1S-7

オブジェクト同期体デザインパターンの多重ロックへの拡張

海老澤 覚　吉田 紀彦
埼玉大学大学院理工学研究科情報システム工学科専攻

1　はじめに

分散処理デザインパターン [1] の一つであるオブジェクト同期体パターン [2][3] では，複数の同期化オブジェクトを同時に操作するための機構が無く，そのような操作を行う時にデッドロックが生じる恐れがある．そこで本研究では，既存のオブジェクト同期体パターンの設計に，複数の同期化オブジェクトの同期化処理および操作を集中的に管理する役割を追加してこの問題の解決を図った．

まずオブジェクト同期体パターンについて簡単な説明と，複数同期化オブジェクトの同時操作を行う際の問題点を述べた後，本研究で考案した拡張オブジェクト同期体パターンを紹介する．

2　オブジェクト同期体

オブジェクト同期体パターンは，クライアント／サーバ環境において複数クライアントおよび単一サーバで共有される同期化オブジェクトを操作する際に，同期を行うための処理をオブジェクトに追加するためのデザインパターンである．同期化のための処理とはオブジェクトのロックを取得する，オブジェクトの操作による変更を通知するというようなものを指す．

オブジェクト同期体パターンの構造を表すクラス図を図１に示す．

図１において，FunctionalObject は同期化を行いたいオブジェクトである．FunctionalObject には同期化のための機能はない．SynchronizationInterface は FunctionalObject の同期化を行うインターフェースで，クライアントはこれを通して FunctionalObject の操作 method() の呼び出しを行う．SynchronizationData は FunctionalObject を同期化するための大域的なデータを提供する．

Synchronizer と SynchronizationPredicate は同期化を行うための機能を提供する．Synchronizer は同期化

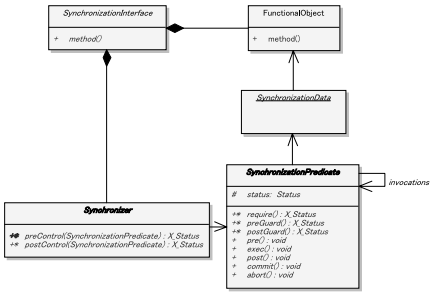

図 1: オブジェクト同期体パターンのクラス図

の一般的な動作（更新のタイミング，エラーハンドリングなど）を，SynchronizationPredicate は SynchronizationData を用いて個々の同期化オブジェクトが持つ性質に依存する動作（ロックの取得および解放，更新や変更の中止など）を与える．これらは SynchronizationInterface によって生成され，FunctionalObject が操作を行う前後に同期化のための処理を行うように指示される．

3　オブジェクト同期体パターンの問題点

オブジェクト同期体パターンでは，FunctionalObject と SynchronizationInterface は一対一で対応している．Synchronizer および SynchronizationPredicate オブジェクトは FunctionalObject オブジェクトに一対一で与えられる．もしここで同期化オブジェクトが複数で，それらが互いに関連しながら変更されるという状況を考える場合，既存の設計では問題が生じる．

既存の設計のままで同期化オブジェクトを複数扱う場合，もし同期化されたオブジェクトの操作中に他の同期化オブジェクトの操作を行うような設計にすると，デッドロックとなり得る．

また，複数の同期化オブジェクトが一斉にある操作を行い，その内一つが失敗した場合は全体で失敗したと見なし，全ての操作を中止するという状況を考える．この時は同期化の失敗を発見し，全ての同期化オブジェクトが行う更新を取りやめる必要があるが，更新を行うタイミングは具象 Synchronizer が管理しており，操

Extension of Object Synchronizer Pattern to Multiple Locking
Satoshi EBISAWA and Norihiko YOSHIDA
Saitama University

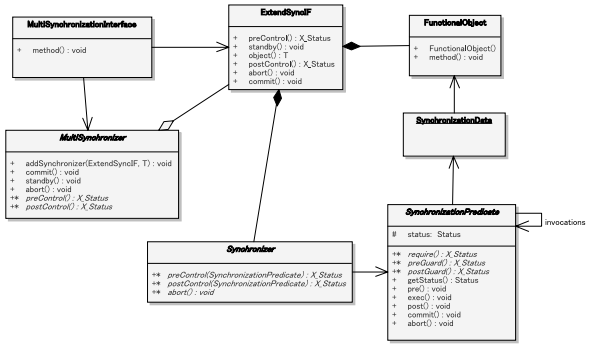

図 2: 拡張オブジェクト同期体パターンのクラス図

作の事後処理を行う中でロックの取得と連続して行う場合があり，ロックの取得や解放と切り離されていない．このため，一つ一つの同期化オブジェクトを順番に同期化処理するような場合でも，途中で失敗した場合にはそれまで正常に終了していた同期化オブジェクトの変更をロールバックするために同期化処理を行わなければならない．

4 拡張オブジェクト同期体パターン

先述した問題を解決するため，オブジェクト同期体パターンを拡張した．本研究で考案した拡張オブジェクト同期体パターンのクラス図を図２に示す．

FunctionalObject，SynchronizationData，SynchronizationPredicate は同期体パターンの同名クラスと同一の物である．

SynchronizationInterface があった部分を拡張して，MultiSynchronizationInterface，MultiSynchronizer，ExtendSyncIF (ExtendSynchronizationInterface) の３クラスを追加した．

ExtendSyncIF は各 FunctionalObject と，それを同期化するための Synchronizer，SynchronizationPredicate を保持し，管理する．外部からの命令によって管理する FunctionalObject に関する事前処理，FunctionalObject の提出，事後の同期化準備，同期化の反映と同期化終了処理，中止を行う．

MultiSynchronizer は行おうとする同期化操作に必要な同期化オブジェクト群を ExtendSyncIF 単位で管理する．具象 MultiSynchronizer が提供する複数ロックのポリシに従い，複数ロックの為のデータ T を基に同期化処理の順番を決定し，更新操作 commit() や中止操作 abort() を全ての ExtendSyncIF に対して行うよう指示する．

MultiSynchronizationInterface は MultiSynchronizer を用いて，同期化の事前制御，オブジェクトの提出，事後制御を行わせ，提出された FunctionalObject を用いて操作を行う．

5 使用例

例としてオブジェクトの獲得に優先順位を付ける場合を考える．この時各 FunctionalObject から int 型のデータを ExtendSyncIF と共に MultiSynchronizer に登録しておく．MultiSynchronizationInterface が操作の実行をするとき，まず MultiSynchronizer に対して事前処理を行うように要求する．MultiSynchronizer は優先順位の高い順にロックを取得する．次に MultiSynchronizationInterface は与えられた FunctionalObject を用いて操作を実行し，MultiSynchronizer に各オブジェクトの事後処理を行わせる．ここまでに問題がなければ MultiSynchronizationInterface は MultiSynchronizer に指示して，全てのオブジェクトの commit() を呼び出し更新させる．

もし事後処理までの段階でロック取得の失敗，不正な操作，その他の理由で中止せざるを得ない状況になった場合，abort() を実行することで全ての取得したロックの解放，操作の中止あるいは取り消しを行う．

6 まとめ

デッドロックを回避するためのロックの取得や更新などを画一的に行うため，新たに拡張オブジェクト同期体パターンを設計し，プログラム例を通じて有効性を確認した．現在は，オブジェクト復旧パターン [2] との協調や，同期化処理中における新たな同期化オブジェクトの追加を行う際の問題などについて検討を進めている．

参考文献

[1] E. Gamma et al.,（監訳：本位田 他）, **オブジェクト指向における再利用のためのデザインパターン**, ソフトバンクパブリッシング, *1995*

[2] PLoPD Editors（監訳：細谷 他）, **プログラムデザインのためのパターン言語**, ソフトバンクパブリッシング, *2001*

[3] A. R. Silva, et al., **“Object Synchronizer”,** *Pattern Languages of Program Design*, **Vol.4**, Addison-Wesley, pp.111–132,*1999*