プログラム分散化のためのアスペクト指向言語

西澤 無我* 立堀 道昭† 千葉 滋‡

要 旨

本稿では、プログラム分散化のためのアスペクト指向言語（AOP）である Addistant を拡張した、Addistant 2 を提案する。Addistant はプログラム変換器であり、アスペクト指向言語で指定した分散に関する記述に従って、単一の Java 仮想機械（JVM）上で実行することを目的とした既存プログラムを、自動的に分散プログラムに変える。この分散化はロード時に、バイトコードレベルで行われる。Addistant 2 では、アスペクト指向言語の *join point* の種類を増やし、大幅に分散アスペクトの記述力を強化した。

1 はじめに

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になっている。分散プログラムでは、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、それらの処理の記述を含んだプログラムは煩雑になる。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として、プログラムの分散に無関係な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（crosscutting concerns）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。分散に無関係な記述が分かりにくくなる上、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。

我々は分散プログラミングをアスペクト指向技術（AOP）[2] により支援する Addistant[1] を開発している。アスペクト指向とは、オブジェクト指向との相補性を意識した概念である。オブジェクト指向とは、機能性という点に着目して全体をオブジェクトと呼ばれるモジュールに分割していく技術である。しかし、個々のオブジェクトに分割できない処理もある。このような処理を、オブジェクトとは異なる形でモジュール化する技術を**アスペクト指向**といい、そのモジュールを**アスペクト**という。

Addistant は、分散を意識しないで書かれた既存の Java プログラムを、利用者の指示にしたがって自動的に分散処理をおこなうプログラムに変換するツールである。Addistant は次のような特徴をもつ。

- 利用者は、プログラムをどのように分散化させるかの指示を、変換されるプログラムから独立した異なるファイルに記述する。この記述のことを、**分散アスペクト**と呼ぶ。これにより、変換されるプログラムには一切手を加えずに、分散化をおこなうことができる。

- Addistant は、プログラムの変換をバイトコード・レベルでロード時におこなう。変換されるプログラムのソースコードは必要ない。変換後の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。バイトコード変換には Javassist[3] を用いる。

  対象プログラムのバイトコードを変換して、分散アスペクトによって指定されたクラスが、遠隔ホストで動作している Java 仮想機械（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 Addistant の実行系により各ホストに自動的に配布される。

* 東京工業大学理学部情報科学科 Tokyo Institute of Technology, Faculty of Science
† 筑波大学大学院工学研究科 Doctral Program in Engineering, University of Tsukuba
‡ 東京工業大学大学院情報理工学研究科 Graduate School of Information Science and Engineering Tokyo Institute of Technology

Addistant の開発にあたっての研究課題は、主にオブジェクトの遠隔参照の実現と、分散アスペクトの記述言語である。オブジェクトの遠隔参照は、もしプログラム全体が変換可能であれば、**プロキシ・マスタ方式** [6][7] により容易に実現できる。しかし Java 言語には、変換を禁じたシステム・クラスが存在するので、単純なプロキシ・マスタ方式では対応しきれない。この問題については、Addistant の最初のプロトタイプ (区別のため Addistant 1 と呼ぶ) で対処した [1]。

本稿は、2 つめの研究課題に対処するため、現在我々が開発している Addistant 2 について述べる。分散アスペクトの記述を、例えば差分パッチのように単純に記述すると、記述の可読性や保守性が、実用にならないほど低下してしまう。我々はアスペクトの記述言語の設計にあたり、join point を注意深く選ぶことで、アスペクト記述の抽象度を一定以上に保ちつつ、記述力が高まるようにした。

以下、第 2 章では Addistant プロジェクトの概要と研究課題、および我々が既に開発済みの Addistant 1 での研究成果について述べる。第 3 章では、我々が現在開発中の Addistant 2 での分散アスペクト記述の詳細を、第 4 章では、Addistant 2 を利用したソフトウェアの機能分散例、第 5 章では、関連研究を、そして第 6 章でまとめを述べる。

2 分散プログラミング用アスペクト指向言語

分散プログラミングの難しさはよく知られており、これまでにもさまざまな分散処理技術が提案され、またそれを実現する支援ツールが開発されてきた。本章では、我々が開発している支援ツール Addistant を紹介し、開発における研究課題を述べる。

2.1 Addistant プロジェクト

我々は、既存の非分散 Java プログラムを、分散処理をおこなうプログラムに、ユーザの指示にしたがって自動的に変換するツール Addistant を開発している。ここで既存の非分散 Java プログラムとは、単一の Java 仮想機械（JVM）上で実行されることを目的として開発された Java プログラムのことである。Addistant のユーザが、各オブジェクトをどのホストに分散配置するか等を記述すると、Addistant はその記述にしたがって、コンパイル済みの既存 Java プログラムを、バイトコードレベルで変換し、分散プログラムにする。

Addistant を実現する上で、基幹となる技術は以下の 2 つであると我々は考えている。

- **オブジェクトの遠隔参照**

  異なるホスト上に存在する遠隔オブジェクトの参照を可能にしたプログラミング言語は数多く開発されてきている [8]。オブジェクトの遠隔参照は、さまざまな方法で実装できるが、典型的にはプロキシ・オブジェクト を使う**プロキシ・マスタ方式**を使って実装することができる。プロキシ・オブジェクトを使えば、遠隔参照だけでなく、オブジェクトの動的な migration 等も自然に実装できることが知られている。

- **分散アスペクトの記述**

  Addistant では、ユーザがオブジェクトの配置や migration の指示を明示的に行う。この指示は、通常の Java プログラムからは独立したファイルに記述される。我々はこの記述のことを、**分散アスペクト記述**と呼ぶ。静的･動的な負荷分散によって高速な並列処理をねらう場合には、分散アスペクト記述をユーザに書かせるのではなく、システムが自動的に判断する方法が可能であろう [9]。しかしながら、我々がねらっている分散処理は、機能分散処理と呼ばれるものである。機能分散処理では、例えば、GUI にかかわるオブジェクトを、GUI のハードウェア資源、すなわちディスプレイやマウス、があるホスト上に配置することで、より高速な応答性を実現することが目的である。与えられたオブジェクトが利用しているハードウェア資源が何で、それをもっとも効率的に使えるのはどのホストであるか、等をシステムが自動的に判断するのは困難であると考えられる。そこで我々は、この点については、ユーザが分散アスペクト記述を書き、明示的にシステムに指示する方法を選んだ。

2.2 研究の課題

オブジェクトの遠隔参照の実現方法については、数多くの研究が存在する。Java 言語のようにポー

タビリティが重要で、JVM の改変が現実的でない場合の実現方法も存在する。例えばプロキシ・マスタ方式に基づいて適当なプログラム変換をおこなえば、既存の非分散プログラムを、標準の JVM 上で動く分散プログラムに変換することが可能である。

しかしながら、システムの根幹にかかわる一部のクラス (システム・クラスと呼ぶ) を含むプログラムについては、単純なプロキシ・マスタ方式に基づいたプログラム変換によって分散化することができない。Java 言語の実行系の仕様では、セキュリティ上の理由により、システム・クラスについてはプログラム変換を許していない。プロキシ・マスタ方式は、プログラム変換によってプログラム中の変数や引数の型を自由に別な型に変更できることを前提にしているので、変換が禁止されているシステム・クラスを含んだプログラムでは、うまく働かない。オブジェクトの遠隔参照の実現方法が盛んに研究されていた 1980 年代から 90 年代にかけて主流であった C++ 言語では、特定のクラスのプログラム変換を制限するような機構が存在しなかったため、このようなプロキシ・マスタ方式の限界を回避する方法は研究されてこなかった。

一方の分散アスペクトの記述については、現在、アスペクト指向プログラミングとして世界中で研究されている課題のひとつであり、さらなる研究が必要である。従来の分散プログラミング言語では、分散アスペクト記述に含まれるような指示が、プログラム中に直接埋め込まれていた。例えば、オブジェクトを生成する際、どのホスト上で生成するかを指示するのに、`new` 演算子の機能を拡張して、オブジェクトを生成するホスト名を引数としてとれるようにしていた。しかしながら、このような方法では、分散処理にかかわる指示がプログラム中に散らばってしまい、プログラムの可読性や保守性が低下してしまう。

我々は、分散処理にかかわる指示を直接プログラム中に埋め込むのではなく、独立した別のファイルにまとめて記述することで、この問題を解決しようとしている (SOC: Separation of Concerns)。我々はこのファイルのことを分散アスペクト記述と呼ぶ。分散処理にかかわる指示とは、オブジェクトを生成するホスト名や、オブジェクトの遠隔参照の実現方法、migration のアルゴリズム、等である。また分散化によって、元のアプリケーションのロジック (あるいは挙動) が変える必要がある場合、それをどのように変えるかの指示も、分散処理にかかわる指示に含まれる。従来の研究では、分散にかかわる指示として、ホスト間での引数の渡し方など、アプリケーションのロジックから完全に独立している指示だけを扱っていた [5]。しかし、このような指示だけでは、Addistant の目的である、既存の非分散 Java プログラムの分散処理プログラムへの変換には不十分であることがある。我々は、アプリケーションのロジックから完全に独立している指示だけを分散アスペクト記述に含めるのではなく、変換に必要な指示は全て含められるように開発を進めている。Addistant の分散アスペクト記述は、既存の非分散プログラムを分散化するのに必要な指示を集めたもの、と考えることもできる。

分散処理にかかわる指示を、分散アスペクト記述として分離して記述するのは必ずしも容易ではない。それぞれの指示は、対応するプログラム中の特定のコードと密に結びついていることが多いので、この結びつきをうまく表現しなければならない。このような結びつきのことをアスペクト指向プログラミングでは *join point* と呼ぶ。

例えば、特定の `new` 演算子によるオブジェクトの生成の際に、どのホスト上にオブジェクトを生成するかを指示する場合を考える。このような指示を簡単に記述する方法としては、`diff` プログラムのように、元のプログラムに対する差分パッチとして指示を記述する方法が考えられる。しかしこのような記述方法は、柔軟性がなく、プログラム全体の可読性や保守性をかえって悪化させる。もし、対象とする `new` 演算子を行番号で指定すると、元のプログラムがわずかでも修正された場合に、結びつきが壊れてしまう。行番号だけで `new` 演算子を指定するのではなく、その演算子の前後のコードも合わせて指定し、行番号が多少ずれてしまっても、前後のコードの文字列のパターンマッチによって、目的とする演算子を見つける方法 (文脈付差分パッチ) も知られている。しかしこの方法も、前後のコードが変更されてしまうと無力である。このように、行番号や表面的な文字列のパターンマッチによって、分散アスペクト記述とプログラムを結びつける方法は、十分な柔軟性がない。

2.3 Addistant 1

我々は Addistant の既に最初のプロトタイプを完成させ、さまざまな実験をおこなった [1]。以下、区別のため、このプロトタイプのことを Addistant 1 と呼ぶ。

Addistant 1 の主要な貢献は、書き換えが禁止されているシステム・クラスのオブジェクトも遠隔参照可能にする技術を開発したことである。システム・クラスとは、JVM とともに配付される標準の基本クラスのことであり、java.awt.Window などがそれにあたる。既存プログラムではシステム・クラスが多用されており、Addistant を現実的な分散化システムとするためにはシステム・クラスへの対応は欠かせなかった。実際、この技術により Swing コンポーネントを Addistant で分散化することが可能になった。

Addistant 1 では遠隔参照の実装に、「置き換え」「名前付け替え」「サブクラス」「複製」と呼ぶ 4 つの実装法を用意している。それぞれの実装法は、コード中の書き換えを必要とする箇所が異なるため、ユーザがクラスによって適切な実装法を選択し、システム・クラスを書き換えないですむようにできる。例えば、java.awt.Window を遠隔参照するためには、「名前付け替え」法を適用する。この実装法はクラスの利用側のみの書き換えで済むため、書き換え不可能な java.awt.Window クラスに用いることができる。

一方、Addistant 1 は分散アスペクト記述については十分な機能を用意していない。あえて複雑な指示は可能にせず、平易な記述で実用的な分散化を指示できるようにすることを主眼においた。これにより、分散処理にかかわる指示を別のファイルに分離して記述しても、記述の複雑度が実用の範囲内におさまることを示した。

Addistant 1 での分散アスペクト記述は、XML を使って記述する。Addistant 1 のユーザは、クラス単位でオブジェクトの配置を指定することができる。以下に分散アスペクトの記述例を示す。

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

この例の分散アスペクト記述により、Component クラスとそのサブクラス全てのオブジェクトは、変数 app で指定するホストに配置される。また、オブジェクトの遠隔参照を「名前付け換え」法を用いて実装する。この例では、Component オブジェクトの配置を指定しているが、ワイルドカード * を用いてパッケージに含まれる全てのクラスのオブジェクトの配置を指定することもできる。

3 Addistant 2 - 記述力強化版

我々はより複雑な分散アスペクト記述を可能にするように Addistant 1 を改良した Addistant 2 を現在開発中である。Addistant 1 は、*join point* の種類を意図的に少なくしていたため、分散アスペクトに記述できる事柄の範囲が狭かった。Addistant 2 ではこれを改善し、 *join point* の種類を増やし、分散アスペクトのより詳細な記述を可能にした。ただし記述力を強化することによって、アスペクト記述が非実用的なほど複雑にならないように注意を払った。

Addistant 2 で、最も大切な *join point* の拡張は、より詳細なオブジェクト配置の指定を可能にした点である。2.3 節で述べたように、Addistant 1 は、オブジェクトの配置をクラス単位で指定することができた。現在開発中の Addistant 2 では、既存コード中の `new` 演算子のあるクラス、メソッドの種類といった、オブジェクトの生成文脈に応じた、オブジェクトの配置を指定することができる。

それ以外の追加された *join point* の種類としては

- システムの実行環境の属性
- Object Request Broker の実装
- ネットワーク・ストリームの実装

があげられる。Addistant 2 の分散アスペクト記述では、これらの *join point* を XML[10] を使って細かく指定できる。以下では、上記の Addistant 2 の *join point* を説明し、分散アスペクトの記述例を詳しく解説していく。

3.1 オブジェクトの配置

Addistant 2 では、どのホストにオブジェクトを配置し動作させるか、という指示を既存コード中の `new` 演算子のあるクラス、メソッドの種類といった、

オブジェクトの生成文脈に応じて、オブジェクト毎に細かく指定することができる。

Addistant 2 の、オブジェクト配置の記述の例を以下にあげる。

```
<import proxy="subclass"
        in="FrameClient.createViewerFrame()"
        from="rmt">
    <subclass name="javax.swing.JFrame"/>
</import>
```

この分散アスペクトの記述により、FrameClient クラス内の createViewerFrame() メソッド中で生成される JFrame クラスとそのサブクラスのオブジェクトは変数 rmt で指定するホストに配置される。また、遠隔オブジェクト参照は、「サブクラス」法を用いて実装される。Addistant 2 で新たに導入した in 属性は、その値にクラス名だけでなく、メソッド名まで指定することができ、in 属性値で指定された範囲が、オブジェクト配置の指定の有効範囲となる。

3.2 システムを実行する環境の属性

Addistant 2 ではシステムの実行環境も細かく指定できる。システムの実行環境とは、ネットワークでつながれたホスト名や、ネットワーク通信でソケットを張るためのポート番号、また手元にないクラスファイルをネットワーク経由でクラスローダが探してくる際の、検索場所（ディレクトリ）のことである。

以下では、具体的に分散アスペクト中で実行環境を記述する例を示す。

```
<start name="app">
    <host name="picard"
          rmiPort="14004"
          bytecodePort="14005"
          searchDir=apprun"/>
</start>
<remote name="rmt">
    <host name="taro"
          rmiPort="14006"
          bytecodePort="14007"
          searchDir="rmtrun"/>
</remote>
```

この記述は、ノード値 start のノードと、ノード値 remote のノードの 2 つに分けることができる。前者は変数 app で指定されるホストの情報を記述するノードであり、後者は変数 rmt で指定されるホストの情報を記述するノードである。変数 app と rmt は Addistant 2 独自の変数である。さらに、ホストの詳細の情報はノード host が表す。ノード host の中身は、ホスト名（name 属性）、ポート番号（rmiPort,bytecodePort 属性)、ディレクトリ（searchDir 属性）の 4 つの属性からなり、これらを分散アスペクト記述により指定することができる。

3.3 Object Request Broker の実装

Addistant 2 ではユーザ定義の Object Request Broker（ORB）を利用することもできる。ORB とは、遠隔オブジェクトを生成したり遠隔メソッドを呼び出すなど、遠隔オブジェクトに対するネットワーク越しの要求の処理をする runtime システム である。ユーザ定義の ORB クラスを分散アスペクトに記述することで、Addistant 2 標準の ORB とは異なる ORB を利用することができる。ただし、ユーザ定義の ORB クラスは RequestBroker インターフェイスを `implements` していなくてはならない。ORB クラスの指定が分散アスペクト内に記述されていないときは、Addistant 2 標準の ORB クラスを用いてシステムが実行される。

この機能を用いると例えば、システムの実行中にネットワークが切断されたときにも動作を続けられるよう、耐故障性を備えた ORB を利用することができる。また、変数 rmt で指定される複数のホストにオブジェクトのコピーを保持させる多重化 ORB を利用することもできる。

3.4 ネットワーク・ストリームの実装

分散アスペクトの ORB と同様、ネットワーク・ストリームの実装クラスもユーザ定義のクラスで拡張することができる。この場合、ユーザが定義したクラスは CommChannel クラスを `extends` しなくてはならない。

分散用アスペクト言語で、ネットワーク・ストリームの実装の指定方法を記述する例を以下に示す。

```
<channel classname="SSLCommChannel"/>
```

これは通信チャンネルを暗号化させるクラスにより、ネットワーク・ストリームをすべて暗号化するよう

に拡張できる。

4 応用例

ここでは、既存の非分散 GUI アプリケーションを Addistant 2 を用いて分散化する例を示す。まず、詳細にオブジェクトの配置を指定する例を示す。その後に、Addistant 2 標準の ORB をユーザ定義の多重化 ORB クラスを用いて差し換える例を示す。

4.1 オブジェクト配置の指定

例えば、異なるホストを利用する 2 人に同じ画面を見せるアプリケーションを考える。このアプリケーションの実装としては、プログラム内に 2 つの JFrame オブジェクトを生成する必要があるだろう。さらに、具体的に実装を考えると、createViewerFrame() メソッド内で JFrame のオブジェクト viewerFrame を、createEditorFrame() メソッド内で JFrame のオブジェクト editorFrame をそれぞれ生成し、表示させるようなプログラムになるだろう。Addistant 2 はこの既存プログラムを、次のように機能分散させなければならない。

> 2 つのオブジェクトのうち viewerFrame は手元のホストで生成し、手元のディスプレイに表示し、editorFrame は遠隔のホストで生成し、遠隔のディスプレイに表示させる。

以下に、既存プログラムを分散化するための分散アスペクト記述を示す。

```
<import proxy="subclass"
        in="FrameClient.createViewerFrame()"
        from="rmt">
    <subclass name="javax.swing.JFrame"/>
</import>
<import proxy="subclass"
        in="FrameClient.createEditorFrame()"
        from="app">
    <subclass name="javax.swing.JFrame"/>
</import>
<start name="app">
    <host name="picard"
          rmiPort="14003"
          bytecodePort="14004"
          searchDir="apprun"/>
</start>
<remote name="rmt">
    <host name="taro"
          rmiPort="14005"
          bytecodePort="14006"
          searchDir="rmtrun"/>
</remote>
```

この記述を Addsitant 2 に与えると、既存プログラムは 3.1 節で述べたように、viewerFrame オブジェクトは変数 rmt で指定されるホストに、editorFrame オブジェクトは変数 app で指定されるホストに配置されるようになる。

4.2 ORB クラスの拡張

Addistant 2 標準の ORB をユーザ定義の多重化 ORB クラスで差し換える機能は、様々な用途がある。例えば、先生が自分のコンピュータのディスプレイ上のウインドウを、生徒のコンピュータのディスプレイ上にも表示させ、その場でウインドウ内の記述を編集する、等である。ここでは、その用途の 1 つをあげる。

今、既存の非分散 Tic-Tac-Toe（マルバツゲーム）を、異なるホストを使う 2 人が対戦して遊びたいとする。

このアプリケーションの実装には、それぞれのホスト上に同じ画面を表示する JFrame オブジェクトを生成する必要がある。このためには、片方の JFrame オブジェクトのメソッドが呼ばれたときには、他方のメソッドも同じ引数で呼び出してやり、2 つのオブジェクトの内部状態が常に同じになるようにすればよい。このような遠隔メソッド呼び出しを多重化する ORB を我々はユーザ定義の ORB として実装した。ただし、既存のプログラムは一手ごとに 2 人が交互にポインティングデバイスを使うという暗黙の紳士協定があり、このルールを、Addistant 2 による分散化で自動的に付加させることは困難である。

この ORB を使って、Tic Tac Toe を分散化するには、次のような分散アスペクトを記述すればよい。

```
<import proxy="subclass"
        in="TTTStarter"
```

```
        from="app">
    <subclass name="javax.swing.JFrame"/>
</import>
<import proxy="subclass"
        in="EventHandler"
        from="rmt">
    <subclass name="javax.swing.JFrame"/>
</import>
<orb classname="RequestMultiplyingBroker"/>
<start name="app">
    <host name="picard"
          rmiPort="14003"
          bytecodePort="14004"
          searchDir="apprun"/>
</start>
<remote name="rmt">
    <host name="taro"
          rmiPort="14005"
          bytecodePort="14006"
          searchDir="rmtrun"/>
    <host name="yulian"
          rmiPort="14007"
          bytecodePort="14008"
          searchDir="rmtrun2"/>
</remote>
```

この記述を Addistant 2 に与えると、ORB としてユーザ定義のクラス RequestMultiplyingBroker が用いられる。

変数 rmt には複数のホストが指定されているが、このホストは次の 2 種類に分かれる。

- 変数 app で指定されるホストと直接データのやり取りをする、ただ 1 つのホスト。このホストは app で指定されるホストからの遠隔メソッド呼び出しを、オブジェクトのコピーをもつ他のホストへ、多重化して配送する役目をもつ。このホストを「遠隔リーダーホスト」と呼ぶ。

- 「遠隔リーダーホスト」から多重化された遠隔メソッド呼び出しを受け取って処理するホスト。

現在の実装では、変数 rmt に指定された最初のホストを「遠隔リーダーホスト」とみなす。上の例の場合、taro が遠隔リーダーホストである。

5 関連研究

これまでの分散処理プログラムの開発ツールは、プログラムを書く際に、初めから一定のスタイルに沿って書かなければならないものがほとんどであった。例えば、Java 標準 API に組み込まれている JavaRMI [4] では、遠隔オブジェクトは全てインタフェース型を通じて参照されなければならないという制約がある。この制約に沿ってプログラムを書かないと、JavaRMI が提供するスタブ生成コンパイラと ORB ライブラリを利用することができない。したがって、Addistant と異なり、初め分散を意識しないで書かれたプログラムを、後から分散対応にするような目的には JavaRMI は適さない。

D 言語 [5] もまた同様の問題がある。D 言語は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り、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扱う Interface Definition Language（IDL）を提供する。しかし、分散化の指示を、プログラム内に直接、修飾子の形で埋め込むため、分散を意識しないで書かれたプログラムを、後から分散対応にする場合には、元のソースファイルをあちこち修正しなければならない。とくに、Addistant と異なり、元のソースファイルが手に入らなければ分散対応にできないという問題がある。

AspectJ [2] は、Java 言語用の汎用的 AOP 言語である。AspectJ では、元のソースファイルからは独立したアスペクト記述を書くだけで、オブジェクトの挙動を変えるコードを *join point* に自動的に挿入することが可能である。これにより、ロギングやトランザクション管理を、元のソースファイルを修正せずに、アスペクト記述を書くだけで実装することができる。AspectJ と Addistant との違いは、前者が汎用的 AOP 言語であるのに対し、後者は分散処理に特化した AOP 言語であることである。AspectJ が提供する *join point* は、言語の基本機能に関係するものだけなので、AspectJ だけを使って既存プログラムを分散対応に拡張しようとすると、それは非常に困難になるだろう。

6 まとめ

本稿では、複数の 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分散用アスペクト指向言語、Addistant 2 を提案した。Addistant 2 の利用者は、分散に関する記述を、分散とは無関係な Java プログラムから、分離してまとめることができる。従来、分散環境を変更する際にはプログラム全体にわたって修正をおこなわなければならなかった。一方、Addistant 2 を使えば、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分散環境を変更できるようになった。Addistant 2 では、*join point* の種類を増やすことにより、Addistant 1 の分散アスペクトの記述力を強化することができた。これにより、Addistant 2 の利用者は、オブジェクトの生成文脈に従って、オブジェクトがどこに配置され、それらのオブジェクトの遠隔参照がどのように実装されているかを指定するファイルを与えるだけでよい。その他にも、分散アスペクトに記述するだけで、ユーザ定義の ORB クラスやネットワーク・ストリームの実装クラスを拡張したり、アプリケーションの実行環境を変更したりすることができる。Addistant 2 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、同一クラスから生成された GUI オブジェクトが複数のホスト上で動作する例を示した。

Addistant 2 は Addistant 1 のコードを再利用せずに現在開発中である。現在までのところ、Addistant 1 相当の部分がほぼ完成している。ただし、遠隔オブジェクト参照は「サブクラス」法しか指定できない。また、Addistant 2 での拡張はユーザ定義の ORB を指定する機能が完成している。

参考文献

[1] M. Tatsubori,T. Sasaki,S. Chiba, and K. Itano, A Bytecode Translator for Distributed Execution of "Legacy" Java Software, In *Proccedings of ECOOP 2001, LNCS 2072*, Springer, pp.313–336, 2001.

[2] G. Kiczales, J. Lamping,C. Maeda, Aspect-Oriented Programming, In *Procedings of ECOOP 1997, LNCS 1241*, June, 1997.

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

[4] J. Farley, JAVA Distributed Computing, pp.47-79.

[5] N. Nagaratnam, A. Srinivasan, and D. Lea, Remote Objects in Java, In *IASTED '96, International Conference on Networks*, 1996.

[6] E. Gamma, R.Helm, R. Johnson, and J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley 1994.

[7] H. Rohnert, The Proxy Design Pattern Revisited, *Pattern Languages of Program Design 2*, Addison-Wesley, chapter 7, pp.105-118, 1995.

[8] A. Black, N. Hutchinson, E. Jul, H. Vevy, and L. Carter, "Distribution and Abstract Types in Emerald." *IEEE Transactions on Software Engineering* SE-13, no. 1, 1987.

[9] Galen C. Hunt, Michael L. Scott, "The Coign Automatic Distributed Partitioning System" *Operating Systems Design and Implementation*, pp.187-200, 1999.

[10] Sun Microsystems, Java Technology and XML, http://java.sun.com/xml/docs.html.