5L-6

マルチスレッドに関するデザインパターン検出

樋口　翔　　深海　悟
大阪工業大学大学院情報科学研究科

1.はじめに

デザインパターンとは，オブジェクト指向ソフトウェア設計にあらわれる典型的な問題に対する解法をまとめたものである。デザインパターンを利用することによって設計の品質は向上する[1]。また，既存のソフトウェアからデザインパターンを検出することで，適切な拡張方法や修正方法を見つけやすくなることが期待される。このようなことから，多くのデザインパターンの検出手法が研究され提案されている[5]。これらの研究は，GoF デザインパターンと呼ばれる有名なデザインパターンを対象としている。

一方，マルチスレッドプログラムに関するデザインパターンなど，GoF デザインパターンではないデザインパターンも多く存在する。マルチスレッドプログラムはシングルスレッドでは考えられないような問題が存在し，安全性，生存性，再利用性，パフォーマンスに考慮して設計する必要がある。

以上のことから，既存研究の GoF デザインパターン検出と同様に，マルチスレッドに関するデザインパターンの検出も重要な課題と言える。

本研究では，GoF デザインパターンではない，マルチスレッドプログラムに関するデザインパターン検出手法を提案・実装し，既存の Java プログラムに対する検出実験によりその正しさを確認した。

2.マルチスレッドプログラムの評価基準

マルチスレッドプログラムを設計する際には表 1 のような評価基準を考慮する必要がある[2]。

表1　マルチスレッドプログラムの評価基準

評価基準	重要度	内容
安全性	必須	インスタンスが壊れない
生存性	必須	必要な処理が必ず行われる
再利用性	任意	排他制御の隠蔽など
パフォーマンス	任意	スループットや応答性

3.マルチスレッドに関するデザインパターン

本研究で検出するデザインパターン[2]について紹介する。

3.1 SingleThreadedExecution パターン

文脈:複数のスレッドがインスタンスを共有している。
問題:複数のスレッドがインスタンスを勝手に変更すれば安全性が失われる。
解決法:クラス図と概念図を図 1，2 に示す。インスタンスの安全性が失われてしまう範囲を定め，その範囲がひとつのスレッドだけで実行されるようにする。これにより安全性が確保される。

Java による実装:
- フィールドの可視性は private
- フィールドへのアクセスを syncronized 内に限定
- private メソッドからのフィールドへのアクセスの場合，このメソッドを呼び出しているすべてのメソッドが syncronized 内にある

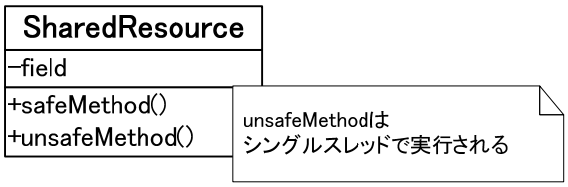

図 1: SingleThreadedExecution パターンのクラス図

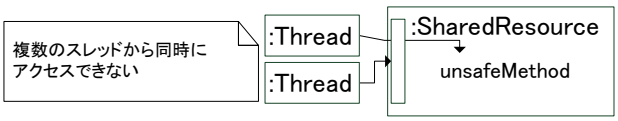

図 2: SingleThreadedExecution パターンの概念図

3.2 GuardedSuspension パターン

文脈:複数のスレッドがインスタンスを共有している。
問題:複数のスレッドがインスタンスを勝手に変更すれば安全性が失われる。
解決法:クラス図と概念図を図 3，4 に示す。インスタンスの状態が不適切なときには，適切な状態（ガード条件）になるまでスレッドを待たせることで安全性が保たれる。

Java による実装:
- guardedMethod と stateChangingMethod に syncronized 修飾子をつける
- guardedMethod でガード条件を満たすまでループ内で wait メソッド呼び出し
- stateChangingMethod で state を変更，notifyAll メソッド呼び出し

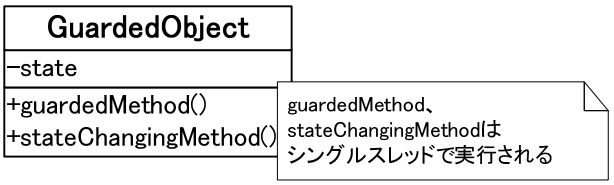

図 3: GuardedSuspension パターンのクラス図

Design Pattern Detection For Java Multithread Program
Higuchi Sho, Fukami Satoru
Osaka Institute of Technology

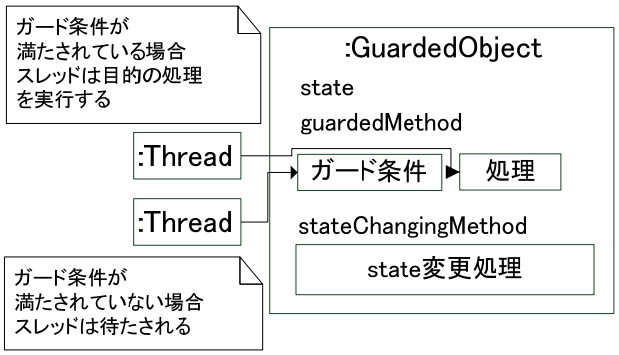

図 4: GuardedSuspension パターンの概念図

4.提案手法

本システムの入力は Java の class ファイルである。class ファイルから表 2 に示す情報を抽出する。

抽出された情報をもとに，3 節で示した各デザインパターンの Java の実装と照らし合わせ検出する。

本稿では，SingleThreadedExecution パターンと GuardedSuspension パターンの検出についてのみ述べるが，Immutable パターンも検出手法は同様である。

表 2　class ファイルから抽出する情報

クラスに関する情報		フィールドに関する情報	
継承関係	メソッド	修飾子	可視性
可視性	フィールド	フィールドの型	フィールド名
メソッドに関する情報			
メソッド名	戻り値	修飾子	メソッド呼び出し
引数	可視性	ループ処理	フィールドアクセス

4.1 SingleThreadedExecution パターンの検出

SingleThreadedExecution パターンのクラスとそれにより守られているフィールドを検出する。

検出のポイントは，守られているフィールドがあり，そのフィールドにアクセスする場合常に syncronized 内であるということである。図 5 に検出の流れを示す。

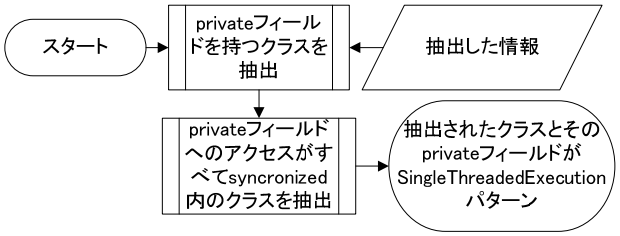

図 5: SingleThreadedExecution パターン検出の流れ

なお，syncronized でない privavte メソッドから private フィールドへアクセスがある場合は，private メソッドの呼び出し元のすべてが syncronized 内である必要がある。

4.2 GuardedSuspension パターンの検出

GuardedSuspension パターンのクラスを検出する際のポイントは，以下の３つである。

・インスタンスの状態が適正な状態になるまで目的の処理を実行させないパターンであること。

・目的の処理の前に適正な状態かを判定する繰り返し文が存在し，その中でスレッドをウェイトセットの中に入れる wait メソッドが存在すること。これによって，適正な状態になるまでスレッドは目的の処理を実行できない。

・インスタンスの状態を変更するメソッドにはウェイトセットからスレッドを取り出す notifyAll メソッド呼び出しが存在すること。

図 6 に検出の流れを示す。

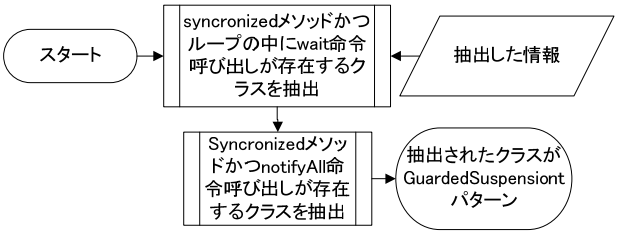

図 6: GuardedSuspension パターン検出の流れ

5.実験

JUnit[3]および log4j[4]に対して検出実験を行った。結果を表 3 に示す。

表 3　検出実験の結果

プログラム	クラス数	SingleThreaded Execution	Immutable	Guarded Suspension
Junit	246	1 箇所	4 箇所	1 箇所
log4j	308	5 箇所	7 箇所	0 箇所

実際に検出された箇所を確認したところ，すべてデザインパターンであることを確認することができた。

しかし,今回の検出手法では，syncronized 修飾子ではなく syncronized ブロックを使った場合など，変形されたデザインパターンは検出できていない。

6.おわりに

本研究では，マルチスレッドプログラムに関するデザインパターン検出手法を提案・実装し，既存の Java プログラムに対して検出実験を行った。その結果，提案手法によってマルチスレッドプログラムに関するデザインパターンを検出することが確認できた。

今後の課題として，動的な解析手法を導入し，変形されたデザインパターンに対して検出する必要がある。

参考文献

[1]Erich Gamma,et.al: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley (1995).

[2]結城 浩: 増補改訂版 Java 言語で学ぶデザインパターン入門 マルチスレッド編, ソフトバンククリエイティブ (2006).

[3]JUnit.org Resources for Test Driven Development http://www.junit.org/

[4]Apache Logging Services log4j http://logging.apache.org/

[5]水野恵祐:デザインパターン検出能力の向上を目的とした複数検出手法の併用, 奈良先端科学技術大学院大学情報科学研究科情報システム学専攻修士論文 (2010).