI では，すべ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．

3. Addistant

Addistant を利用するプログラマは，Java で書く非分散プログラムとは別に分散に関する事項をまとめた分散アスペクトを記述する．このように，プログラム中に拡散してしまう事項をまとめて別に記述できるようにするプログラミングのパラダイムは，**アスペクト指向プログラミング**[4] と呼ばれ，近年さかんに研究されてきている．本研究もその中の 1 つに位置づけられる．

3.1 設計目標

プログラム中に拡散する分散に関する事項をまとめて記述できるようにするためには，特別なツールの支援が必要である．通常，機能分散プログラムを記述するには，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なコードを記述しなければならない．手動でこの作業を行うのは骨が折れるうえ，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．

この種の支援を行うプログラミング・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ためのプログラムの変更をする必要がない．

簡便なオブジェクト配置 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容易に指定できる．オブジェクトの配置は適切な抽象レベルで 1 つのファイルにまとめて記述できるべきである．配置方針を修正する場合にも，利用者は，プログラム全体を修正する必要がない．

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．

3.2 遠隔参照

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．これらの手法はすべて，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，それだけではすべ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なければならない，その手法特有の制約がある．したがって，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．たとえば，ある手法は，マスタクラスの宣言を変更する必要がある．しかし，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はできない．

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ならない制約は，次の事項である．

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．あるクラスのすべてのインスタンスが一方のホストにしか存在しない場合は，ローカルと遠隔の参照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．たとえば，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それぞれを適用する状況を述べていく．先に，4 手法の制約別の適用可能性をまとめたものを**表 1** に示しておく．

これより後，あるクラスWidget を用いて説明を行う．Widget のクラス宣言とそれを利用しているコードが**図 1** のようであるとする．

「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．したがって，ユーザ定義のクラスに用いられる．

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，すべての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（**図 2** 参照）．

表 1 4 手法の適用可能性
Table 1 Applicability of the four approaches.

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．

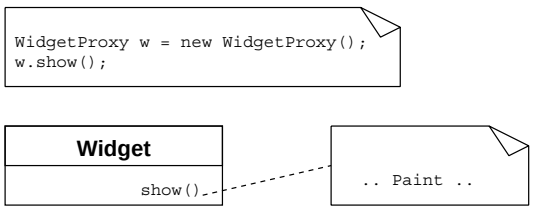

図 1 Widget クラスとその呼び出し側のコードの元の姿
Fig. 1 The original code of Widget and its caller-side.

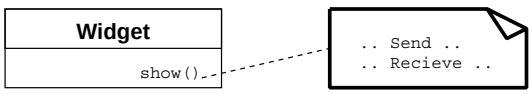

図 2 遠隔Widget オブジェクトを「置き換え」手法により実装したコード．図 1 のWidget クラスのメソッド本体のみがプロキシクラスの実装に置き換わっている．
Fig. 2 The code implementing remote Widget objects by the “replace” technique. Only the method bodies of the class Widget in Fig. 1 are replaced by the one of proxy class.

「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」手法も，異種性が必要な場合には適用できない．java.awt.Window などのシステムクラスに用いられる．

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（**図 3** 参照）．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．

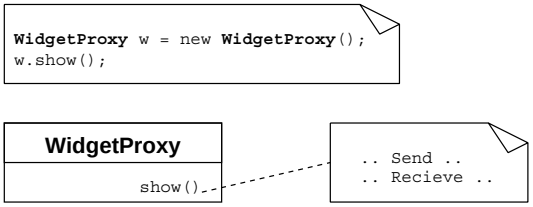

図 3 遠隔Widget オブジェクトを「名前変更」手法により実装したコード．Widget クラスを利用しているコード（図 1）中に現れるWidget というシンボルがプロキシクラスであるWidgetProxy の名前に変更されている．

Fig. 3 The code implementing remote Widget objects by the "rename" technique. The symbols which appear in the caller-side code of the class Widget (in Fig. 1) are renamed to be the name of the proxy class WidgetProxy.

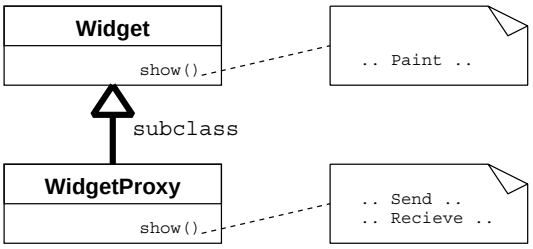

図 4 遠隔Widget オブジェクトを「サブクラス」手法により実装するコード．図 1 のWidget クラスのサブクラスとしてプロキシクラスであるWidgetProxy が宣言されている．

Fig. 4 The code implementing remote Widget objects by the "subclass" technique. The proxy class is declared as a subclass of the class Widget.

1 つの JVM 上ではWidget オブジェクトへの参照はすべ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，元の名前はすべてプロキシクラスの名前WidgetProxy で置き換えられる．すなわち，すべて遠隔参照となる．

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．たとえば，

```
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（**図 4** 参照）．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．java.util.Vector などのシステムクラスや，複数のホスト上にマスタの存在するユーザ定義のクラスに用いられる．

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` メソッドを持つ場合，`final` を取り除かなければならない．取り除かなければ，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が生じたりする．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ならない．たとえば，マスタクラスWidget のコンストラクタが，存在しないローカルのグラフィック装置にアクセスなどの問題が生じる．

「複製」手法

「複製」手法は，`int` のようなプリミティブ型でつねに用いられる．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に反映されないため，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，これらのストリームクラスを用いて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」手法も提供する．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．たとえば，`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．

3.3 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 のすべ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，我々が分散アスペクトと呼ぶ独立した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．分散アスペクトはXML 風に記述される．たとえば，

```
<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すべ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.*という指定は，java.awt に含まれるすべ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すべてのクラスを指定するときは，java.awt.-を用いる．

あるクラスのすべてのサブクラスという指定もできる．たとえば，

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

は,「名前変更」手法を，Component とそのサブクラスすべてに適用することを意味している．指定したクラス自身を除いたすべ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は，from 属性を指定できない．

4. 分散 Swing アプリケーション

Swing ライブラリ[11] を用いた既存プログラムに分散アスペクトを加え，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．

遠隔表示は，既存の X Window システム[9] でも行うことができる．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，遠隔表示を実現できる．しかし，Addistant による遠隔表示では，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．たとえば，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，帯域が狭い場合，反応速度の低下を引き起こす．さらに Addistant では，たと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．

4.1 分散アスペクト

Swing を用いた GUI を分散させる分散アスペクトは**図 5** のように記述した．

GUI オブジェクトのためのクラス，すなわちjava.awt，javax.swing など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 などの，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,「名前変更」手法を適用している．

java.util.AbstractCollection などのサブクラス（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

```
<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>
```

図 5 GUI 部品が遠隔のホストで動作するように指示した分散アスペクト
Fig. 5 A distribution aspect specifying that GUI components run on a remote host.

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．

4.2 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，ユーザがマウスをクリックするたびに，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．その JVM 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，クリックしてウィンドウ表示されるまでに要した時間と通信量を測定した．何も表示され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 500 MHz PentiumIII（Linux 2.2），App ホストに 440 MHz 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．ネットワーク接続には，100 Base-TX full-duplex と 10 Base-T half-duplex の両方を試した．

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
Table 2 The response time to a mouse click.

応答時間（秒）（10 Base-T(100 Base-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 秒は 0.1 秒以下を表す）

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

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 10 Base-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うる．

5. 関連研究

5.1 遠隔表示

X Window システム[9] を利用すれば，非分散 Java プログラムの GUI を遠隔ホストのディスプレイに表示することが可能である．しかしながら，4 章で述べたように，X Window システムは，Addistant を用いて作成された分散プログラムよりもしばしば効率が悪い．

Rawt[3] は Java の標準 GUI ライブラリ互換である．標準ライブラリを Rawt のもので置き換えれば，非分散 Java プログラムで遠隔表示を得ることができる．Addistant は標準のライブラリから半自動的に Rawt のようなライブラリを生成するためのツールと見なすこともできる．ただし，Addistant によって変換される部分はユーザのコードも含むために，結果として Rawt を利用するよりも効率の良いソフトウェアが得られやすい．

5.2 デザイン・パターン

オブジェクトの生成を AbstractFactory と呼ばれるクラスを用いて隠蔽する設計は，Abstract Factory パターン[2] として知られている．このデザイン・パター

ンを利用したプログラムでは，AbstractFactory クラスを継承した具象クラスの実装を変更することにより，実際に生成されるオブジェクトを，プロキシとマスタのどちらにするかを制御できる．

しかしながら，この設計で対応できるクラスの種類は限られる．Abstract Factory パターンでは，プロキシ・クラスとマスタ・クラスの両方が同一のインタフェースを`implements`しているか，またはどちらかがもう一方のサブクラスにならなければならない．したがって，Addistant の「サブクラス」手法と同様の制約が生じるために，java.awt.Window などのクラスに適用することができない．

5.3 分散アスペクト

D[6] もまた，アスペクト指向の分散プログラミングを支援する処理系である．D では分散に関するアスペクトとして，並列に動作するスレッド間の協調動作を扱う coordination アスペクトと，遠隔手続き呼び出しを司る Interface Definition Language（IDL）を扱う remote interface アスペクトを記述できる．これに対し，Addistant の扱うアスペクトはオブジェクトの分散配置と遠隔参照の実装である．我々は，D と Addistant の支援するアスペクトは相補的であり，組み合わせることによってより良いシステムを得られると考えている．

6. ま と め

本稿では，複数の Java 仮想マシンを利用して，機能分散を行うソフトウェアの開発を支援するシステム Addistant を提案した．Addistant を利用するプログラマは，分散環境におけるオブジェクトの配置を，分散に関係のない Java プログラムから分離してまとめて記述することができる．従来，配置を変更する際にプログラム全体にわたって修正を行わなければならなかった作業は，我々が分散アスペクトと呼ぶ独立したファイルの記述の修正のみですむようになった．Addistant は，Java の拡張クラスローダとして実装されており，処理系は分散アスペクトに基づいてバイトコードを変換にする．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．遠隔参照の実装は，Addistant により提供される 4 種類の実装方式からクラスごとに選ぶことができる．

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．同様の機能分散は，従来の X Window システムで実現されてきたことであるが，Addistant による機能分散により，GUI の反応速度を改善することができた．

謝辞 本研究は部分的に文部科学省・科学研究費の補助を受けている．本稿を作成するにあたり，東京大学の光来健一氏には多大な助言をいただいた．匿名の査読者の方々には有益なコメントをいただいた．

参 考 文 献

1) Chiba, S.: Load-time Structural Reflection in Java, *ECOOP 2000 — Object Oriented Programming*, LNCS 1850, Springer, pp.313–336 (2000).
2) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns — Elements of Reusable Object-Oriented Software*, Addison-Wesley (1994).
3) IBM: Remote Abstract Windowing Toolkit (RAWT) (2000). http://www.s390.ibm.com/java/rawt.html
4) Kiczales, G., Lamping, J., Mendhekar, A., Cristina V., Lopes, J.-M.L. and Irwin, J.: Aspect-Oriented Programming, *ECOOP '97 — Object Oriented Programming*, LNCS 1241, pp.220–242, Springer (1997).
5) Liang, S. and Bracha, G.: Dynamic Class Loading in the Java Virtual Machine, *Proc. ACM Conf. on Object-Oriented Programming Systems, Languages, and Applications*, ACM SIGPLAN Notices, Vol.33, No.10, pp.36–44 (1998).
6) Lopes, C.V. and Kiczales, G.: D: A Language Framework for Distributed Programming, Technical Report SPL97-010, Xerox Palo Alto Research Center, Palo Alto, CA, USA (1997).
7) Nagaratnam, N., Srinivasan, A. and Lea, D.: Remote Objects in Java, *Proc. IASTED '96, International Conference on Networks* (1996).
8) Rohnert, H.: The Proxy Design Pattern Revisited, *Pattern Languages of Program Design 2*, chapter 7, pp.105–118, Addison-Wesley (1995).
9) Scheifler, R. and Gettys, J.: The X Window System, *ACM Trans. Graphics*, Vol.5, No.2, pp.79–109 (1986).
10) Sun Microsystems: The Java Remote Method Invocation Specification (1997). http://java.sun.com/products/jdk/rmi/
11) Sun Microsystems: Java Foundation Classes (1998). http://java.sun.com/products/jfc/

12) Tatsubori, M., Sasaki, T., Chiba, S. and Itano, K.: A Bytecode Translator for Distributed Execution of “Legacy” Java Software, *ECOOP 2001 — Object Oriented Programming*, LNCS 2072, pp.236–255, Springer (2001).

(平成 13 年 7 月 11 日受付)
(平成 13 年 12 月 28 日採録)

立堀 道昭（学生会員）

1974 年生．1997 年筑波大学第三学群情報学類卒業．1997 年より同大学院博士課程工学研究科．1999 年同大学院工学修士号取得．言語処理系，プログラミング，システムソフトウェアに関する研究に従事．日本ソフトウェア科学会，ACM，USENIX 各学生会員．

千葉 滋（正会員）

1968 年生．1991 年東京大学理学部情報科学科卒業．1993 年同大学院理学系研究科情報科学専攻修士課程修了．1996 年同専攻博士（理学）取得．2000 年より東京工業大学情報理工学研究科数理・計算科学専攻講師．言語処理系およびオペレーティングシステム等システムソフトウェアの研究に従事．日本ソフトウェア科学会，ACM 各会員．

板野 肯三（正会員）

1948 年生．1977 年東京大学大学院理学系研究科物理学専門課程博士課程単位取得後退学．1993 年より筑波大学電子・情報工学系教授．計算機のアーキテクチャ，分散処理システム，プログラミングシステム等に関する研究に従事．理学博士．日本ソフトウェア科学会，電子情報通信学会，ACM，IEEE 各会員．