5M－3

An Implementation of Reflective Concurrent Component Architecture

中村 顕朗　児玉 靖司　武田 正之*
東京理科大学大学院理工学研究科情報科学専攻†

1 はじめに

Java言語は柔軟性のあるオブジェクト指向言語であり、簡単にさまざまなアプリケーションの記述が出来る。また並行処理のためのライブラリを提供している。しかし並行プログラミングにおいて、デッドロック等の問題があり、これらの問題を考慮した記述は非常に複雑である。そこで本研究では、並行プログラミングのためのリフレクティブアーキテクチャReCCA(Reflective Concurrent Component Architecture)をJava言語で実装した。これによって並行処理を必要とするアプリケーションをよりわかりやすく記述する事ができる。Java言語によるリフレクティブアーキテクチャは[1]によって提案されているが、本稿では、新たに`MetaObject`クラスを導入する事のみで、リフレクティブな機能を提供している。

2 リフレクション

ReCCAでは個々のオブジェクトxにメタレベルが存在する。メタオブジェクト$\uparrow x$を生成するために、クラス`MetaObject`を提供する。`MetaObject`の機能の提供を受ける場合は、Java言語の仕様が単一継承であり、既存のライブラリを継承したクラスにもリフレクティブ機能を提供したいため、クラス`MetaObject`のインスタンスを生成し、そのメソッドを呼び出す(図1)。あるオブジェクトに対する、メタオブジェクトを生成するためには、以下のようにベースレベルのオブジェクトが生成した`MetaObject`インスタンスを保持する。

```
MetaObject m = new MetaObject(this);
```

各メタオブジェクトは`MetaObject`のインスタンスを保持し`MetaObject`の機能を拡張できる形をとる。この`MetaObject`インスタンスを保持するメタオブジェクトに関して、さらに`MetaObject`インスタンスを生成する。これを繰り返す事によりリフレクティブタワーを形成する(図2)。

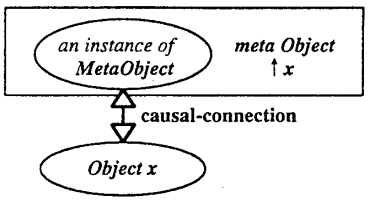

図1: `MetaObject`のインスタンス

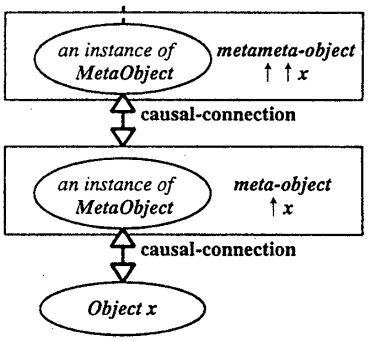

図2: リフレクティブタワー

3 MetaObjectの構成

`MetaObject`はメソッドリスト、フィールドリスト、スレッド、イベントキューで構成される(図3)。これらはクラス`MetaObject`の用意したメソッドを介して操作される。メソッドリスト、フィールドリストはオブジェクトのメソッド、フィールド情報が含まれている。これらを操作することによって、メソッド更新(追加・削除)や動的継承を行う。Javaのクラスの継承関係は静的に決定されるが、`MetaObject`により任意のオブジェクトのメタオブジェクトを操作することによって、動的に継承関係を変更することができる。

ベースレベルのメソッドコールは`MetaObject`の`mcall`、`amcall`メソッドを通じて行う。`mcall`は普通のメソッド呼びだしと、同期をとる(Javaでいう`synchronized`)呼び出しを指定する事ができる。`amcall`は非同期

*Akio Nakamura, Yasushi Kodama and Masayuki Takeda
†Dept. of Information Sciences, Science University of Tokyo

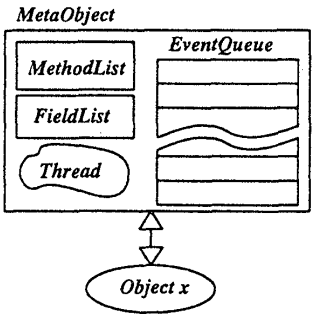

図 3: MetaObject の構成

呼び出しである。mcall(amcall) は引数にコールしたいメソッド名と、そのメソッドの引数を含む Object の配列が入る。mcall で呼ばれたメソッドは、メソッドリストを参照して、指定されたメソッドを見つけてそれを実行する。フィールドへのアクセスの場合もフィールドリストを用いて同じ要領で行われる。同期のための機能として、Java で言う synchronized ブロックに相当するメソッドも提供している。図 4 ではクラス X についての例を示している。ここにある MetaX クラスは MetaObject で提供されているメソッド等を使って、さらにメタレベルの機能を拡張する場合に、ユーザが定義するメタオブジェクトである。メタオブジェクトは 1 つのベースオブジェクトに対して 1 つであり、MetaObject はその生成元である、オブジェクト自身 (this) の中で作られる。

```
class X{
    MetaObject mo;
    X(){
      mo = new MetaObject(this,true,true);
           ←オブジェクトXのMetaObjectインスタンス
      MetaX mx = new MetaX(mo);  ←ユーザにより拡張
    }
      mo.mcall("methodA",null);  ←メソッドコール
    methodA(){…}  ←----------
}                  メソッド定義
```

図 4: class X 記述例

スレッドはベースレベルのオブジェクトが Runnable を実装している場合に生成するスレッドや、呼び出すメソッドをスレッド化したものである。スレッドはメタレベルで操作する。

イベントキューはオブジェクトに対して、メッセージを受け付ける。排他制御指定の mcall(amcall) により、呼びだされたメソッドのイベントは呼び出し先のメタオブジェクトのイベントキューに貯められる。これらイベントはキューに入った順番で実行される。しかし、メタメタレベルでイベントキューを直接操作する事により、イベントの実行の順番、優先度、受け付けるイベントを動的に変更できる。また、どのオブジェクトに対してスレッドがどのメソッドにアクセスしたのかを記録するカウンターがありメソッドがコールされる度に記録される。イベントキューとスレッドは MetaObject 生成時、コンストラクタで使用するかしないかを指定できる。

4 実装・評価

ReCCA を用いて「哲学者の食卓問題」を記述した。フォークと哲学者がそれぞれ各 5 つずつオブジェクトがあり、哲学者はそれぞれのフォークのやり取りをする。哲学者もフォークもそれぞれリフレクティブなオブジェクトである。哲学者は各々フォークの取り方に制約は無く、先に空いているフォークを取りに行く。

このとき、デッドロックとスタベーションの問題があるが、哲学者とフォークオブジェクトのメタメタレベルにおいて、これらを監視するオブジェクトを用いてこの問題を解決する方法と、哲学者、フォーク各々が両隣のオブジェクトの参照のみで、監視するオブジェクトを用いない方法で問題を解決する方法を記述することができた。

5 おわりに

このアーキテクチャを用いて、哲学者の食卓問題のような問題を様々な形で簡単に記述できた。またこのアーキテクチャ自体が、既存の JavaVirtualMachine 上で実現されているため、JavaBeans 等への応用が考えられる。今後はこのアーキテクチャを使ってアプリケーションを実装しながら、検証していくことが必要である。

参考文献

[1] 立堀道昭, 千葉滋, 中田育男: Java のための新たな自己反映機構の提案 日本ソフトウェア科学会第 14 回大会論文集,1997

[2] 児玉靖司, 葛西毅郎: リフレクティブアーキテクチャによる仮想空間の実現 ワークショップ システムソフトウェアとオペレーティングシステム 情報処理学会, 1998